
longturn-manuals Documentation

Longturn.net

Apr 17, 2024

CONTENTS

1	Getting	3
1.1	About Freeciv21	3
1.2	Installing Freeciv21	4
1.3	Windows Installation	5
1.4	Compiling Freeciv21	16
2	Playing	25
2.1	Frequently Asked Questions (FAQ)	25
2.2	How to Play Freeciv21	46
2.3	Citizen Governor (aka Citizen Management Agent, or CMA)	53
2.4	The Turn Change Sequence	58
3	Manuals	65
3.1	Game Manual	65
3.2	Server Manual	129
3.3	Advanced Game Topics	165
3.4	Program Manual Pages	171
3.5	Using the Modpack Installer Utility	194
3.6	Common Ruleset Reference Pages	196
4	Modding	205
4.1	Rulesets	205
4.2	Tilesets	252
4.3	Soundsets	277
4.4	Musicsets	278
4.5	Scenarios	280
4.6	Installer	281
5	Contributing	287
5.1	Filing Bugs and Enhancement Requests	287
5.2	Set up a Development Environment	288
5.3	Setting up MSYS2 for Windows	290
5.4	Visual Studio for Windows	292
5.5	How to Submit a Pull Request	297
5.6	Evaluating a Pull Request	300
5.7	The Release Process	302
5.8	Workarounds	303
5.9	Maintaining the Stable Branch	304
5.10	Documentation Style Guide	305
6	Coding	313

6.1	Architecture	313
6.2	Freeciv21 Hacker's Guide	314
6.3	Artificial Intelligence (AI)	323
6.4	Attribute Blocks	328
6.5	Coding Guidelines	328
6.6	Internationalization	334
6.7	Logging	336
6.8	Network Protocol	337
6.9	Format Description of the Scorelog	342
7	Glossary of Terms	343
	Index	345

Welcome to the documentation of the Freeciv21 project brought to you by the [Longturn.net](#) community. The documentation is organized in six categories:

Getting

The Getting category is where you to go learn about Freeciv21 and how to get it installed on your system.

Playing

The Playing category is an area for documentation editors to provide tips and tricks on ways to play Freeciv21 and any of they varying rulesets that the Longturn.net community likes to play. You will also find the FAQ here.

Manuals

The Manuals category is where we place all of our varying manuals such as the game manual, advanced user manual and ruleset manuals.

Modding

The Modding category is an area for documentation editors to provide tips and other details on modifying aspects of Freeciv21 such as Rulesets, Musicsets, Soundsets and Tilesets. All of these areas allow for a large amount of variability in game play that is not hardcoded in the software. This is one of the strengths of Freeciv21.

Contributing

The Contributing category is where documentation editors provide instructions on how to contribute to the Freeciv21 project.

Coding

The Coding category is an area for documentation editors to provide information on how to alter the source code of Freeciv21.

GETTING

The Getting category is where you to go learn about Freeciv21 and how to get it installed on your system.

1.1 About Freeciv21



Fig. 1.1: Freeciv21 Overview

Freeciv21 is a free open source turn-based empire-building 4x strategy game, in which each player becomes the leader of a civilization. You compete against several opponents to build cities and use them to support a military and an economy. Players strive to complete an empire that survives all encounters with its neighbors to emerge victorious. Play begins at the dawn of history in 4,000 BCE.

Freeciv21 takes its roots in the well-known FOSS game Freeciv and extends it for more fun, with a revived focus on competitive multiplayer environments. Players can choose from over 500 nations and can play against the computer or other people in an active online community.

The code is maintained by the team over at Longturn.net and is based on the QT framework. The game supports both hex and square tiles and is easily modified to create custom rules.

Achieving success requires a balance between economic expansion, military strength, and technological development. Not only must you develop all three in concert to both expand and successfully defend your empire, but any of these four items may provide victory over your opponents:

- As in other games of conquest and expansion, you are declared the winner by default once the last city and unit of every other civilization is destroyed.
- Once technological progress has brought you into the space age, you may launch a spaceship destined for Alpha Centauri. The first civilization whose spaceship reaches the system wins.
- In the absence of other means to determine victory, the game will end after 5,000 turns if no spaceships have been launched. The surviving civilizations are then rated, and the one with the highest score is the winner.
- If playing a multiplayer game with only humans, victory can be declared based on diplomatic factors and other pre-set game rules.

Note: Spacerace Victory is set by the ruleset and thereby may not be not enabled for all games.

1.2 Installing Freeciv21

The developers of Freeciv21 provide pre-compiled binaries and installation packages for tagged releases. They can be found on the Longturn GitHub Repository for Freeciv21 at <https://github.com/longturn/freeciv21/releases>. The Longturn community provides binary packages for Debian-based Linux distros (Debian, Ubuntu, Mint, etc.), Microsoft Windows® (32 and 64-bit), and Apple macOS®. If you are an Arch Linux user, you can find Freeciv21 in the AUR at <https://aur.archlinux.org/packages/freeciv21>.

1.2.1 Windows

For more information on using the Windows Installer package, you can read about it at [Windows Installation](#).

1.2.2 Debian Linux

To install the Debian package, use the apt command with elevated privileges like this:

```
$ sudo apt install ./freeciv21_*_amd64.deb
```

1.2.3 macOS

To install the macOS .dmg package, you start by double-clicking the file to mount it in Finder. Drag the game to your Applications folder, or a place of your choosing. When finished, unmount the package.

1.3 Windows Installation

The Freeciv21 community provides a Microsoft Windows® installer when new releases are tagged. The following is instructions for installing on Windows.

To obtain the installer, go to the Freeciv21 [Releases](#) page. Each release has an assets section and the Windows installer and its hash are included. Click to download and double-click the file to run.

1.3.1 Local Admin Installation

The following steps assume you *are* logged on as a local Administrator. If you *are not* then jump to the *non-admin* section below.

If you *are* logged on as a local Administrator, a Windows User Access Control (UAC) window will appear and ask: “Do you want to allow this app from an unknown publisher to make changes to your device”. Click *Yes* and the welcome panel will load after an uncompress process has completed.

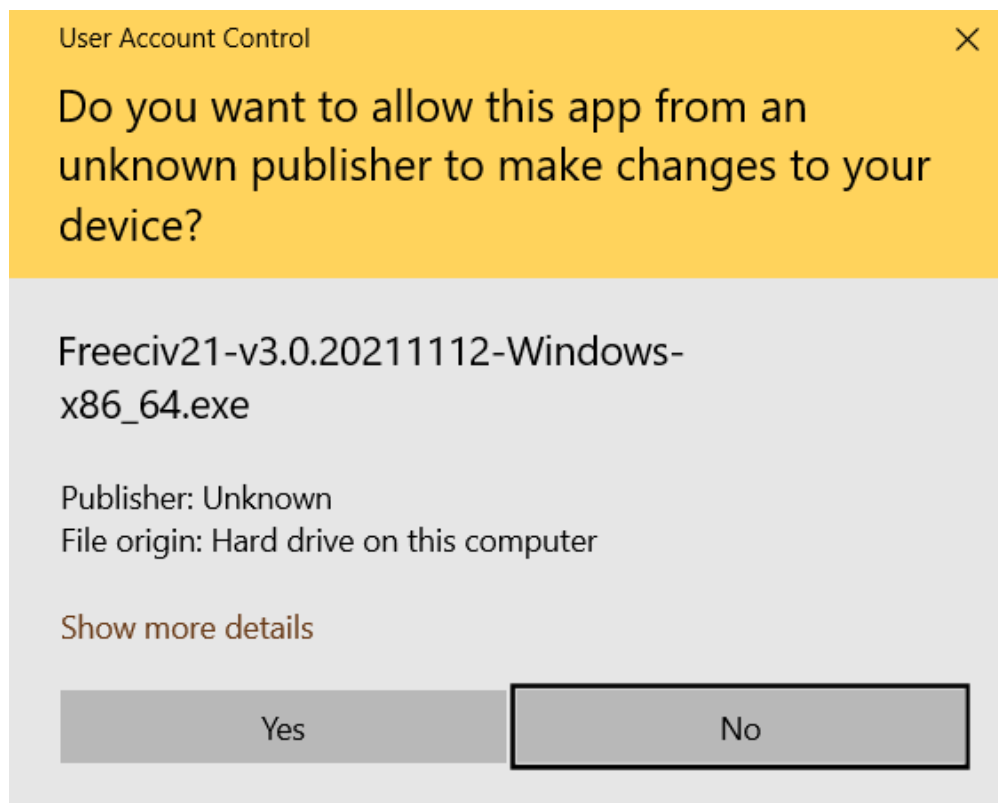


Fig. 1.2: Admin Install - Windows Defender

Click *Next* to continue...

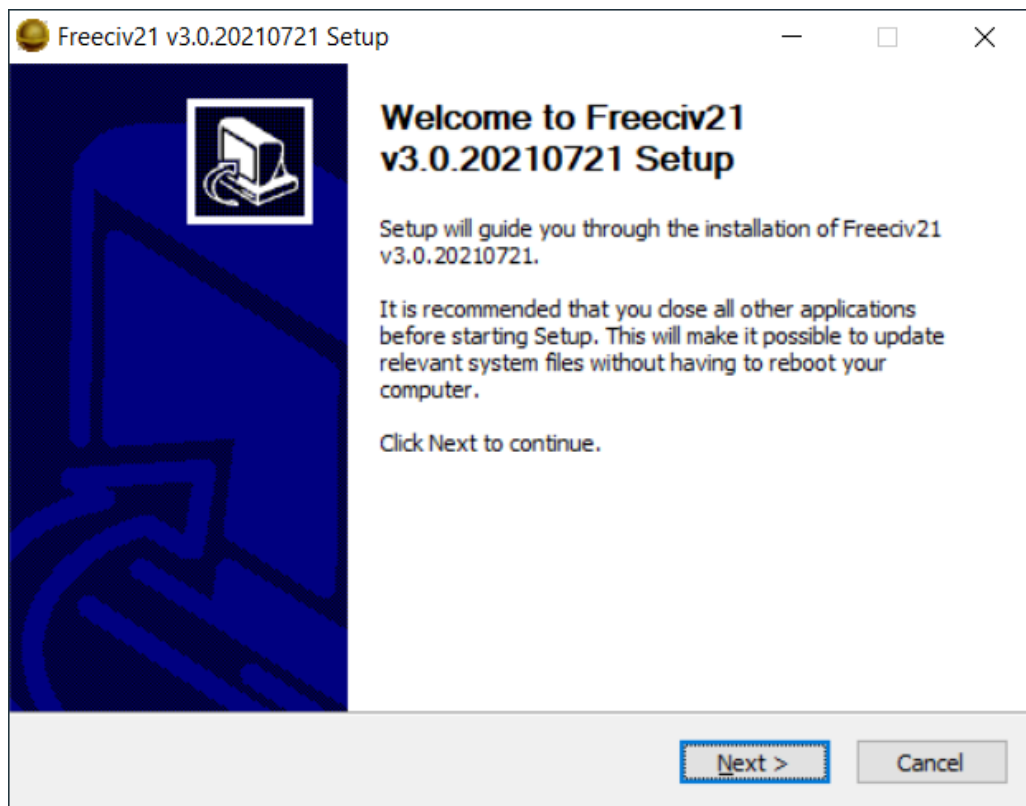


Fig. 1.3: Admin Install - Welcome Page

By default, the installer only installs the client, server and modpack installer. If you want other utilities, you will need to pick them from the list. Click *Next* to continue...

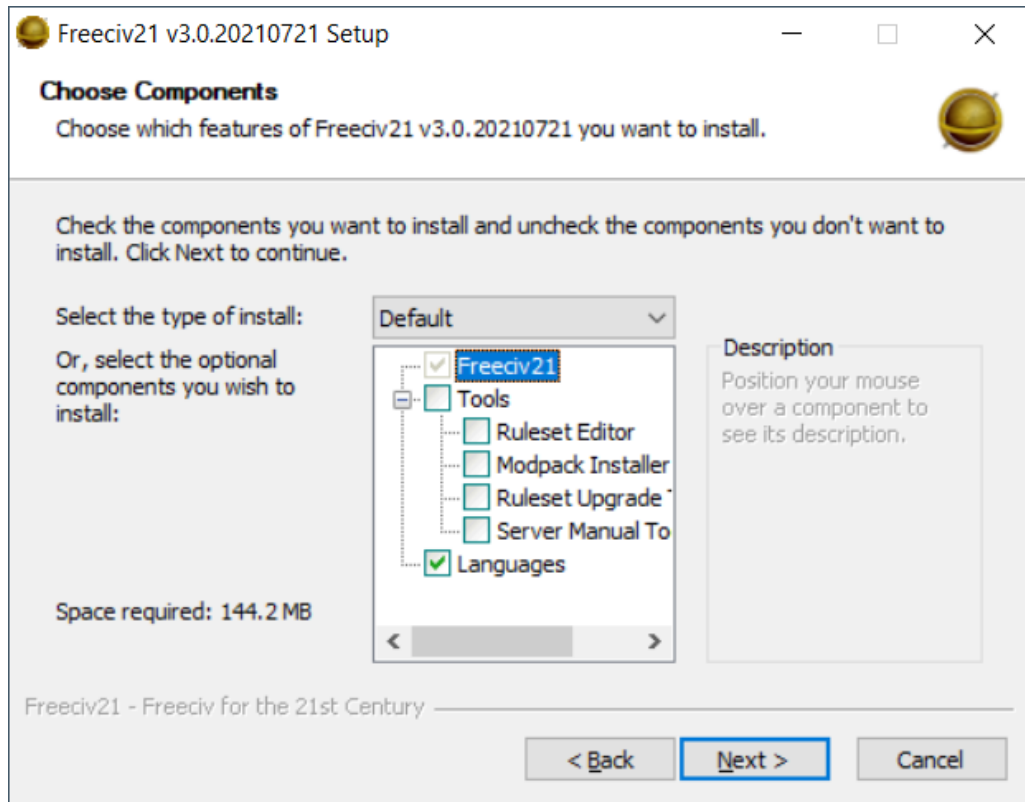


Fig. 1.4: Admin Install - Select Components

Select the installation type and then click *Next* to continue...

By default, the Admin installer will place the files in the C:\Program Files folder. Change if you want and click *Next* to continue...

This panel shows the files being installed. No action is needed.

If you want to run Freeciv21 right away, leave the box checked and click *Finish*. Otherwise, uncheck the box before clicking *Finish* to complete the installation.

You will find an icon for the client on the Desktop or in the Start Menu at *Start Menu* → *Freeciv21* → *Freeciv21 Client*.

1.3.2 Non-Local Admin (Unprivileged User) Installation

If you *are not* logged on as a local Administrator there is no UAC prompt and the following panel will load after an uncompress process has completed.

Click *Next* to continue...

By default, the installer only installs the client, server and modpack installer. If you want other utilities, you will need to pick them from the list. Click *Next* to continue...

The Non-Admin Installer places the files in your User Directory by default. Change if you want and click *Next* to continue...

This panel shows the files being installed. No action is needed.

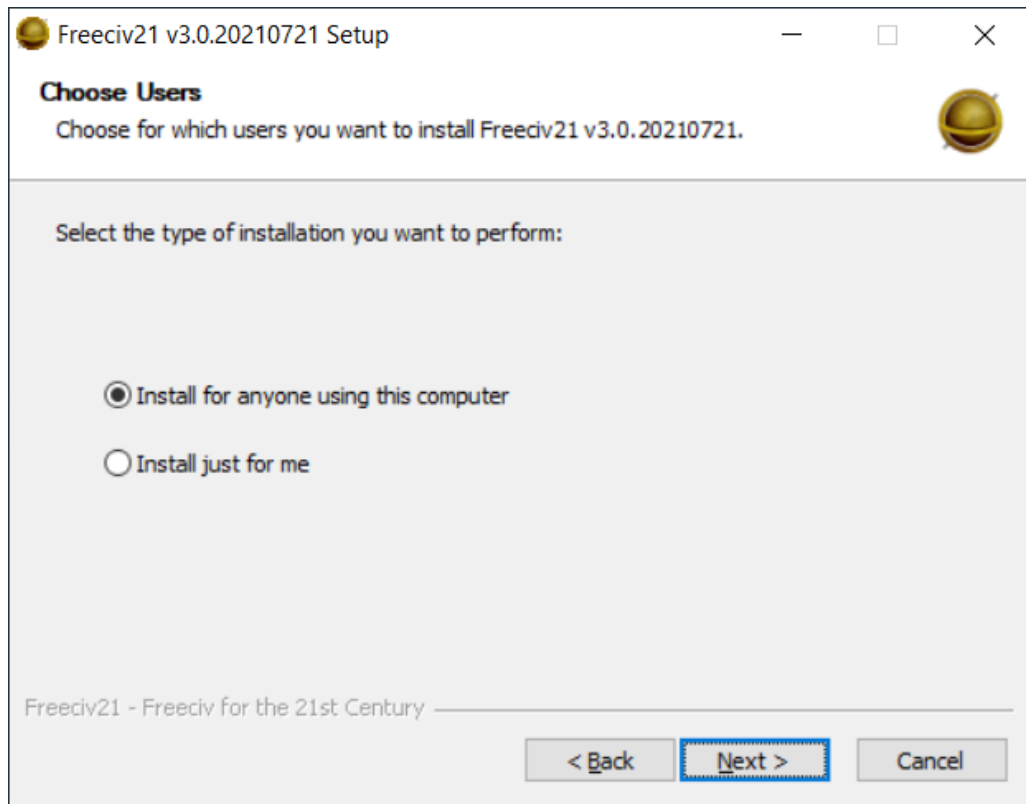


Fig. 1.5: Admin Install - Multi-User Selection

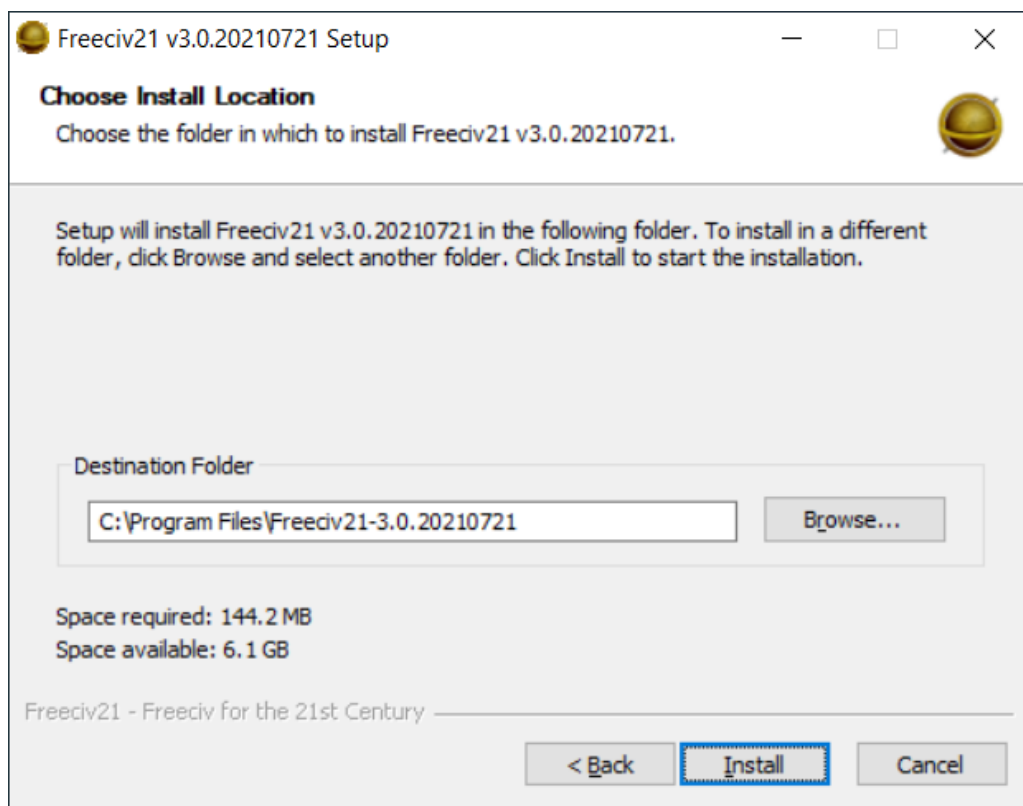


Fig. 1.6: Admin Install - Installation Folder

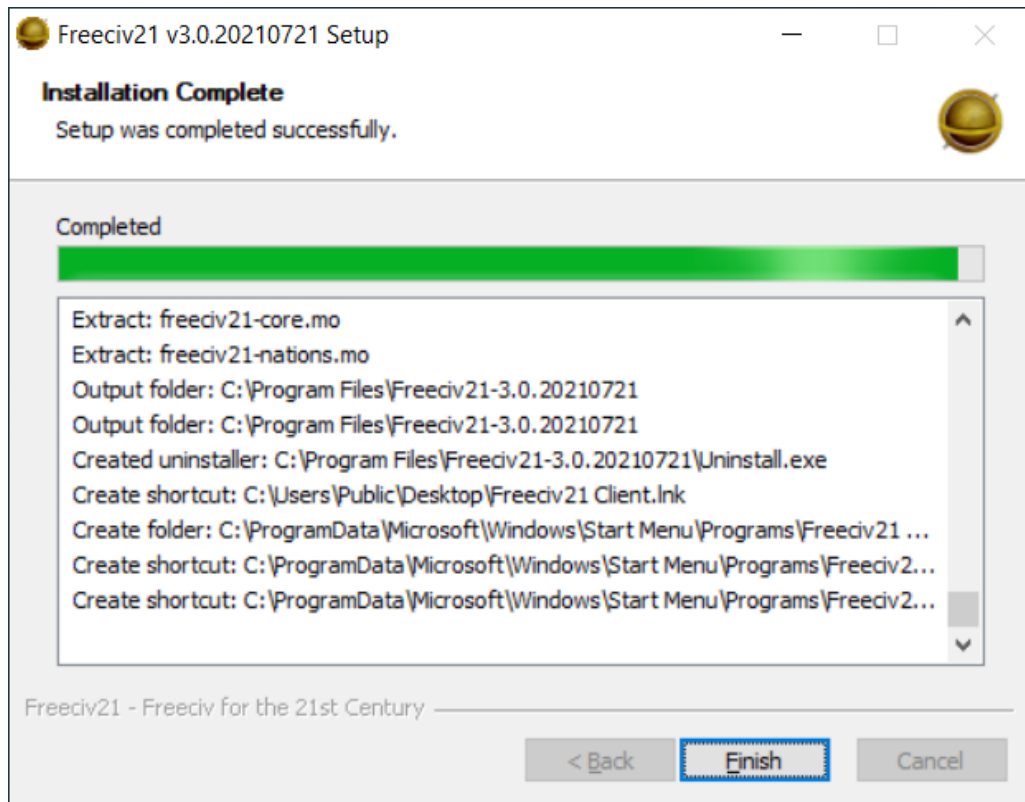


Fig. 1.7: Admin Install - Copy Files

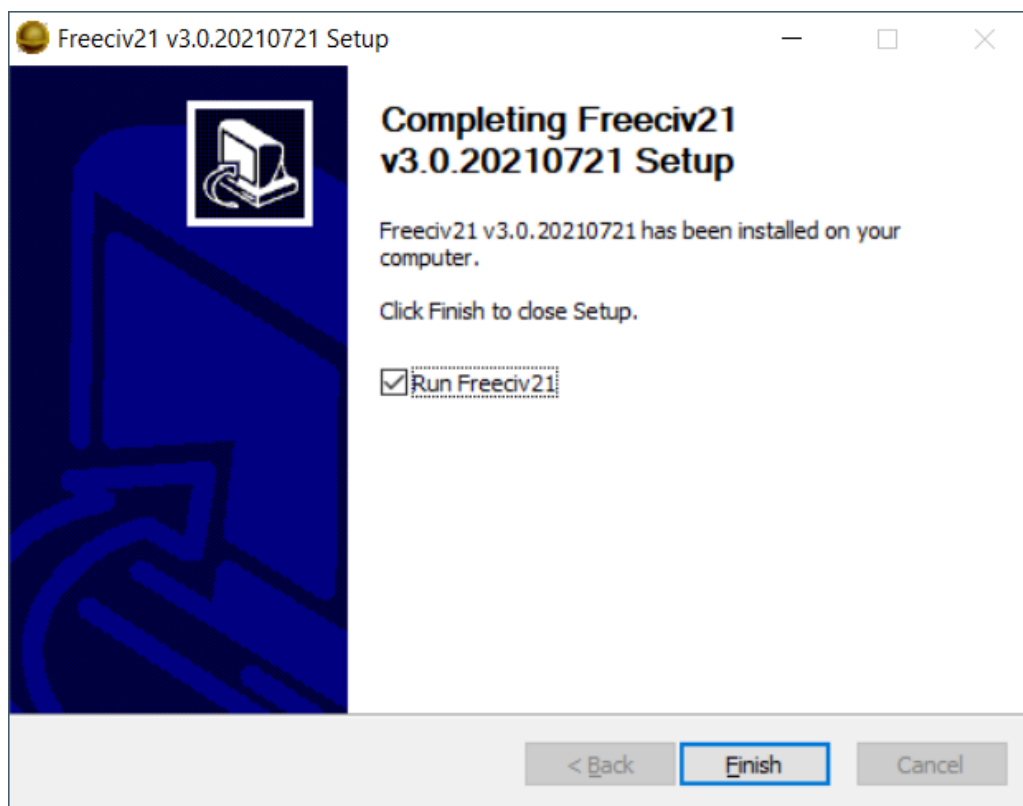


Fig. 1.8: Admin Install - Finish

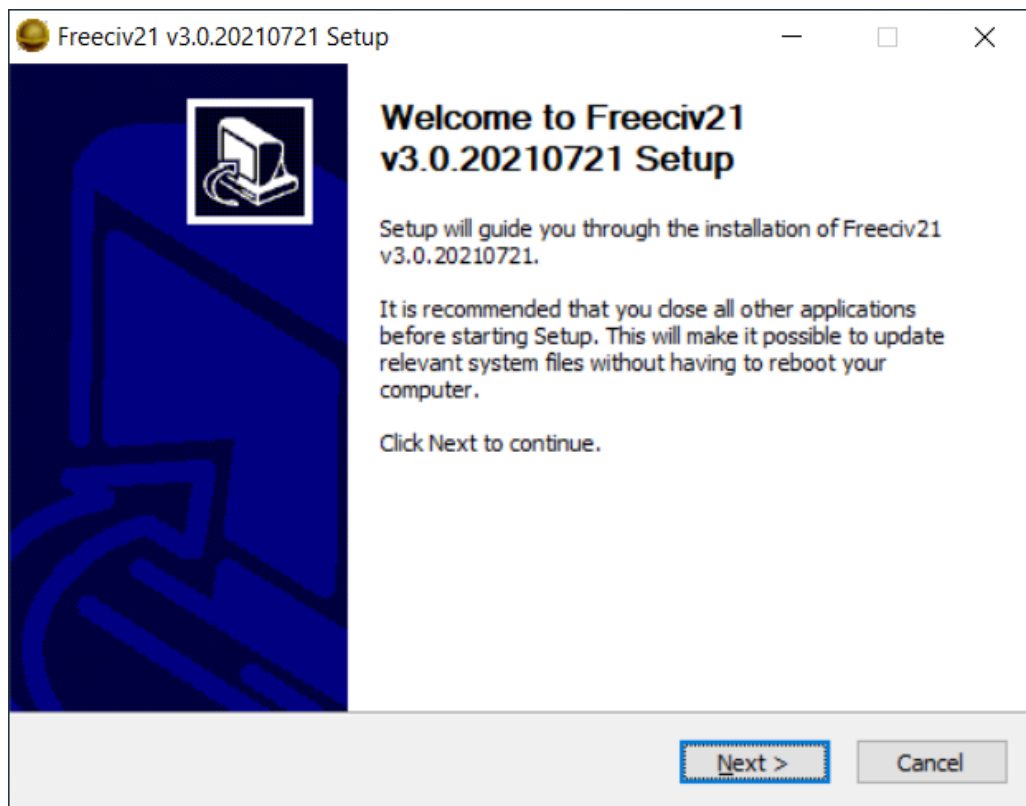


Fig. 1.9: User Install - Welcome

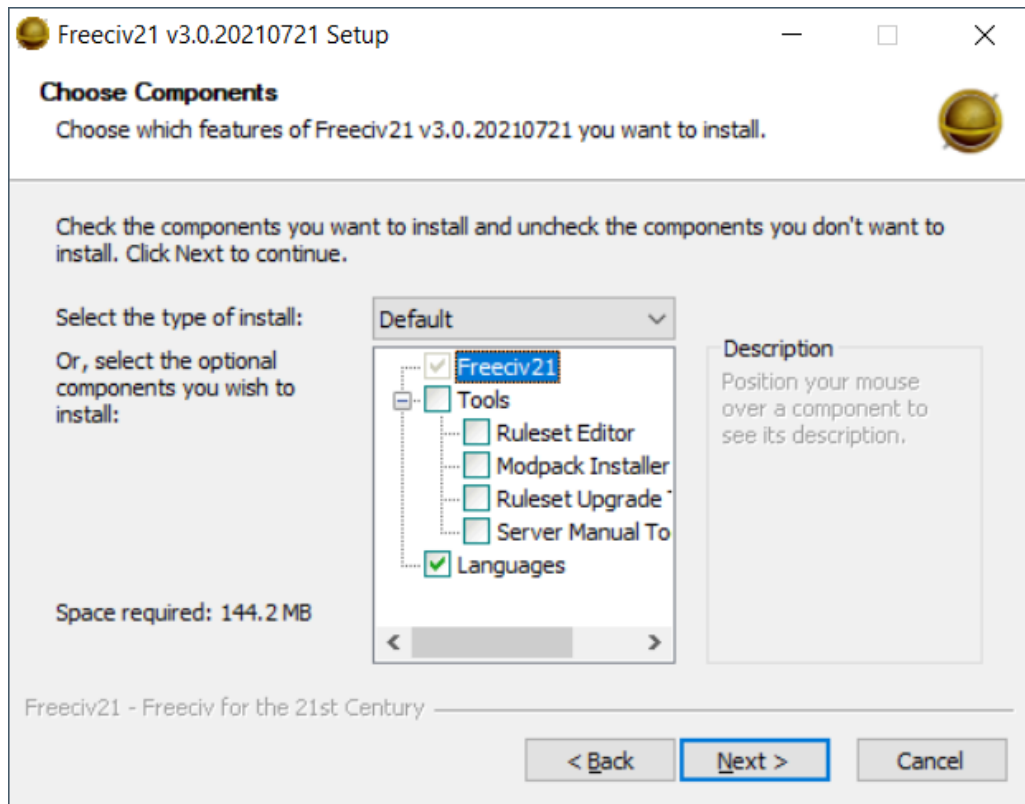


Fig. 1.10: User Install - Select Components

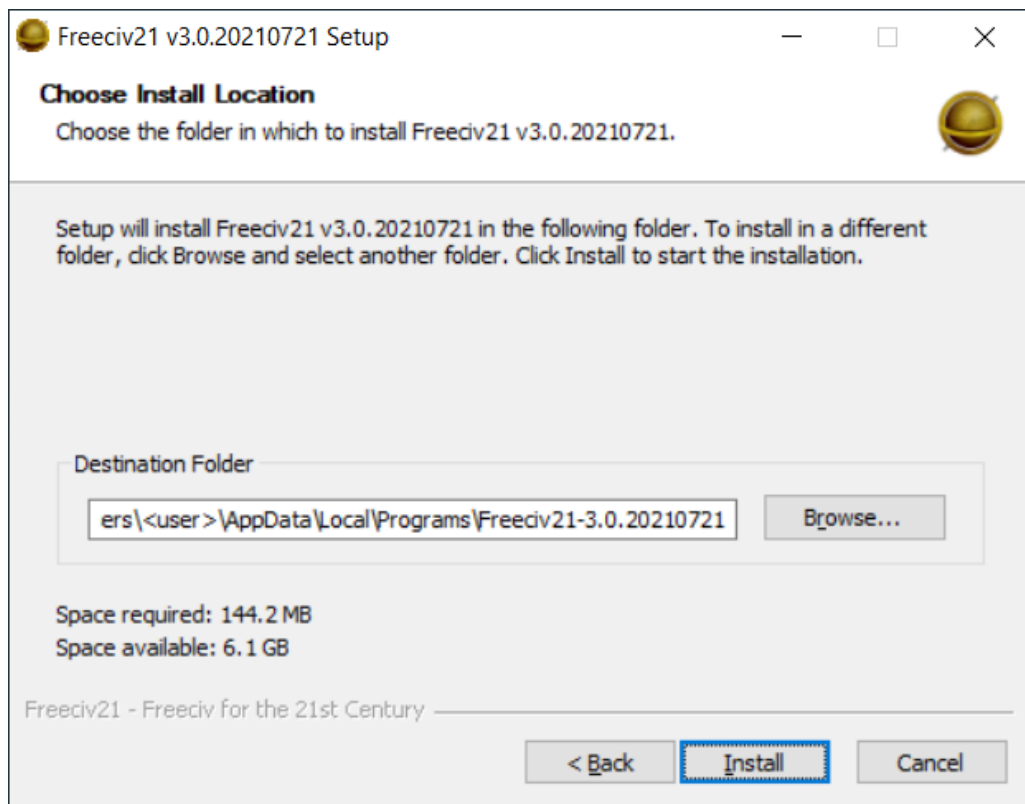


Fig. 1.11: User Install - Installation Folder

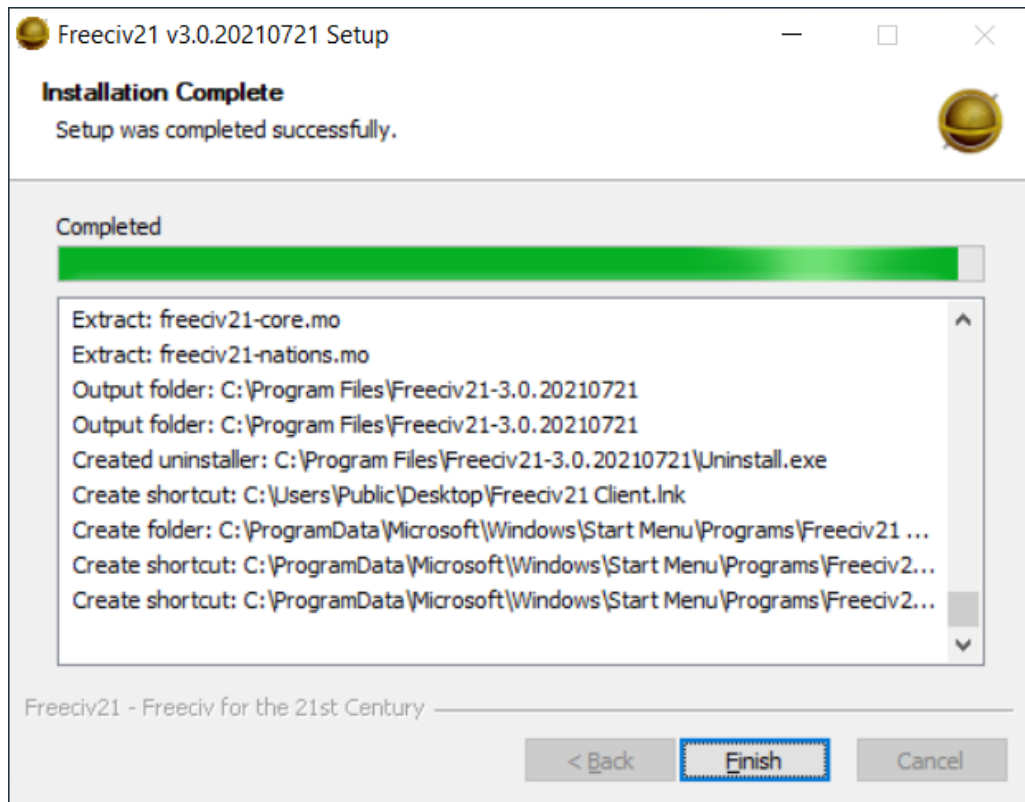


Fig. 1.12: User Install - Copy Files

If you want to run Freeciv21 right away, leave the box checked and click *Finish*. Otherwise, uncheck the box before clicking *Finish* to complete the installation.

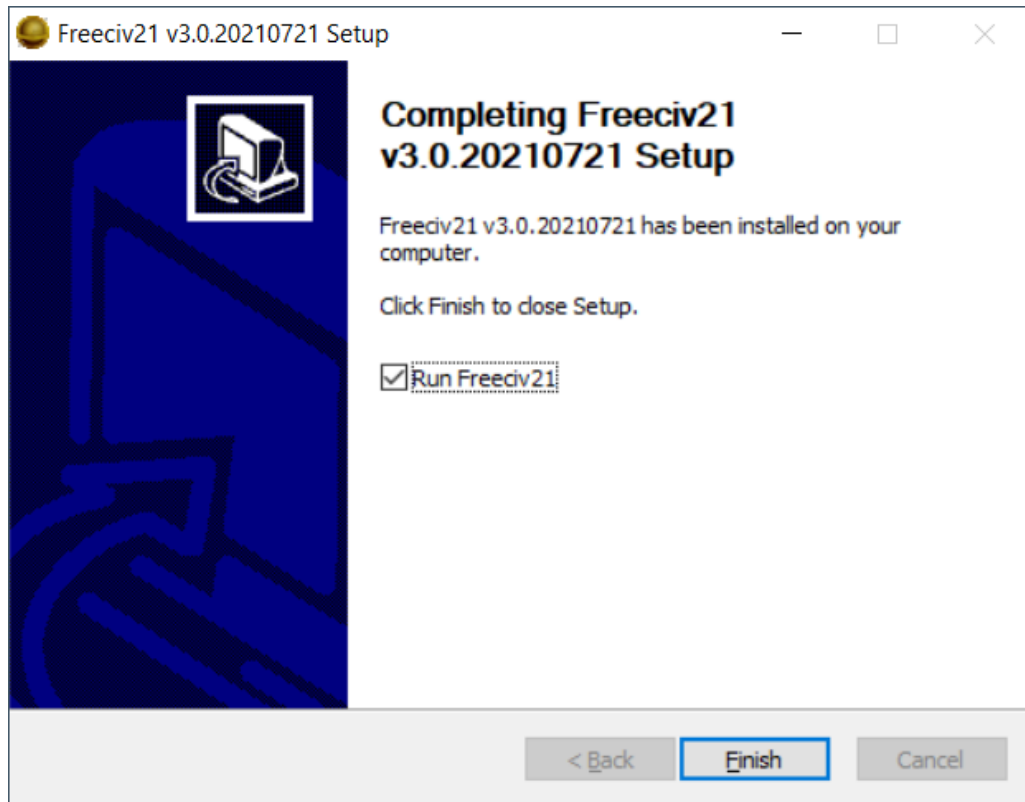


Fig. 1.13: User Install - Finish

You will find an icon for the client on the Desktop or in the Start Menu at *Start Menu* → *Freeciv21* → *Freeciv21 Client*.

1.4 Compiling Freeciv21

Freeciv21 has a number of prerequisites. Note, that apart from the first prerequisite, the Freeciv21 configuration process is smart enough to work out whether your system is suitable. If in doubt, just try it.

An operating system that support Qt

Any modern operating system that supports Qt 5.15+ is required. As of this writing this is Linux, Microsoft Windows® and Apple macOS®.

Linux Distributions:

- Arch
- Debian 11+ (Bullseye)®
- Fedora 30+
- Gentoo
- KDE Neon
- Manjaro
- Mint 20+ or Mint Debian Edition (set to Bullseye)

- openSUSE 15.3+
- Slackware
- Ubuntu 22.04 LTS+
- Fedora 37+

Note: The above list of Linux distributions is, of course, not exhaustive. The Freeciv21 Community has simply listed the mainline, well supported, distributions here. The code repository has Continuous Integration enabled and all code commits pass through Ubuntu, macOS and Windows for testing. It is assumed that the user is keeping his/her computer OS up to date. Support by the community for these distributions will be better than for some of the others out there, so keep that in mind if you are not an experienced Linux user.

Note: The following instructions on this page are for Linux, MSYS2 and macOS environments. You will need to *install MSYS2* first before continuing here if using MSYS2 on Windows. However, you can also compile on Windows with Microsoft *Visual Studio*. The Visual Studio instructions are self contained. No need to return to this page after following the installation instructions.

A C and C++ compiler

Freeciv21 is written in very portable C and C++. Both 32 and 64-bit machines are supported. You cannot use a “K&R C” compiler. The C++ compiler must support C++ 17.

Development of Freeciv21 is primarily done with gcc, the GNU project’s excellent C and C++ compiler. For complete cross-platform support the Longturn community uses the LLVM project’s clang-cl compiler, which is supported on Linux, Windows and macOS.

The Cmake program

Freeciv21 developers generally use cmake, the Kitware make program. You can check if you have cmake installed on your system by typing the following command. The output should include “Kitware cmake” somewhere and the version should be >=3.21.

```
$ cmake --version
```

The Ninja cmake build program

Freeciv21 uses the ninja build tool. You can check if you have ninja installed on your system by typing the following command. The output should include ninja version >=1.10.

```
$ ninja --version
```

GNU Libtool

GNU Libtool is a generic library support script that hides the complexity of using shared libraries behind a consistent, portable interface. Freeciv21 requires version 2.2 or better.

<https://www.gnu.org/software/libtool/>

SQLite

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day. Freeciv21 requires version 3.

<https://www.sqlite.org>

GNU Gettext

GNU Gettext is used for Internationalization support. Freeciv21 requires version 0.15 or better. The `xgettext`

program is required to create the *.gmo files which are not included in the git tree.

<https://www.gnu.org/software/gettext/>

Lua

Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description. Exact version 5.3 is preferred.

<https://www.lua.org/about.html>

KF 5 Archive Library

KDE frameworks KArchive provides classes for easy reading, creation and manipulation of “archive” formats like ZIP and TAR.

<https://invent.kde.org/frameworks/karchive>

SDL2_Mixer

SDL_mixer is a sample multi-channel audio mixer library.

<https://www.libsdl.org>

Python

Freeciv21 requires version 3 of Python

<https://www.python.org>

1.4.1 Prerequisites for the Client and Tools

The Freeciv21 project maintains a single Qt based client.

C++ compiler.

The client is written in C++, so you need an appropriate compiler. In Freeciv21 development, g++ has been used as well as tests against LLVM’s compiler (clang++)

QT Libraries

Freeciv21 uses the Qt libraries, specifically Qt5Core, Qt5Gui, Qt5Network, Qt5Svg, and Qt5Widgets libraries and headers.

At least version 5.15 is required.

1.4.2 Installing Package Dependencies

See the *Debian Linux Packages* section below on the steps to install the components for Debian Linux and its variants.

See the *Fedora Linux Packages* section below on the steps to install the components for Fedora Linux

See the *macOS Packages* section below on the steps to install the components for Apple macOS.

If you are running Windows and want to use the MSYS2 environment and have not set it up yet, then *do so now*, before continuing.

Lastly, if you are running Windows and want to use Visual Studio, you can follow the Microsoft *Visual Studio* instructions. The Visual Studio instructions are self contained. You do not need to return here in that case.

1.4.3 Debian Linux Packages

Below are all the command line steps needed to start with a fresh install of Debian or its variants (e.g. Ubuntu, Linux Mint) to install Freeciv21.

Start with ensuring you have a source repository (deb-src) turned on in apt sources and then run the following commands:

```
$ sudo apt update

$ sudo apt install git \
    cmake \
    ninja-build \
    python3 \
    qtbase5-dev \
    libqt5svg5-dev \
    libkf5archive-dev \
    liblua5.3-dev \
    libsqlite3-dev \
    libsdl2-mixer-dev \
    libmagickwand-dev \
    libunwind-dev \
    libdw-dev \
    python3-sphinx \
    clang-format-11
```

At this point, follow the steps in *Obtaining the Source Code* section below.

1.4.4 Fedora Linux Packages

Below are all the command line steps needed to start with a fresh install of Fedora or its variants (e.g. RHEL, CentOS Stream) to install Freeciv21.

```
$ sudo dnf install git \
    cmake \
    ninja-build \
    python \
    gettext \
    qt5-qtbase-devel \
    qt5-qtsvg-devel \
    kf5-karchive-devel \
    lua-devel \
    sqlite-devel \
    SDL2_mixer-devel \
    readline-devel \
    zlib-devel \
    libunwind-devel \
    elfutils-libs \
    python3-sphinx
```

At this point, follow the steps in *Obtaining the Source Code* section below.

1.4.5 macOS Packages

Below are all the command line steps needed to start with a fresh install of macOS.

```
$ brew update

$ brew install \
  cmake \
  ninja \
  python3 \
  gettext \
  vcpkg
brew link gettext --force

$ export VCPKG_ROOT="$HOME/vcpkg"
```

1.4.6 Obtaining the Source Code

In order to compile Freeciv21, you need a local copy of the source code. You can download a saved version in an archive file (.tar.gz or .zip) of the code from the project releases page at <https://github.com/longturn/freeciv21/releases>. Alternately you can get the latest from the master branch with the git program with this command:

```
$ git clone https://github.com/longturn/freeciv21.git
$ cd freeciv21
```

1.4.7 Configuring

Configuring Freeciv21 for compilation requires the use of the cmake program.

On Debian Linux, to build with defaults enter the following command from the freeciv21 directory. Continue reading in the *Other CMake Notes* section below for more notes about other command line options you can give cmake.

```
$ cmake . -B build -G Ninja
```

Note: If you are compiling on Windows MSYS2, alter the above command to add

-DCMAKE_INSTALL_PREFIX=\$PWD/build/install

at the end.

On macOS, you need to use a preset that is defined in the CMakePresets.json file. When complete you can go to the *Compiling/Building* section below to continue.

```
$ cmake --preset fullrelease-macos -S . -B build
```

Note: The first time you run the this command, cmake invokes the vcpkg installation process to download and compile all of the project dependencies listed in the manifest file: vcpkg.json. **This will take a very long time.** On a fast computer with a good Internet connection it will take at least 3 hours to complete. Everything will be downloaded and compiled into the \$HOME/vcpkg directory. Binaries for the packages will be copied into the ./build/ directory inside of the main Freeciv21 directory and reused for subsequent builds.

1.4.8 Compiling/Building

Once the build files have been written, then compile with this command:

```
$ cmake --build build
```

1.4.9 Installing

Once the compilation is complete, install the game with this command.

```
$ cmake --build build --target install
```

Note: If you did not change the default install prefix, you will need to elevate privileges with `sudo`.

Tip: If you want to enable menu integration for the installed copy of Freeciv21, you will want to copy the `.desktop` files in `$CMAKE_INSTALL_PREFIX/share/applications` to `$HOME/.local/share/applications`.

This is only necessary if you change the installation prefix. If you do not and use elevated privileges, then the files get copied to the system default location.

At this point, the compilation and installation process is complete. The following sections document other aspects of the packaging and documentation generation process.

1.4.10 Debian and Windows Package Notes

Operating System native packages can be generated for Debian and Windows based systems.

Debian

Assuming you have obtained the source code and installed the package dependencies in the sections above, follow these steps to generate the Debian package:

```
$ cmake --build build --target package
```

When the last command is finished running, you will find a `.deb` installer in `build/Linux-$arch`

Microsoft Windows

There are two platforms available for installing Freeciv21 on Windows: *MSYS2* and *Visual Studio*. The package target is only supported on MSYS2 due to licensing [constraints](#).

Once your MSYS2 environment is ready, start with *Obtaining the Source Code* above. Instead of installing, use this command to create the Windows Installer package:

```
$ cmake --build build --target package
```

When the command is finished running, you will find an installer in `build/Windows-$arch`

1.4.11 Documentation Build Notes

Freeciv21 uses python3-sphinx and <https://readthedocs.org/> to generate the well formatted HTML documentation that you are reading right now. To generate a local copy of the documentation from the docs directory you need two dependencies and a special build target.

The Sphinx Build Program

The sphinx-build program is used to generate the documentation from reStructuredText files (*.rst).

<https://www.sphinx-doc.org/en/master/index.html>

ReadTheDocs Theme

Freeciv21 uses the Read The Docs (RTD) theme for the general look and feel of the documentation.

<https://sphinx-rtd-theme.readthedocs.io/en/stable/>

If you are running Debian Linux, the base program is installed by the instructions in the *Debian Linux Packages* section above. The documentation is not built by default from the steps in *Compiling/Building* above. To generate a local copy of the documentation, issue this command:

```
$ cmake --build build --target docs
```

1.4.12 Other CMake Notes

To customize the compile, cmake requires the use of command line parameters. cmake calls them directives and they start with -D. The defaults are marked with **bold** text.

Table 1.1: Cmake Parameters

Directive	Description
FREECIV_ENABLE_TOOLS={ ON /OI}	Enables all the tools with one parameter (Ruledit, FCMP, Ruleup, and Manual)
FREECIV_ENABLE_SERVER={ ON /C}	Enables the server. Should typically set to ON to be able to play AI games
FREECIV_ENABLE_NLS={ ON /OFF}	Enables Native Language Support
FREECIV_ENABLE_CIVMANUAL={	Enables the Freeciv21 Manual application
FREECIV_ENABLE_CLIENT={ ON /O}	Enables the Qt client. Should typically set to ON unless you only want the server
FREECIV_ENABLE_FCMP_CLI={ON	Enables the command line version of the Freeciv21 Modpack Installer
FREECIV_ENABLE_FCMP_QT={ON,	Enables the Qt version of the Freeciv21 Modpack Installer (recommended)
FREECIV_ENABLE_RULEEDIT={ON/	Enables the Ruleset Editor
FREECIV_ENABLE_RULEUP={ON/C}	Enables the Ruleset upgrade tool
FREECIV_ENABLE_MANPAGES={O	Enables the generation of manpages. This is primarily relevant for Unix-like operating systems
FREECIV_USE_VCPKG={ON/ OFF }	Enables the use of VCPKG
FREECIV_DOWNLOAD_FONTS{ ON	Enables the downloading of Libertinus Fonts
CMAKE_BUILD_TYPE={ Release /Del	Changes the Build Type. Most people will pick Release
CMAKE_INSTALL_PREFIX=/some/p	Allows an alternative install path. Default is /usr/local/freeciv21

For more information on other cmake directives see <https://cmake.org/cmake/help/latest/manual/cmake-variables.7.html>.

Once the command line directives are determined, the appropriate command looks like this:

```
$ cmake . -B build -G Ninja \  
-DCMAKE_BUILD_TYPE=Release \  
-DCMAKE_INSTALL_PREFIX=$HOME/Install/Freeciv21
```

A very common Debian Linux configuration command looks like this:

```
$ cmake . -B build -G Ninja -DCMAKE_INSTALL_PREFIX=$PWD/build/install
```


PLAYING

The Playing category is an area for documentation editors to provide tips and tricks on ways to play Freeciv21 and any of the varying rulesets that the Longturn.net community likes to play.

2.1 Frequently Asked Questions (FAQ)

The following page has a listing of frequently asked questions with answers about Freeciv21.

2.1.1 Gameplay

This section of the FAQ is broken into a collection of sub-sections all surrounding general Gameplay aspects of Freeciv21:

- *Gameplay General*
- *Diplomacy*
- *Game Map and Tilesets*
- *Cities and Terrain*
- *Units - General*
- *Units - Military*
- *Other*

Gameplay General

This subsection of the Gameplay section is a generalized discussion of questions.

OK, I installed Freeciv21. How do I play?

Start the game. Depending on your system, you might choose it from a menu, double-click on the `freeciv21-client` executable program, or type `freeciv21-client` in a terminal window.

Once the game starts, to begin a single-player game, select *Start new game*. Now edit your game settings (the defaults should be fine for a beginner-level single-player game) and press the *Start* button.

Freeciv21 is a client/server system. However, in most cases you do not have to worry about this. The client starts a server automatically for you when you start a new single-player game.

Once the game is started you can find information in the *Help* menu. If you have never played a Civilization-style game before you may want to look at help on *Strategy and Tactics*.

You can continue to change the game settings through the *Game* → *Game Options* menu. Type `/help` in the *server/chatline* widget to get more information about server commands.

A more detailed explanation of how to play Freeciv21 is available in *How to Play Freeciv21*, and in the in-game help.

There is also a detailed *Game Manual* available for your reference.

Where is the chatline you are talking about, how do I chat?

There are two widgets on the *Main Map* View (F1): *server/chatline* and *messages*. The *messages* widget can be toggled visible/not-visible via a button on the *Top Bar*. The *server/chatline* is a floating widget on the *Main Map* View (F1) and is always visible.

The *server/chatline* widget can be used for normal chatting between players. To issue server commands, you start by typing a forward-slash `/` followed by the server command. You will see resulting server output messages.

During *pre-game* There is a *server/chatline* feature of the dialog box that can be used to set server parameters before a game starts. All game parameters are also available in the *Game Options* dialog.

See the in-game help on *Chatline* for more detail.

That sounds complicated and tedious. Is there a better way to do this?

No, there is no other better user interface way at this time. This is a big reason why the Longturn Community prefers using Discord. There are plans to improve this, but it is not implemented yet.

Is there a way to send a message to all your allies?

In the game, there is an option to the lower right of the *server/chatline* widget. When selected, any messages typed will only go to your allies.

Note: This option only shows up if you are playing an online Longturn Community game with a remote server. If you are playing a local single-player game against *AI*, this option does not show up since you cannot chat with the *AI*.

How do I find out about the available units, improvements, terrain types, and technologies?

There is extensive help on these topics in the *Help* menu, but only once the game has been started. This is because the in-game help is generated at run-time based on the settings as configured.

The game comes with an interactive tutorial scenario. You can run it by clicking on the *Tutorial* button on the start menu.

How do I play against computer players?

This is commonly called a single-player game and is the default. See *OK, I installed Freeciv21. How do I play?*

Can I build up the palace or throne room as in the commercial Civilization games?

No. This feature is not present in Freeciv21, and will not be until someone draws the graphics and writes the game related code for it. Feel free to *contribute*.

My opponents seem to be able to play two moves at once!

They are not. It only seems that way. Freeciv21's multi-player facilities are asynchronous: during a turn, moves from connected game interfaces are processed in the order they are received. Server managed movement is executed in between turns (e.g. at *TC*). This allows human players to surprise their opponents by clever use of *Goto* or quick fingers.

In single-player games against the *AI*, the moves from the computer are made at the beginning of the turn.

A turn in a typical Longturn game lasts 23 hours and it is always possible that they managed to log in twice between your two consecutive logins. Once a player has made all of their moves, a *TC* event must occur before they can move again. This does mean that a player can move a unit just before *TC* and just after and in between your two logins. In short, a player cannot *move twice* until you do.

The primary server setting to mitigate the *TC* problem is called *unitwaittime*, which imposes a minimum time between moves of a single unit on successive turns. This setting is used to prevent a varying collection of what the Longturn community calls "turn change shenanigans". For example, one such issue is moving a Worker into enemy territory just before *TC* and giving it orders to build a road. After *TC* you go in and capture a city using the road for move benefit. Without *unitwaittime* you would be able to move the Worker back to safety immediately, thereby preventing it from being captured or destroyed. With *unitwaittime* enabled, you have to wait the requisite amount of time. This makes the game harder, but also more fair since not everyone can be online at every *TC*.

Note: The *unitwaittime* setting is really only used in Longturn multi-player games and is not enabled/used for any of the single-player rulesets shipped with Freeciv21.

Why are the AI players so hard on 'novice' or 'easy'?

Short answer is... You are not expanding fast enough.

You can also turn off Fog of War. That way, you will see the attacks of the *AI*. Just type `/set fogofwar disabled` on the *server/chatline* before the game starts or by unchecking the box for fog of war in the *Game Options* dialog on the Military tab.

Why are the AI players so easy on ‘hard’?

Several reasons. For example, the *AI* is heavily play tested under and customized to the default ruleset and server settings. Although there are several provisions in the code to adapt to changing rules, playing under different conditions is quite a handicap for it. Though mostly the *AI* simply does not have a good, all encompassing strategy besides “eliminate nation x”.

To make the game harder, you could try putting some or all of the AI into a team. This will ensure that they will waste no time and resources negotiating with each other and spend them trying to eliminate you. They will also help each other by trading techs. Refer to *Advanced Player Setup* for more information.

You can also form more than one *AI* team by using any of the different predefined teams, or put some *AI* players teamed with you. Another alternative is to create *AI*’s that are of differing skill levels. The stronger *AI*’s will then go after the weaker ones.

What distinguishes AI players from humans? What do the skill levels mean?

AI players in Freeciv21 operate in the server, partly before all game moves, partly afterwards. Unlike the game interface, they can in principle observe the full state of the game, including everything about other players, although most levels deliberately restrict what they look at to some extent.

All *AI* players can change production without penalty. Some levels (generally the harder ones) get other exceptions from game rules. Conversely, easier levels get some penalties, and deliberately play less well in some regards.

For more details about how the skill levels differ from each other, see the help for the relevant server command (for instance `/help hard`).

Other than as noted here, the *AI* players are not known to cheat.

Does the game have a combat calculator, like other Civ games have?

There is no integrated combat calculator. You can use the one on the Longturn.net website here: <https://longturn.net/warcalc/>. You can also select an attacking unit and then middle-click over a defending unit and in the pop-up window you will see the odds of win/loss.

Where in the game does it say what government you are currently under?

On the top bar near the right side there is a *national status view* that shows what your national budget consists of as well as what Government you are under, chance for Global Warming, Nuclear Winter, and how far along you are with research. You can hover your mouse over any of these icons to see more details.

What government do you start under?

You start under Despotism in *LTT*. This is a ruleset configured item.

Do things that give more trade only give this bonus if there is already at least 1 trade produced on a tile?

The short answer is yes in *LTT*. This is a ruleset configured item.

Diplomacy

This subsection of the Gameplay section is a discussion around Diplomacy.

Why cannot I attack another player's units?

You have to declare war first. See the section for *How do I declare war on another player?* below.

Note: In some rulesets, you start out at war with all players. In other rulesets, as soon as you make contact with a player, you enter armistice towards peace. At lower skill levels, *AI* players offer you a cease-fire treaty upon first contact, which if accepted has to be broken before you can attack the player's units or cities. The main thing to remember is you have to be in the diplomatic state of war in order to attack an enemy.

How do I declare war on another player?

Go to the *Nations and Diplomacy* View (F3), select the player row, then click *Cancel Treaty* at the top. This drops you from *cease fire*, *armistice*, or *peace* into *war*. If you have already signed a permanent *alliance* treaty with the player, you will have to cancel treaties several times to get to *war*.

See the in-game help on *Diplomacy* for more detail.

Note: The ability to arbitrarily leave *peace* and go to *war* is also heavily dependent on the form of government your nation is currently ruled by. See the in-game help on *Government* for more details.

How do I do diplomatic meetings?

Go to the *Nations and Diplomacy* View (F3), select the player row, then choose *Meet* at the top. Remember that you have to either have contact with the player or an embassy established in one of their cities with a Diplomat.

How do I trade money with other players?

If you want to make a monetary exchange, first initiate a diplomatic meeting as described in the section *How do I do diplomatic meetings?* above. In the diplomacy dialog, enter the amount you wish to give in the gold input field on your side or the amount you wish to receive in the gold input field on their side.

Note: In some rulesets there might be a “tax” on gold transfers, so watch out that not all gold will make it to its intended destination nation.

Is there a way to tell who is allied with who?

The *Nations and Diplomacy* View (F3) shows diplomacy and technology advance information if you have an embassy with the target nation. To see what is going on, select a nation and look at the bottom of the page.

Game Map and Tilesets

This subsection of the Gameplay section is a discussion around the game map and tilesets (the graphics layer).

Can one use a regular square tileset for iso-square maps and vice versa?

While that is technically possible, hex and iso-hex topologies are not directly compatible with each other, so the result is not playable in a good (visualization) way. In the game interface you can force the change of tileset by going to *Game* → *Load Another Tileset*. If the game interface can change, it will and you will be able to experiment a bit. If there is a complete discrepancy, the game interface will throw an error and will not make the requested change.

How do I play on a hexagonal grid?

It is possible to play with hexagonal instead of rectangular tiles. To do this you need to set your topology before the game starts. Set this with Map topology index from the *game options*, dialog or in the *server/chatline*:

```
/set topology hex|iso|wrapx
```

This will cause the game interface to use an isometric hexagonal tileset when the game starts. Go to *Game* → *Set local options* to choose a different one from the drop-down. hexemplio and isophex are included with the game.

You may also play with overhead hexagonal, in which case you want to set the topology setting to hex|wrapx. The hex2t tileset is supplied for this mode.

Can one use a hexagonal tileset for iso-hex maps and vice versa?

See the question *Can one use a regular square tileset for iso-square maps and vice versa?* above.

Cities and Terrain

This subsection of the Gameplay section is a discussion around cities and the terrain around them.

My irrigated grassland produces only 2 food. Is this a bug?

No, it is not – it is a feature. Your government is probably Despotism, which has a -1 output penalty whenever a tile produces more than 2 units of food, production, or trade. You should change your government. See the in-game help on *Government* for more detail to get rid of this penalty.

This feature is also not 100% affected by the form of government. There are some small and great wonders in certain rulesets that get rid of the output penalty. See the in-game help on *City Improvements* and *Wonders* for more information.

Can I build land over sea or transform ocean to land?

Yes. You can do that by placing an Engineer in a Transport and going to the ocean tile you want to build land on. Click the Transport to display a list of the transported Engineers and activate them. Then give them the order of transforming the tile to swamp. This will take a very long time though, so you had better try with 6 or 8 Engineers at a time. There must be 3 adjacent land tiles to the ocean tile (e.g. a land corner) you are transforming for this activity to work.

Is there an enforced minimum distance between cities?

This depends on the ruleset. In *LTT* there is a minimum distance of 3 empty tiles between two cities. You can think of it as “no city can be built within the work radius of another city”, since the work radius of a city is also 3 tiles in *LTT*.

This setting, known as `citymindist`, can be set in the server settings before a local game starts or by changing at the *server/chatline*:

```
/set citymindist 4
```

If your city is going to grow next turn and you rush-buy a Granary, do you still get the food savings?

Yes. Production is “produced” before growth at turn change. This is true for all rulesets as it is part of the standard *Turn Change* process.

How much population do Settlers take to build?

Two in the *LTT* ruleset. This is a ruleset configurable item. See the in-game help on *Units* for more detail to see what the settings is for the ruleset you loaded at game start.

Do tiles remember terraforming progress?

If you change orders for the unit doing the terraforming and do not change them back within the same turn, the terraforming progress is lost. If you change orders and then change them back, nothing special happens. Terraforming is always processed at *TC*.

How frequently do natural disasters happen?

This depends on the ruleset. For the *LTT* ruleset, all natural disasters have a 1% probability to happen each turn. The default is 10%. This setting, known as `disasters`, can be set in the server settings before a local game starts or by changing at the *server/chatline*:

```
/set disasters 20
```

Does the city work area change in any way during the game?

This is a ruleset configured option. In *LTT* the initial value is 15, effectively giving 3 tiles “out” from the city center in all directions. Varying technologies or buildings can be programmed into the ruleset to change the vision radius (e.g. the work area) of a city.

Is it worth it to build cities on hills (potentially with rivers), or is the risk of earthquakes and floods too large?

It is usually worth it, since hills and rivers have great defense values. Rivers also allow you to build an Aqueduct, River without the knowledge of Construction and it is much cheaper to build and requires no upkeep.

Note: It is a great strategy if you can do this to place your first city (Capital) either on a river or adjacent to one to get this “fresh water” effect. You can get the city up to size 16 very fast with the right growth strategy.

Is there a benefit to lake tiles over ocean tiles? What are their differences?

There are at least the following differences:

- Lake tiles allow an adjacent city to build a cheap Aqueduct, Lake with no upkeep and before the discovery of Construction. This is commonly referred to as the “fresh water” effect.
- Lake tiles give more food than ocean tiles, especially with the Fish tile special.
- Shallow ocean gives +1 production with Offshore Platform. The Offshore Platform city improvement often comes with the discovery of Miniaturization in most rulesets, but this is a ruleset configurable item.
- Some ships cannot travel on deep ocean (such as Triremes)
- Shallow ocean has a 10% defense bonus.
- Ocean tiles allow you to build Harbor, giving +1 food. The Harbor city improvement often comes with the discovery of Seafaring in most rulesets, but this is a ruleset configurable item.

Is “Aqueduct, River” identical to “Aqueduct, Lake”?

In most rulesets, yes. It is part of the “fresh water” effect of giving a cheap Aqueduct that has no gold upkeep and does not require the discovery of Construction.

Is the city tile worked for free?

Yes. This is hard-coded in the server. In all rulesets a size 1 city will always have two tiles being worked by the citizens of the city: the city center tile and another one in its working radius that is not being worked by an adjacent city.

Do you get free irrigation on the city tile?

You get a “virtual” irrigation effect. It works the same way as regular irrigation for food purposes, but does not allow you to build irrigation next to the city by itself. You will have to build regular irrigation on the city center tile to do that. The free irrigation is lost if you build a mine on the city tile (just like regular irrigation on a regular tile is lost with a mine). This means that a desert tile that is mined has zero (0) food, even when on a city center tile. Since the “virtual” irrigation works like regular irrigation, if you build a city on a tile that cannot be irrigated normally (e.g. a forest), you do not get any food bonus.

In the late game, many rulesets have a Supermarket, that comes with the discovery of Refrigeration. A player can then use Workers or Engineers to add Farmland on top of the existing irrigation for an additional food bonus. In this sense, if you want to get the Farmland food effect on a city center tile that is already “virtually” irrigated you will have to actually irrigate the tile and then add Farmland on top of it, just like any other regular tile.

Does the city tile have any production bonuses?

A city tile has a +1 production bonus, added after any other bonuses (such as Railroad).

Does LTT have the extra food from rivers on a desert tile when irrigated, like other rulesets have?

Yes, an irrigated desert tile with a river gives an extra +1 food in addition to the regular irrigation food bonus. This is a game engine (server) item and is not driven by a ruleset, such as *LTT*.

Is there any penalty when changing a city production task?

There are 4 “categories” of production: units, city improvements (e.g. Buildings), great wonders, and small wonders. If you change within a “category” (e.g. Phalanx to Horsemen, or Library to Bank), there is no penalty. If you change across categories (e.g. Archers to Library, or Leonardo’s Workshop to Frigate), there is a 50% penalty. If you change back to the same category within the same turn, the penalty is reversed. If you change multiple times, the penalty is only applied once, which means that if you change the production target more than once in a turn there will be no penalty as long as you land on the same “category” as was active at the beginning of the turn.

Is there a way to claim tiles using Diplomats?

No. In Freeciv21 there are generally 4 ways to gain tile ownership:

- Build a city and claim the tiles first.
- Grow your cities super big and much bigger than your neighbor’s cities. National borders can move at Turn Change based on culture score.
- Build a Fortress and place a Military unit (e.g. a Phalanx) inside the Fortress.
- Conquer the city and take its tiles for your own.

Is there a way to create a hill other than terraforming a mountain?

You can also terraform a hill from plains with Engineers. In some rulesets, such as *LTT*, this is very expensive in worker *MP* and can take some time unless you place many Engineers on the tile at the same time.

Can you build a hill under a city?

Yes, you sure can!

Units - General

This subsection of the Gameplay section is a generalized discussion around units.

When does the game inform you of enemy movement within your units' field of vision?

It depends on the status of the unit. If the unit is fortified or working on another task (e.g. irrigation) then you will not be notified. Only the **Sentry** status will give you a notification.

If I move a unit onto a mountain, does that change how many movement points the unit has next turn?

All units that end their turn on a mountain start with 1 less *MP* the following turn. The exception to this rule are units that ignore terrain movement completely (e.g. Explorers, and Alpine Troops). This is known as “ignoring terrain effects”.

What is a unit's terraforming speed based on?

It is based on the base amount of *MP*'s for that unit and veteran level bonus. The base terraforming duration is specified in the ruleset files.

Todo: This is discussed in detail in a forthcoming LTT Gamer's Manual. Update this entry at that time.

Can workers do all land conversions? Or are most land conversions locked behind engineers?

All Workers can do land conversions except for major land transformations, which are available only with Engineers.

Does a damaged worker work slower than normal?

No, *HP*'s do not factor in a Worker's ability to conduct infrastructure improvements to tiles.

When terraforming, does some movement get used on the last turn of terraforming? Does the unit start with less movement points?

Terraforming does not affect a unit's *MP*'s in any way.

Does damage reduce the amount of movement points the unit has?

This depends on the unit class and the ruleset configuration. If a unit's help text specifies it is "slowed down when damaged", then it does. If it does not say anything about it, then it does not.

Do Caravans give full production?

The Caravan unit is a special unit that allows a player to move production (shields) from one city to another in order to increase the speed of constructing wonders (both small and great). This effect only works for wonders and no other city improvement. The Caravan unit acts like any other unit when disbanded in a city: it gives back 50% of the shields it took to construct it in the first place. In many rulesets the Freight becomes available in the late game and obsoletes the Caravan. The Freight works the same way.

How does unit leveling work?

Freeciv21 calls this unit "Veterancy" or "Veteran Levels". You have a chance every turn for any kind of unit to gain an upgrade via experience. The experience depends on the unit and what they are doing. For example, a Worker gains experience by creating terrain infrastructure, or a Phalanx gains experience during both defense and offense (attack) movements. See the following table:

Level	Combat Strength	Move Bonus	Promotion Chance	
			In Combat	By Working
Green	1x	0	50	9
Veteran 1 (v)	1.5x (from Green)	1/3 (from Green)	45	6
Veteran 2 (vv)	1.75x	2/3	40	6
Veteran 3 (vvv)	2x	1	35	6
Hardened 1 (h1)	2.25x	1 1/3	30	5
Hardened 2 (h2)	2.5x	1 2/3	25	5
Hardened 3 (h3)	2.75x	2	20	4
Elite 1 (e1)	3x	2 1/3	15	4
Elite 2 (e2)	3.25x	2 2/3	10	3
Elite 3 (e3)	3.5x	3	0	0

Is it possible to change a unit's home city?

To be clear, a unit's "home city" is the city that produced it.

It is possible when the unit is moved to a city that is not its current home city. You then get an option to change the home city. With the unit in a city you can either use shortcut key "h" or *Unit* → *Set Home City* to re-home the unit to the city it is inside.

Note: Some rulesets allow “unhomed” units. These kind of units will never have a home city and you cannot change it, even if you wanted to. These units have no upkeep, so they can stay unhomed.

Are queued goto commands executed before or after units and city improvements are built?

After. For example, you can beat an enemy attacking unit with a queued *Goto* to your city by rush-buying a defensive unit (it will get built first during normal *turn change processing*), and the attacking unit will move after that.

Units - Military

This subsection of the Gameplay section is a discussion around military units specifically.

My opponent’s last city is on a 1x1 island so I cannot conquer it and they will not give up. What can I do?

It depends on the ruleset, but often researching Amphibious Warfare will allow you to build a Marine. Alternatively research Combined Arms and either move a Helicopter or airdrop a Paratrooper there. When viewing the in-game help text for *Units*, be on the look out for Can launch attack from non-native tiles. This is the unit’s feature that allows you to attack from the ocean or air versus land, which is a native tile.

If you cannot build Marines yet, but you do have Engineers, and other land is close-by, you can also build a land-bridge to the island (i.e. transform the ocean). If you choose this route, make sure that your Transport is well defended!

Does a unit with less than 1 movement point remaining have weaker attacks?

Yes, the base attack is multiplied by the remaining *MP*’s when the unit has less than 1 *MP* left. This is commonly known as “tired attack”. As an example, a green Knights (base attack 6) with 6/9 *MP*’s remaining will attack as if it had attack 4.

How can I tell what final defense a unit will have after applying all bonuses from terrain, fortification, city, and such?

The game interface does not show this information, so you will have to calculate it manually. Math is an important element of all Freeciv21 games, and especially the *LTT* and *LTX* multi-player games the Longturn Community enjoys playing. As in the game of Chess, the “board” does not do the math for you. You must gauge the risk-reward ratios of your moves and counter-moves. This is the same in Freeciv21. The game will not do the math for you. This table should help you in doing the math:

Terrain		Open (Sen- tried)	Open (Forti- fied)	Fortress (Open)	Fortress (Fortified)	City <= 8	City >= 9	City <= 8 w/ Walls	City >=9 w/ Walls
Grass, Desert, Desert	Plains, Tundra,	1.0x	1.5x	2.0x	3.0x	2.25x	3.0x	3.75x	4.5x
Forest, Swamp	Jungle,	1.25x	1.88x	2.5x	3.75x	2.81x	3.75x	4.69x	5.63x
Hills		1.5x	2.25x	3.0x	4.5x	3.38x	4.5x	5.63x	6.75x
Mountains		2.0x	3.0x	4.0x	6.0x	N/A	N/A	N/A	N/A
w/ River		+1.25x on top of the other modifiers above							

Tip: The legacy Freeciv WiKi gives some good information in the Game Manual about Terrain here: <https://freeciv.fandom.com/wiki/Terrain>.

What is the math for upgrading units in LTT?

The basic upgrade cost is the same as disbanding the old unit in a city, and then rush-buying the new unit from the contributed shields (production).

Example: Phalanx → Pikeman

The Phalanx contributes 7 shields ($15 \div 2$ rounded down). The Pikeman costs 25 shields. The remaining 18 shields ($25 - 7$), is bought with gold using the formula for rush-buying units. For the math folks out there, the formula for rush-buying units (in all cases, not just for upgrades) is: $2p + \frac{p^2}{20}$ where p is the remaining production (or shields).

Are diplomats used up when investigating an enemy city?

Yes, they are destroyed/consumed after conducting an “investigate city” action. In some rulesets (notably *LTT* and *LTX*), there is a Spy available when you research Espionage. The Spy is not consumed by the same actions as the Diplomat.

Are there any other diplomatic units other than Diplomats and Spies?

The units available is highly dependent on the ruleset. For the *LTT* and *LTX* rulesets there are a couple “tech” stealing units: Scribe and Scholar. They can be used to steal (incite) units from other players and also to steal technology. In the *LTT* and *LTX* rulesets, there is no technology trading (between allies), so these two units were created as a way to allow technology trading, but at a risk.

When my unit moves in my territory on rivers, it costs 1/3 MP per tile. If I move on a river in enemy territory, it costs the full MP for the tile. Why?

Tile improvements that affect movement (rivers, roads, railroads) only apply when the unit is on allied territory, or on territory not owned by anyone. When moving through enemy territory, the terrain acts as if those improvements do not exist. The server setting that controls this is called `restrictinfra`. This value can be set in the server settings before a local game starts or by changing at the *server/chatline*:

```
/set restrictinfra FALSE
```

The *LTT* and *LTX* rulesets used by the Longturn Community have this value set to TRUE. This is also the default setting for many of the single-player rulesets shipped with Freeciv21.

Is there a way to see potential battle odds?

Yes. Select the unit you want to attack with and then middle-click (or Alt-click on Windows) over the potential target and a pop-up window will show you the odds of attack and defense taking into account all aspects of the attack (or defense) include terrain bonus, unit veterancy, etc.

When a city is captured, all units homed in that city that are currently in another city of yours are re-homed to that city. What happens to the the other units?

Any units not in a native city (e.g. your own city) are lost. This includes allied cities or outside of any city in the field.

Other

This subsection of the Gameplay section is a catchall area for questions do not fit nicely into the other subsections.

Can I change settings or rules to get different types of games?

Of course. Before the game is started, you may change settings through the *Game Options* dialog. You may also change these settings or use server commands through the *server/chatline* widget. If you use the *server/chatline*, use the:

```
/show
```

command to display the most commonly-changed settings, or

```
/help <setting>
```

to get help on a particular setting, or

```
/set <setting> <value>
```

to change a setting to a particular value. After the game begins you may still change some settings, but not others.

You can create rulesets or **modpacks** - alternative sets of units, buildings, and technologies. Several different rulesets come with the Freeciv21 distribution, including a *civ1* (Civilization 1 compatibility mode), and *civ2* (Civilization 2 compatibility mode). Use the `rulesetdir` command to change the ruleset (as in `/rulesetdir civ2`). For more information refer to *Modding*.

How compatible is Freeciv21 with the commercial Civilization games?

Freeciv21 was created as a multiplayer version of Civilization ® with players moving simultaneously. Rules and elements of Civilization II ®, and features required for single-player use, such as *AI* players, were added later.

This is why Freeciv21 comes with several game configurations (rulesets): the *civ1* and *civ2* rulesets implement game rules, elements and features that bring it as close as possible to Civilization I and Civilization II respectively, while other rulesets such as the *Classic* ruleset tries to reflect the most popular settings among Freeciv21 players. Unimplemented Civilization I and II features are mainly those that would have little or no benefit in multi-player mode, and nobody is working on closing this gap.

Little or no work is being done on implementing features from other similar games, such as SMAC, CTP or Civilization III+.

So the goal of compatibility is mainly used as a limiting factor in development. When a new feature is added to Freeciv21 that makes gameplay different, it is generally implemented in such a way that the *traditional* behaviour remains available as an option. However, we are not aiming for absolute 100% compatibility; in particular, we are aiming for bug-compatibility.

I want more action.

In Freeciv21, expansion is everything, even more so than in the single-player commercial Civilization games. Some players find it very tedious to build on an empire for hours and hours without even meeting an enemy.

There are various techniques to speed up the game. The best idea is to reduce the time and space allowed for expansion as much as possible. One idea for multiplayer mode is to add *AI* players: they reduce the space per player further, and you can toy around with them early on without other humans being aware of it. This only works after you can beat the *AI*, of course.

Another idea is to create starting situations in which the players are already fully developed. Refer to the section on *scenarios*.

2.1.2 Non-Gameplay Specific Questions

This section of the FAQ deals with anything not related to general gameplay aspects of Freeciv21.

Longturn Multiplayer

How do I play multi-player?

You can either join a network game run by someone else, or host your own. You can also join one of the many games offered by the Longturn community.

To host your own game, we recommend starting a separate server by hand. See *Server Manual* for more information.

If all players are on the same local area network (LAN), they should launch their game interfaces, choose *Connect to Network game* and then look in the *Internet Servers* section. You should see the existing server listed. Double-click on it to join.

To play over the Internet, players will need to enter the hostname and port into their game interfaces. The game admin will need to tell the other players those details. To join a Longturn.net server you start by clicking *Connect to Network Game* and then in the bottom-left of the dialog fill in the *Connect*, *Port*, and *Username* fields provided by the game admin. Once ready, click the *Connect* button at the bottom-right, fill in your Longturn.net password in the *Password* box and you will be added to the game.

Note: Hosting an Internet server from a home Internet connection is often problematic, due to firewalls and network address translation (NAT) that can make the server unreachable from the wider Internet. Safely and securely bypassing NAT and firewalls is beyond the scope of this FAQ.

Where do I see how much time is left in the current turn?

On the *Minimap* in the bottom right of the map view, where the *Turn Done* button shows for single-player games. For Longturn multi-player games will also add a count-down timer to show when the turn will change.

When connecting to a game, is the username field case-sensitive?

Yes, both the username and password is case-sensitive.

After typing in the hostname, port, and username, the password field is greyed out. What is up with that?

You have to click the *Connect* button to ask the game interface to connect to the server and then you enter your password after connecting to authorize your entry into the game.

How do I take over an AI player?

On the *server/chatline* you use the `/take <playername>` command to take over an *AI* player. You can also right-click on the player you wish to take on the *players list table*.

How do I take over an idle player that was assigned to me?

Same procedure as *How do I take over an AI player?* above.

Does capturing work like MP2?

Unit capturing is ruleset defined. Capturing in *LTT* works slightly differently than in the *MP2* ruleset. You can capture any “capturable” unit with a “capturer” unit, if the target is alone on a tile. Units that are “capturable” have a mention of this in their help text. Units that are “capturers” also have a mention of this in their help text.

Tip: Due to the game interface mechanics, you can capture units from boats. This cannot be done using the regular *Goto* command, but has to be done using the number pad on your keyboard.

Where do I go to see the rules for a game? Like how big a victory alliance can be?

All rules and winning conditions are posted to the <https://forum.longturn.net/index.php> under the Games index. Each game has a section for varying posts related to the game. Winning conditions are also often posted on the Longturn Discord <https://discord.gg/98krqGm> in the channel for the game.

Does the Nations view show whether the player is idling?

Yes, you may have to enable the visibility of the column. Right-click the header bar to see what columns are enabled. You are looking for the column named `idle`.

Can you make hideouts in LTT?

Hideouts are a purely *FCW* thing. There is no such thing in *LTT*. Other rulesets could offer this as it is a ruleset configurable item. The Longturn Community does not like them as they are overpowered and easily exploited.

How does research in LTT compare to MP2a/b/c?

MP2a/b/c and *LTT* are all different rulesets, so obviously this is a ruleset configured item. In MP2a/b, all bulbs carry over to the new research. In MP2c, bulbs researched towards a technology stay with that technology. In *LTT*, bulbs do not stay with a particular technology. There is a 10% penalty when switching research. This penalty is processed at *TC*, so if you change your research again within the same turn, you do not suffer any additional penalties. If you then change your research back to the original technology within the same turn, you do not suffer the 10% penalty.

Is stack kill enabled in LTT?

Yes, it is. This is a game server setting and is enabled on *LTT* games as without it a player could bring a stack of 100 units onto the same tile. With stack kill enabled, it eliminates this very overpowered capability.

Is it really so that in LTT there is no rapture, but you get a trade bonus in celebrating cities instead?

The concept of “rapture” is a ruleset and server configured item. The *LTT* ruleset does not do rapture. Instead “celebration” is used under *Republic* and *Democracy*. In the *LTX* Ruleset the *Federation* government also allows for celebration. Under other governments, celebration does not provide any bonuses.

Are trade routes enabled in LTT?

Technically yes, they are enabled. However in reality they are not enabled, because the required city to city distance is 999. They are overpowered and would cause game balance issues in the multi-player environments targeted by *LTT*.

Game Interface Configuration

How do I make the font bigger for help text?

You can change a collection of fonts and font sizes by going to *Game* → *Set local options* and then clicking on the *Fonts* tab.

Is it possible to save login info in the game so it does not have to be entered each time?

Yes, you can set a number of items by going to *Game* → *Set local options* and then clicking on the *Network* tab. You can set the server, port and username. You cannot save the password as that is a security risk.

Where can I turn off “connected / disconnected” messages filling up the chat window?

You can adjust a collection of things by going to *Game* → *Messages*. Anything checked in the out column will go to the *server/chatline* widget of the game interface. Anything in the mes column will show in the *messages* widget. Lastly, anything checked in the pop column will produce a pop-up window message.

Many players actually enable a lot of things that normally show in the *messages* widget and put them in the *server/chatline* widget as well. You can copy text from the *server/chatline*, but can not in *messages*. Being able to copy and paste text to your allies comes in very handy.

Refer to *Game Message Options* for more information.

How do I enable/disable sound or music support?

The game can be started without sound by supplying the command-line arguments `-P none`. The default sound plugin can also be configured in the game settings by going to *Game* → *Set local options* and then clicking on the *Sound* tab.

If the game was compiled with sound support, it will be enabled by default. All pre-compiled packages provided by the Longturn community come with sound support enabled.

How do I use a different tileset?

If the tilesets supplied with Freeciv21 do not do it for you, some popular add-on tilesets are available through the *modpack installer utility*. To install these, just launch the installer from the Start menu, and choose the one you want; it should then be automatically downloaded and made available for the current user.

If the tileset you want is not available via the modpack installer, you will have to install it by hand from somewhere. To do that is beyond the scope of this FAQ.

How do I use a different ruleset?

Again, this is easiest if the ruleset is available through the *modpack installer utility* utility that is shipped with Freeciv21.

If the ruleset you want is not available via the modpack installer, you will have to install it by hand from somewhere. To do that is beyond the scope of this FAQ.

Community

Does Freeciv21 violate any rights of the makers of Civilization I or II?

There have been debates on this in the past and the honest answer seems to be: We do not know.

Freeciv21 does not contain any actual material from the commercial Civilization games. The Freeciv21 maintainers have always been very strict in ensuring that materials contributed to the Freeciv21 distribution or Longturn website do not violate anyone's copyright. The name of Freeciv21 is probably not a trademark infringement. The user interface is similar, but with many (deliberate) differences. The game itself can be configured to be practically identical to Civilization I or II, so if the rules of a game are patentable, and those of the said games are patented, then Freeciv21 may infringe on that patent, but we do not believe this to be the case.

Incidentally, there are good reasons to assume that Freeciv21 does not harm the sales of any of the commercial Civilization games in any way.

How does Freeciv21 relate to other versions of Freeciv?

Freeciv21 is a code fork of Freeciv and is maintained by a community of online players called Longturn. After using legacy Freeciv for many years for our multi-player games, the Longturn Community decided to fork Freeciv because we felt that the development was not going in the right direction for multi-player games. Legacy Freeciv is concentrating on single-player games for the most part.

Besides Freeciv21 and legacy Freeciv, there are also communities playing a version running in the browser, commonly known as Freeciv Web. This version is less flexible and does not fulfill the needs of a diverse community like Longturn.

Where can I ask questions or send improvements?

Please ask questions about the game, its installation, or the rest of this site at the Longturn Discord Channels at <https://discord.gg/98krqGm>. The #questions-and-answers channel is a good start.

Patches and bug reports are best reported to the Freeciv21 bug tracking system at <https://github.com/longturn/freeciv21/issues/new/choose>. For more information, have a look at *Filing Bugs and Enhancement Requests*.

Technical Stuff

I have found a bug, what should I do?

See the article on *Where can I ask questions or send improvements?*. You might want to start up a conversation about it in the Longturn Discord channels if you are unsure.

I have started a server but the game cannot find it!

By default, your server will be available on host `localhost` (your own machine) and port `5556`. These are the default values your game uses when asking which game you want to connect to.

If you do not get a connection with these values, your server is not running, or you used `-p` to start it on a different port, or your system's network configuration is broken.

To start your local server, run `freeciv21-server`. Then type `start` at the server prompt to begin!

```
username@computername:~/games/freeciv21/bin$ ./freeciv21-server
This is the server for Freeciv21 version 3.0.20210721.3-alpha
You can learn a lot about Freeciv21 at https://longturn.readthedocs.io/en/latest/index.html
[info] freeciv21-server - Loading rulesets.
[info] freeciv21-server - AI*1 has been added as Easy level AI-controlled player.
[info] freeciv21-server - AI*2 has been added as Easy level AI-controlled player.
[info] freeciv21-server - AI*3 has been added as Easy level AI-controlled player.
[info] freeciv21-server - AI*4 has been added as Easy level AI-controlled player.
[info] freeciv21-server - AI*5 has been added as Easy level AI-controlled player.
[info] freeciv21-server - Now accepting new client connections on port 5556.

For introductory help, type 'help'.
> start
Starting game.
```

If the server is not running, you will *not* be able to connect to your local server.

If you cannot connect to any of the other games listed, a firewall in your organization/ISP is probably blocking the connection. You might also need to enable port forwarding on your router.

If you are running a personal firewall, make sure that you allow communication for `freeciv21-server` and the `freeciv21-client` to the trusted zone. If you want to allow others to play on your server, allow `freeciv21-server` to act as a server on the Internet zone.

For more information on running your own server refer to *Server Manual*.

How do I restart a saved game?

If for some reason you cannot use the start-screen interface for loading a game, you can load one directly through the game or server command line. You can start the game, or server, with the `-f` option, for example:

```
$ ./freeciv21-server -f freeciv-T0175-Y01250-auto.sav.xz
```

Or you can use the `/load` command inside the server before starting the game.

The server cannot save games!

In a local game started from the game interface, the games will be saved into the default Freeciv21 save directory (typically `~/.local/share/freeciv21/saves`). If you are running the server from the command line, however, any game saves will be stored in the current directory. If the `autosaves` server setting is set appropriately, the server will periodically save the game automatically, which can take a lot of disk space. The frequency is controlled by the `saveturns` setting. In any case, you should check the ownership, permissions, and disk space/quota for the directory or partition you are trying to save to.

Where are the save games located by default?

On Unix like systems (e.g. Linux), they will be in `~/.local/share/freeciv21/saves`. On Windows, they are typically found in the `Appdata\Roaming` User profile directory. For example:

```
> C:\Users\MyUserName\AppData\Roaming\freeciv21\saves
```

You could change this by setting the `HOME` environment variable, or using the `--saves` command line argument to the server (you would have to run it separately).

I opened a ruleset file in Notepad and it is very hard to read

The ruleset files (and other configuration files) are stored with UNIX line endings which Notepad does not handle correctly. Please use an alternative editor like WordPad, notepad2, or notepad++ instead.

What are the system requirements?

Memory

In a typical game the server takes about 30MB of memory and the game interface needs about 200MB. These values may change with larger maps or tilesets. For a single-player game you need to run both the game interface and the server.

Processor

We recommend at least a 1GHz processor. The server is almost entirely single-threaded, so more cores will not help. If you find your game running too slow, these may be the reasons:

- **Too little memory:** Swapping memory pages on disc (virtual memory) is really slow. Look at the memory requirements above.
- **Large map:** Larger map does not necessary mean a more challenging or enjoyable game. You may try a smaller map.
- **Many AI players:** Again, having more players does not necessary mean a more challenging or enjoyable game.
- **City Governor (CMA):** This is a really useful agent which helps you to organize your citizens. However, it consumes many CPU cycles. For more information on the CMA, refer to *Citizen Governor (aka Citizen Management Agent, or CMA)*.
- **Maps and compression:** Creating map images and/or the compression of saved games for each turn will slow down new turns. Consider using no compression.
- **Graphic display:** The game interface works well on 1280x1024 or higher resolutions.
- **Network:** Any modern internet connection will suffice to play Freeciv21. Even mobile hot-spots provide enough bandwidth.

Windows

How do I use Freeciv21 under MS Windows?

Pre-compiled binaries can be downloaded from <https://github.com/longturn/freeciv21/releases>. The native Windows packages come as self-extracting installers.

OK, I have downloaded and installed it, how do I run it?

See the document about *Windows Installation*.

macOS

Pre-compiled binaries in a *.dmg file can be downloaded from <https://github.com/longturn/freeciv21/releases>.

2.2 How to Play Freeciv21

If you are looking for how to install Freeciv21, look in *Installing Freeciv21*. If you are looking for how to get Freeciv21 running, look in the *Frequently Asked Questions (FAQ)*.

If you have never played any of the retail Civilization games, it is easiest to start by reading the legacy Freeciv Manual, available separately at: <http://www.freeciv.org/wiki/Manual>

If you are looking for help on playing Freeciv21, then keep reading!

2.2.1 Basic Strategy

First of all, this is not a *perfect* strategy, but it will get you started playing Freeciv21. Part of the appeal of Freeciv21 is in developing new strategies as you play more games. This guide is mostly based upon the Classic ruleset. There are other rulesets out there that ship with the game, but the base information presented here is the same for all rulesets.

The game is divided into several steps or stages:

1. *Choosing the first city site.*
2. *Selecting your first city locations.*
3. *Selecting your technological advance.*
4. *Mapping the countryside.*
5. *Defending the cities.*
6. *Deciding which units to build first.*
7. *Improving the land.*
8. *Deciding where to build cities.*
9. *Taking care of the cities.*
10. *Interacting with other players (diplomacy).*
11. *Exploring the world.*
12. *Things to keep in mind.*

13. *Production phase.*
14. *Annihilation of your opponents.*

Choosing the First City Site

Start the game by wandering around **briefly** to find a good place to build the first city. Do not feel tempted to investigate any yellow roofed villages yet, they might contain Barbarian tribes. Build the city close to Resources (Fish, Wheat, Wine, or Buffalo) and perhaps close to fresh water (River or Lake). Keep in mind that the first city you build will be your capital! It is advised not to place your capital on the coast. That can open you up to losing it from an attack from the sea.

The idea is to balance the quality of the site you find against getting your first city established as early as possible. You will find that *balance* becomes a key aspect of playing Freeciv21.

Selecting your First City Locations

In the Classic ruleset, you start the game with 5 Settlers, 6 Workers, and 2 Explorers. Once you have planted your first city as your capital, you need to find a place for your remaining Settlers as quickly as possible. This initial expansion phase of the game is important to establishing your base cities. Cities are everything in Freeciv21. The more cities you have, the more of every type of resource you can produce. Winning in Freeciv21 is a factor of your nation's ability to out-produce your opponents.

As mentioned in *Choosing the First City Site* above, you want to try and place your Settlers so you can create cities that have Resources available to them. These tile enhancements give bonuses to help your nation grow.

Selecting your Technological Advance

With your first city built, your nation will start to produce scientific output called Bulbs. Bulbs are used to study technological advances. As your nation learns new technologies, new units, city improvements, and wonders become available to build in your cities.

There are too many scenarios of which technological advance path to take, so make your best judgement. You will probably want to learn a new form of government. You start the game in Despotism in the Classic ruleset. Monarchy or The Republic are good early targets. As soon as you have researched a new government type, start a revolution (*Civilization* → *Government* → *Revolution*) and change over to it. Cities operate much better under Republic than they do under Despotism, but note that it is much harder to keep military units outside of city limits under a Republic. Also, do not forget to recheck your *national budget* after you have changed governments, as the maximums vary for each type.

Depending on the amount of fresh water you have available, you might need to make a push for Construction so you can get your cities larger than size 8 by building an Aqueduct. If you are on an island (see *Mapping the Countryside*), you might need to expand. Learning Map Making will give you the Trireme unit that can set sail to find new lands (see *Exploring the World*).

Mapping the Countryside

After the capital city has been built, it will start producing a Warrior unit by default. These units can be used to walk around exploring the countryside. However, the Classic ruleset gives you an excellent unit to explore with. Aptly named the Explorer. Remember, it is risky to leave a city undefended, so perhaps keep the first Warrior in the city, and use the Explorer to explore.

You will come across Huts on the game map. There is a risk / reward with walking your Explorer into one of these Huts. The rewards can be great: gold, free units, free Settlers, and even free technology advances. However, the risk is that you lose Barbarian hordes that swoop through your new, undefended cities like a hot knife through butter. You have been warned.

Defending the Cities

At this point, defend your cities by always leaving a Warrior unit in them. As your civilization develops and new units become available, replace the obsolete units with new ones to ensure your cities have sufficient defense. The units most often used for defense are (in order of strength): Warrior, Phalanx, Pikemen, Musketeers, Riflemen, and Mechanized Infantry.

Keep in mind that with some types of government, military units inside or outside cities can reduce or create unhappiness. Also remember that when a land unit is inside a city, it gets a 50% defensive bonus, as if fortified (*Unit* → *Fortify*). This happens automatically so you can tell the unit to “sentry” (*Unit* → *Sentry*) and watch for enemy units that come through your territory. When a new city is built, the city starts to build the best available defensive unit from the above list by default.

Deciding Which Units to Build First

After building one or two Warrior units, start building Settlers. Settlers take city population, but if you picked a good city site, your city should be big enough by now. The amount of population that it takes to build a Settler is ruleset dependent. In the Classic ruleset it takes 1 population to build Settlers. Settlers are best put to use building new cities, while they can also build infrastructure improvements (see the next section), it is better to use Workers for this when available. You start the game with 6 Workers in the Classic ruleset. It should be noted that unlike Settlers, Workers do not consume food from their parent city. Keep in mind that a large population increases both the amount of productivity and your civilization’s research rate, not to mention that cities secure land for your empire, so building new cities should be a high priority in the early game.

Improving the Land

Each city has an area of land around it that can be used for growing food, producing goods, and generating trade. This area is called the *Working Radius*. This output can be increased by using Workers (or Settlers, and later Engineers) to improve the land close to your cities. The land can be improved with Irrigation (increasing food output), Roads (allowing units to move faster and in some cases increasing trade), and Mines (increasing production), among other improvements. The ability to do some improvements on some tiles may require a technology advance to be learned by your scientists. See *Help* → *Terrain* → *Terrain Alterations* for more information.

Deciding Where to Build Cities

The best location for a city is a matter of taste. A city which is placed near the sea is easier to spot by opponents, but can also serve as a port for seagoing units. They also usually need a Coastal Defense city improvement later on for defense against an attacking navy. The best strategy is to build a few of both, but keep in mind that your opponents will find it harder to locate your city if you do not build it by the sea.

Now is a good time to talk about city planning strategies. There are many strategies to city planning that are good to discuss here in the early game.

Note: There are a few different kinds of Tilesets available in Freeciv21. For sake of example we are talking about the Hex(agonal) type of Tileset in this section. The concepts are very similar with the Square tilesets as well. If you are looking at Square tiles, you can change to Hex tiles by going to *Game* → *Load Another Tileset* and picking *Hexempio* from the list.

Let us first talk about two concepts: *Vision Radius* and *Working Radius*. Each city has a vision radius and a working radius. The vision radius is the tiles that the citizens of a city can see on the map. If an enemy unit comes within the vision radius of a city, a sentried unit inside of the city will wake up and alert you. Within the vision radius is the smaller working radius. When thinking of city planning, you should be most concerned with the working radius of a city. These are the tiles that the citizens of your cities manage to extract Resources (Shields, Food, and Trade). When the working radius of a city overlaps with that of another city, only one city at a time can manage that tile. This is another one of those *balance* items in playing Freeciv21.

When you have a Settler selected, it will have a red outline shown on the map. This is the city's working radius if you placed the Settler at that spot with the *Work* → *Build City* command.

This image represents a city with its working radius. The bolded hexagon represents the city center.

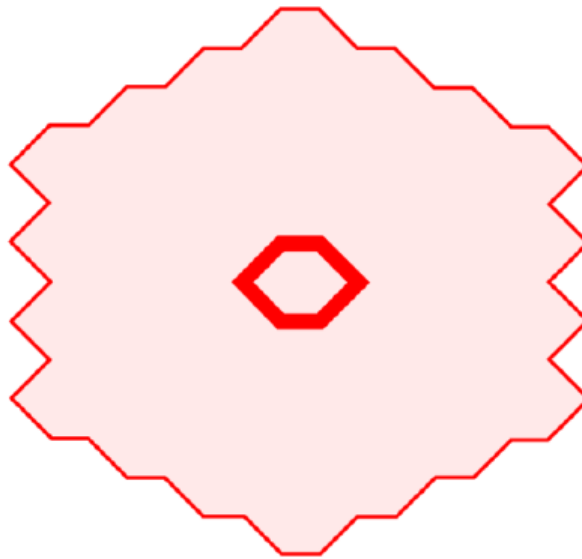


Fig. 2.1: City Center Working Radius

There are two approaches to positioning cities: Smallpoxing and Largepoxing. In Smallpoxing you fit as many cities as you can anywhere you can place them. Right at the minimum distance from each other. Here is an example of what that looks like:

Notice the overlap in the working radius of all the cities.

With the Largepoxing strategy, you give maximum space to all cities with no tile overlap. Here is what that looks like:

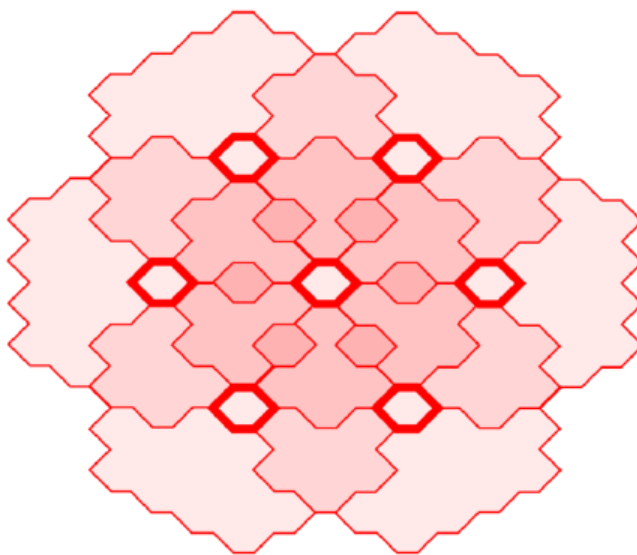


Fig. 2.2: Smallpoxing

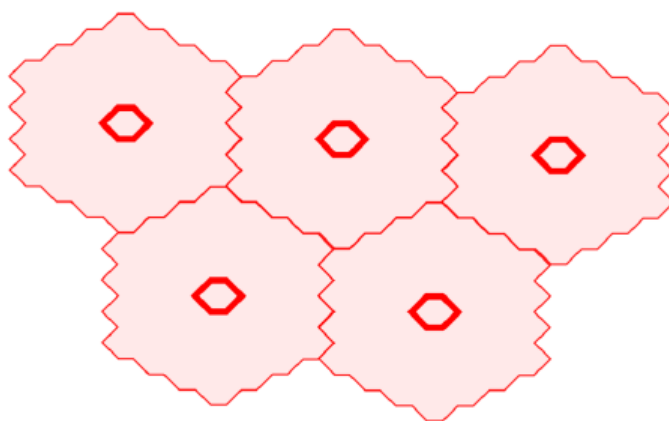


Fig. 2.3: Largepoxing

The two strategies are antonyms of each other. In Largepoxing your cities can get very large in the late game, but at the beginning of the game you will have a lot of land that is not being used. Another challenge is it takes more moves for your Settlers to reach the planned city center of the city. In Smallpoxing you only have to move a few tiles. With Largepoxing you have to move twice the distance and time (in turns).

There is a third strategy. It is not called “Midpoxing” per se, but the strategy is similar to a middle ground between the Smallpoxing and Largepoxing strategies. Here is an example of a Midpoxing strategy:

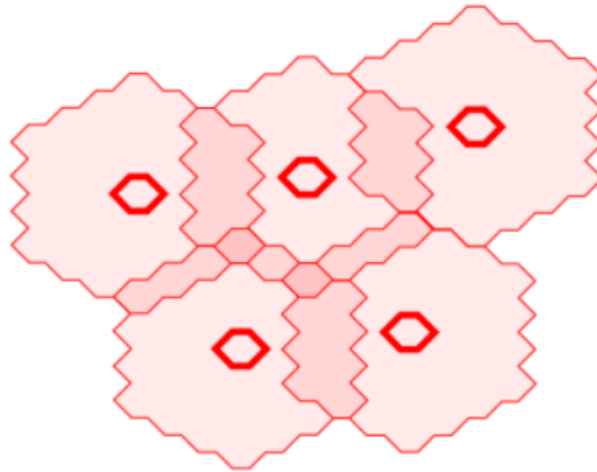


Fig. 2.4: Midpoxing

Recall that there are a number of factors to consider when it comes to city planning and placement:

1. Distance from the Capital.
2. Available Resources on tiles around the potential city centers.
3. Locations of freshwater from rivers and lakes.
4. No wasted tiles. Force overlap of any kind so that every tile is available.

Taking Care of the Cities

Every city has a group of citizens. The number of citizens depends on the city’s population. When you click on a city, the *City Dialog* will open and show how the land around the city is being managed. You can assign the city’s citizens to manage the land, or they can be specialists that contribute to your civilization in other ways. Especially at the start of the game, care should be taken to ensure that the citizens are employed so that they maximize food, trade, and then production.

If too much food is being generated, a citizen can be taken off the land by clicking on the occupied land tile. This citizen can then be transferred to a specialist. There are three specialists: Taxmen (collect gold), Entertainers (create happiness), or Scientist (create bulbs). By default, Freeciv21 will add an Entertainer when you take a citizen off the land.

If you click on the central tile of the land (the “City Center”), the citizens will be rearranged to maximize food production. You will also want to look at the level of food in the Granary and the amount of surplus food the city is producing each turn. The city will lose excess food at turn change. For example: The Granary in the city has 18/20 food (needs two food to grow to next city size) and the city is currently producing +4 food surplus each turn. This means that at turn change you will lose 2 food as waste at turn change. This is a good opportunity to move the citizens around to get food to +2 surplus. This could be accomplished by taking a citizen off a tile producing 2 food and turn it into a Taxman for a turn to get gold. At turn change, open the city and restore the citizen to farming. This kind of individual city management style is called “micro-management” and is a very powerful mechanism of game play.

The golden rule of taking care of a city is that there should be at least as many happy citizens as unhappy citizens. A city where this is not the case falls into disorder. Such cities are labeled with a raised fist or a lightning bolt (depending on the tileset). Take care not to let this happen to any of your cities, as cities in disorder produce nothing, and are prone to revolt.

Interacting with Other Players (Diplomacy)

When one of your units first meets a unit of another nation, or finds one of their cities (or equally if they find you), a basic contact is established between the two nations. This provides each with basic intelligence about the other, which can be accessed from the *Nations and Diplomacy View*.

This communication will lapse after a ruleset defined number of turns with no contact, which is one turn for the Classic ruleset. Establishing an embassy will give a more permanent communication channel, as well as more advanced intelligence such as details of technology. Embassies are one-way, the nation hosting the embassy receives no benefit and once established, cannot be revoked.

If you are in contact with another player, then you can arrange a diplomatic meeting. From the *Nations and Diplomacy View*, this is done by selecting the nation with whom you wish to meet and clicking *Meet*. If the entry under the embassy column is not blank and the other player is connected (or is a server AI) then a treaty dialog will pop up.

In this dialog you can negotiate an exchange of assets (maps, vision, advances, cities, or gold), embassies, or relationship pacts such as a Cease-fire or Peace. The list of items that can be traded through diplomacy is ruleset dependent. There is no need to trade like for like. You can trade, say, an advance or city for gold, you can consider this buying and selling. Each player builds a list of offered items. To remove an item from the list, double-click on it. When both players indicate satisfaction, the pact is concluded.

Pacts affect where your units can go and what they can do, and a pact with one nation can affect your relations with others. Under authoritarian governments such as monarchy you can break a pact at any time, but the representative governments (Republic and Democracy) have a senate which will block the unprovoked cancellation of a treaty, unless a foreign Diplomat or Spy sparks a diplomatic incident. The only way to dissolve a pact in this situation is to dissolve your government by going into anarchy. The details of pacts are described in the *Help -> Diplomacy* section.

A few notes:

1. You cannot give away your capital.
2. You can only request property that you know about; so you cannot request technology unless you have an embassy, and if you cannot see a city on your map, then you cannot request it. Of course, the owner of that city can still offer it to you, in which case the area around the city is shown on your map before you accept the treaty.
3. One important thing to note: when a city is transferred, any units in the field and supported by that city are also transferred, except those sitting in other cities. So make sure the other player is not getting a better deal than you expect.

Exploring the World

After you have fortified your cities with troops, build Triremes in the cities near the sea. Use the Triremes to map the world in search of your opponents and new lands. If you are on an island, you should spend less on military and more on expansion. Or if you are located close to an opponent, it is truly a good idea to make a Peace treaty and share the advances you have made. Diplomatic units are very useful here, and **will** pay off later.

Things to Keep in Mind

- What the next advance you will need is.
- What your tax, luxury goods and research rates are currently set to.
- Treaties are often broken, so do not neglect defense!
- Some wonders can be made obsolete by a new technology.

Production Phase

At some point you will have a large number of cities and your nation has learned many technologies. As you move into the industrial age you can build Factories and Power Plants in your cities. You want to get as much production as possible out of each city. Pollution becomes a problem. As soon as you can, try to research Mass Production for Mass Transits, and Recycling so you can build Recycling Centers. Once you have got all your cities going strong, you must build military units.

Note: If you come into contact with another player, you should immediately build a few attack units, and at least one defense unit per city.

When you want to begin thinking about attacking someone, set science to 0%, and raise taxes as high as you can without provoking disorder. Remember, money can build units too!

Annihilation of Your Opponents

This can happen at any time, but it is more fun with the advanced weapons.

Pick a relatively weak enemy, and send over a few ship loads of troops. Take over their cities, and use them to build more units to take out the rest of them with. Show no quarter! To the death!

Repeat as often as necessary!

Note: For pacifists: Freeciv21 also allows a player to win by building and launching a spaceship which arrives at Alpha Centauri before anyone else. It is known as the Space Race victory.

2.3 Citizen Governor (aka Citizen Management Agent, or CMA)

The Citizen Governor is designed to help you manage your cities, i.e. deploy the citizens on the available tiles around (or make them scientists, taxmen, or even entertainers), to get the most out of the city. You can switch the Governor on or off at any time for any city, but there are some handicaps (explained below), if you have governor controlled and non-governor-controlled cities nearby.

The heart of the Governor system is an optimizing algorithm, that tries to deploy the citizens of a city in such a way, that a user-defined goal is achieved as much as possible. You know probably, there is already a kind of optimizing, when you open a city, and click on the center tile (the city symbol) of the map. This optimization tries to maximize mostly the food output, but it does not care about disorder.

The City Management Agent goes far beyond this old form of optimizing. First, it performs this task every time anything changes with the city. If the city grows or shrinks, troops go in or out, tiles get irrigation or mining, or are occupied by an enemy, the Governor becomes active. Second, it supports all kinds of optimizing, like production (shields), gold, science, or luxury goods. Third, it gives the player a fine-grained control over this, with the possibility of setting

constraints for any kind of city output. The latter includes the constraint of celebration, which makes it very easy to let your cities grow, even in harder times. The forth, and probably most valuable thing in war times, is that it keeps your cities content, preventing them from revolt.

The legacy Freeciv Wiki also contains some other useful information related to the CMA:

- https://freeciv.fandom.com/wiki/City_manager
- <https://freeciv.fandom.com/wiki/CMA>

2.3.1 Usage

You can set up the Governor for a city by opening the *City Dialog* and clicking on the *Governor* tab. You can choose a preset for a specified goal in the middle and on the bottom you can specify more complex goals by moving the sliders. You can choose a preset at first, and then modify it. Once you have created a new setting, you can add a preset name for it. This is not required, but is very useful, since you can watch and even change the city's setting from within the city view, if it is given a name. Do not forget to save settings (in the *Game* menu), when you have created new presets.

The sliders are of two kinds: the rightmost sliders are factors, which gauges how much one product is worth compared to the others (e.g. how much shields are worth with respect to everything else). The leftmost sliders are constraints. You can command the city not to lose food, e.g. by setting the surplus constraint to zero, and you can allow the city to lose gold by setting the gold surplus to -3, and urge them to make at least 5 shields per round by setting the production surplus to 5. The most powerful constraint, though, is the *Celebrate* constraint, which makes the city celebrate at once (which usually takes effect the round after you change it).

Note: Celebration is a ruleset defined setting and is not available for all games.

It is obvious that the Governor cannot fulfill all these constraints in every case. Whenever the constraints cannot be fulfilled, the Governor quits its service for that city, giving a message: **“The agent can’t fulfill the requirements for [city name]. Passing back control.”** You then have the choice of either managing the city on your own (which has some drawbacks, see below), or open that city and change the surplus requirements so that they can be fulfilled.

When you have made a setup for a city, you need to click on *Enable* to switch on the Governor. If this button's text is greyed, either the Governor is already active, or the task is impossible. In the latter case you see dashes instead of numbers in the results block. If you ever want to switch off the Governor deliberately, click *Disable*.

The *Optimize* option for Food output will ensure that the city is always growing. By enabling the *Allow Specialists* option, the Governor will pull citizens from working tiles and turn them into Entertainer, Scientist, or Taxmen specialists to meet the constraints. Lastly, enabling the *Allow Disorder* option will allow the city to go into disorder to meet constraints.

2.3.2 Advanced Usage

Usually the goal(s) of your cities depend on the phase your game is in, whether you want to spread widely, grow quickly, research advanced techs or wage war. You may want to set up a high factor for science to research, or a high shields factor to produce units. The highest factor available is 25, that means: if the shields factor is set to 25, and other to 1, the Governor prefers a single shield over 25 gold (or trade also). This is pretty much because money can buy units too. That also means that the Governor is indifferent about producing gold, science, luxury goods, or food, but when you wage war, you usually prefer gold or luxury goods. So it is probably a good idea to set a second (or even third) preference for the city's output, e.g. gold factor 5. That still prefers 1 shield over 5 gold (and 1 gold over 5 food or anything else).

Constraints are not useful in all cases. If you want a high income, it is probably better to set the gold factor to 25, than to set a minimal surplus of 5 or so. Because a big city can make more gold than a small one, you would end up setting a different surplus for each city.

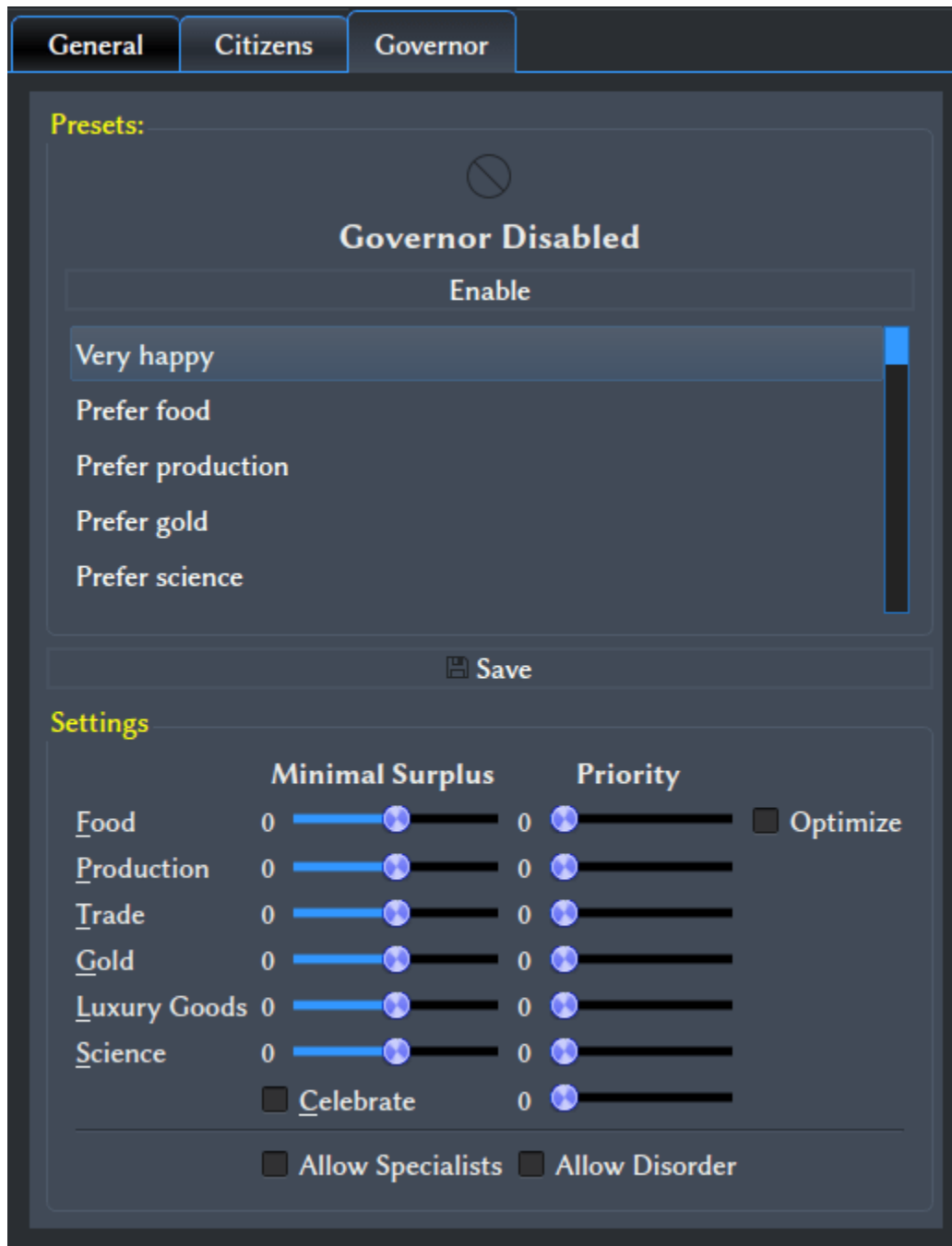


Fig. 2.5: Governor Tab in City Dialog

However, if the shields surplus of a city is below zero, it cannot support all of its units any more. You will lose some of the units the city supports. If the food surplus is negative, the city will starve and eventually (when the granary is empty) shrink. This may be intended, but if the city supports any settlers, you will lose them before the city shrinks. Constraints can also have a warning function.

Which constraints can be fulfilled depends widely on the global science, tax, and luxury good rates. E.g. a gold surplus ≥ 0 is easier to fulfill with a higher tax rate than a lower one. You should always consider to change these rates, when you going to change the Governor settings for the most of your cities.

2.3.3 Drawbacks

The Governor is a very powerful tool, which not only releases you from the micromanagement of your cities, but gives you more performance than you have ever seen (well, for most players).

There are some drawbacks, though. Once you have switched on the Governor, it grabs any good tile it can get. So you encounter very hard times trying to manage a city nearby a Governor-controlled one. This is true for the city dialog and the main map worker's interface as well. If you want to have Governor-controlled and **handmade** cities, they probably should be on different islands.

There are several situations where the Governor cannot fulfill the requirements just temporarily, e.g. when you move a ship from one city to another, or when an enemy walks through your country. The Governor passes back control in these cases, and you have to reenable it manually. A general approach to prevent this might be, to set the minimal surpluses as low as possible (-20). Of course you must be careful with the food and shield surpluses.

While the Governor does a really good job for a single city, no tile will ever be released for the good of another city. Also, the Governor controlled cities are computed in a more random order. The results may depend on it and change, when a recalculation is done (e.g. when tax changes). So, no guarantee is given that the overall results are always optimal.

2.3.4 Settings file

The game allows the user to load and save preset parameters for the agent. Choosing *Game* → *Options* → *Save Settings Now* will not only save your *interface options* and *message options*, but it will save any changes you made to your Governor presets as well.

The format for the options file (usually `~/.local/share/freeciv21/freeciv-client-rc-X.Y`, where X.Y is the version of Freeciv21 in use) is as follows (in case you wish to change these presets manually, i.e. with a text editor).

Under the heading `[cma]`, is a `number_of_presets`. This should be set to the number of presets that are present in the options file. If you manually add or remove a preset, you need to change this number as appropriate.

After this, is an array that houses the presets. Here is the header:

```
preset={ "name","minsurp0","factor0","minsurp1","factor1","minsurp2",
"factor2","minsurp3","factor3","minsurp4","factor4","minsurp5",
"factor5","reqhappy","factortarget","happyfactor"
```

so the order of the preset should be as follows:

- name of preset, minimal surplus 0, factor 0, ... ,
- require city to be happy, what the target should be [0,1],
- the happiness factor

Currently there are 6 surpluses and factors. They are:

- 0 = food

- 1 = production
- 2 = trade
- 3 = gold
- 4 = luxury goods
- 5 = science

Also currently, `factortarget` is not changeable within the client.

The array should be terminated with a squirely brace `}`.

Here are the 5 presets that come with Freeciv21 out of the box:

```
"Very happy",0,10,0,5,0,0,-20,4,0,0,0,4,FALSE,25
"Prefer food",-20,25,0,5,0,0,-20,4,0,0,0,4,FALSE,0
"Prefer production",0,10,-20,25,0,0,-20,4,0,0,0,4,FALSE,0
"Prefer gold",0,10,0,5,0,0,-20,25,0,0,0,4,FALSE,0
"Prefer science",0,10,0,5,0,0,-20,4,0,0,0,25,FALSE,01
```

Here are 16 more that you can add to your client RC file:

```
"+2 food",2,1,0,1,0,1,0,1,0,1,0,0,1
"+2 production",0,1,2,1,0,1,0,1,0,1,0,0,1
"+2 trade",0,1,0,1,2,1,0,1,0,1,0,0,1
"+2 gold",0,1,0,1,0,1,2,1,0,1,0,0,1
"+2 luxury",0,1,0,1,0,1,0,1,2,1,0,0,1
"+2 science",0,1,0,1,0,1,0,1,0,1,2,1,0,0,1
"+20 Celebrating for Gold",20,0,0,16,0,0,0,8,0,1,0,1,TRUE,0
"Max food no gold limit",0,10,0,1,0,1,-20,1,0,1,0,1,0,0,1
"Max production no gold limit",0,1,0,10,0,1,-20,1,0,1,0,1,0,0,1
"Max trade no gold limit",0,1,0,1,0,10,-20,1,0,1,0,1,0,0,1
"Max gold no gold limit",0,1,0,1,0,1,-20,10,0,1,0,1,0,0,1
"Max luxury no gold limit",0,1,0,1,0,1,-20,1,0,10,0,1,0,0,1
"Max science no gold limit",0,1,0,1,0,1,-20,1,0,1,0,10,0,0,1
"Max food+prod. no gold limit",0,10,0,10,0,1,-20,1,0,1,0,1,0,0,1
"Max food+prod.+trade",0,10,0,10,0,10,0,1,0,1,0,1,0,0,1
"Max all",0,1,0,1,0,1,0,1,0,1,0,1,0,0,1
```

Here are 6 more that have been added as an afterthought:

```
"+1 food, max prod. no gold limit",1,1,0,10,0,1,-20,1,0,1,0,1,0,0,1
"+2 food, max prod. no gold limit",2,1,0,10,0,1,-20,1,0,1,0,1,0,0,1
"+3 food, max prod. no gold limit",3,1,0,10,0,1,-20,1,0,1,0,1,0,0,1
"+4 food, max prod. no gold limit",4,1,0,10,0,1,-20,1,0,1,0,1,0,0,1
"+5 food, max prod. no gold limit",5,1,0,10,0,1,-20,1,0,1,0,1,0,0,1
"+6 food, max prod. no gold limit",6,1,0,10,0,1,-20,1,0,1,0,1,0,0,1
```

and even more, some with multiple goals:

```
"research at any cost",0,1,0,5,-20,1,-20,1,-20,1,-20,25,0,0,1
"celebration and growing",1,1,0,25,-20,1,-20,12,-20,1,-20,1,1,0,1
"grow at any cost",1,25,0,5,-20,1,-20,1,-20,1,-20,5,0,0,1
"research and some shields",0,1,0,8,0,1,-3,1,0,1,0,25,0,0,1
"shields and a bit money",0,1,0,25,0,1,-3,3,0,1,0,1,0,0,1
```

(continues on next page)

(continued from previous page)

```
"many shields and some money",0,1,0,25,0,1,0,9,0,1,0,1,0,0,1
"shields and some research",0,1,0,25,0,1,-2,1,0,1,0,8,0,0,1
"celebrate and grow at once",1,1,0,25,-20,1,-20,1,-20,1,-20,8,1,0,1
```

Enjoy using your citizen Governors!

2.4 The Turn Change Sequence

This page has been adapted from the legacy Freeciv Wiki.

Freeciv21 is a turn-based game. A turn is a period of the game when all players (human and *AI*) are allowed to make a specific number of game actions. Longturn online multi-player games have turns that are often set to a counter of 23 hours. This allows players all over the world an opportunity to log in and make their moves at a reasonable time. Single player games (one human against the *AI*) do not have a counter set. The turn ends when all moves have been completed and then the human player will manually clicks the *End Turn* button.

Turns are typically enumerated from T1.

The period of a turn when a specific player or player group can act in a game is called a *phase*. Most often Freeciv21 is played in the concurrent mode — there is only one phase per turn for everybody (AIs do their moving only at phase beginning). But in the server options another mode is available: all players move in a queue, or all teams move in a queue. Most things happen to an empire either at the start or at the end of its phase (since the default mode is one-phase, these periods are commonly referred as *turn start* or *turn end*, but in alternate playing they will happen to different players in different time)¹. Understanding the order in which things happen can give a player considerable advantage.

Cities

Depending on the ruleset, cities will typically produce one item per turn depending on the surplus shields. Some rulesets enable a slots feature and with sufficient surplus shields can produce more than one unit per turn. Only one building can be produced per turn. Surplus shields will be saved between turns.

Units

Units have a specific number of move points (*MP*) allotted per turn. Veterancy levels can increase the number of MPs a unit is allotted per turn. Once the number of moves have been used, there are no more until the next turn. If some move points are left over at the end of the turn, they are lost.

Multi-Player vs AI

As previously noted, online multi-player games have a turn defined as a certain number of hours. This means that “moves” by a player can happen at any time during the turn. One player may login and play his moves at the beginning of the turn, another may play her moves mid-turn and a third may play near the end of the turn. The main point here is that in this scenario, just because you have made your moves does not mean that all moves for all players have completed. During single player games, the AI will make its moves at the beginning of the turn and will be finished. The human player will then make his moves.

¹ A granary influences food stock if built on the growing turn. Barracks will not regenerate all *HP* of units resting in the city the turn they are built.

2.4.1 Turn Change Events

There are a number of functions that are called at *TC* that explains the sequence of events that occurs when either a human player clicks the *End Turn* button or a Longturn multi-player game's timer expires and the turn ends.

Turn start

Function: `begin_turn()`

- A `turn_started` signal (turn, year) is emitted.
- Player scores are calculated.
- If *FOW* setting was changed, the change is applied.
- In the concurrent mode, random sequence of players is defined for the things that happen to them in a queue. Below, *FEP* means iterating players in this order (`phase_players_iterate(pplayer)` macro). In teams-alternating mode, player sequence within a team is permanent through the game. After the procedure exits, `begin_turn` packet is sent to the clients. Then, the phases of the game follow in the order of teams' or players' numbers as shown on the *Nations View*.

Phase Start

Function: `begin_phase()`

- All players get available info on each other. Without contact or embassy, it is at least dead/alive, and great wonders. Add diplomatic states with players you are in contact with.
- A `start_phase` packet is sent to the clients.
- *FEP*: Unit's activities (mining, etc.) are updated for each player's units in their default order: unit activity rate is calculated (if the unit has at least one movement fragment, the rate is proportional to its basic move rate and veteranship power factor) that is added to its activity counter.
- For performing certain activities, the unit might have a chance to become veteran.
- Unit's move points (*MP*) are restored.
- Autoexplorers explore.
- Counters of the units on the tile that do the same are summed to look if the activity is done. If so, changes are applied to the tile with side effects like bouncing ships from ocean converted to swamp.
 - Units doing the same changing activity on the tile become idle. For example, even if we have irrigated swamp to grassland and could irrigate more.
 - Adjacent units are also checked for if they have not they lost their only irrigation source.
 - Fortifying and converting units do the thing.
- *FEP*: Unit orders are executed, units are processed by the default player's unit list (not necessarily in command order).
- *FEP* (human only): Building advisor prepares advice for cities.
- *FEP*: City data are sent.
- *FEP* (alive): Revolution finishes, if it is the time and target government is specified (if not, a notification is sent).
- *FEP* (*AI*): Diplomatic meetings are considered.
- *FEP* (*AI*): Do turn-begin activities (move units, etc.)
- Players that do not have units or cities die from the game.
- The timer is started.

- Global game info is sent to the clients.

Since this moment, turn is considered new even if the game was restarted from savegame. If turn autosaves of game/map image are enabled, now one happens.

Running a phase

Function: `server_sniff_all_input()`

- Server watches for all packets clients send to it. Phase players can play until phase time runs off or every human presses *End Turn* button. After that moment, timer is cleared and phase end starts.

Phase End

Function: `phase_end()`

During phase end, server-client packages go to a buffer, that is unbuffered when the phase ends to end.

- An `end_phase` packet is sent.
- *FEP*: Techs updated:
 - If a player has not set what to research, a tech towards his or her goal is selected, or random tech (by game random tech setting) if no goal.
 - If a tech is going to be lost, future techs are reduced (if any), or random losable tech (holes allowed/not) is lost.
- At this moment, city state updates are stopped due to many things that leave them in intermediate out-of-the-game state.
- *FEP (AI)*: unit end turn *AI* activities (no movement)
- *FEP*:
 - Auto-settlers do their move to work terrain.
 - For AIs, governments, techs, taxes, cities and space program are handled.
 - The `Tech_Parasite` (Great Library) effect may bring techs known to others (depending on ruleset).
 - Auto-upgrade (Leonardo's Workshop depending on ruleset)
- For each player's unit:
 - Hit points (*HP*) regenerate/shrink;
 - If they shrink to zero or below, the unit dies;
 - Fueled units running out of fuel try to seek a resort automatically within left movepoints.
- Fueled units are refueled if possible, or their fuel is reduced, and they crash if it goes zero or below.
- Spaceship parts autoplace.
- Cities are updated. For cities in their normal order:
 - Citizen assimilation handled.
 - Traderoutes that no longer can exist are cancelled.
- Now, for the player's cities in a randomized order:
 - City is refreshed². Workers are auto-arranged if vision radius has changed significantly.
 - Unit upkeep is recalculated.
 - If something changes, the citizen workers are arranged by some default manager.

² The `City_Build_Slots` effect works for making units only.

- If not enough shields, units upkeep with them are disbanded (in city units list order). If it does not balance without touching undisbandable units, a citizen is spent on the upkeep of each such unit. The city may be destroyed in effect. If, otherwise, some surplus shields remain, they are added to the shield stock.
- The production is remembered for the case it is changed to another genus.
- City tries to produce something:
 - * For Mint-like buildings, remained shields are converted to gold, then, if something else is on the plan, the production changes.
 - * For other improvements, here production target is upgraded if it becomes obsolete (to the `replaced_by` building).
 - * If you still cannot in principle build the improvement any more (your techs do not allow it, your spaceship is finished etc.), here you get a notification and a signal is emitted.
 - * Otherwise, if your shield stock³ is greater or equal to the improvement cost:
 - For small wonders that can be built, another instance of this small wonder in the player's empire is removed.
 - For space parts, they are produced⁴. Other improvements appear in the city; wonders are updated right this moment to the cache used by requirements; for global wonders, notifications are sent to everybody. Then shield stocks are reduced on the used cost, and the `building_built` signal is emitted (as any signal, might potentially destroy the city right here).
 - City vision radius is updated.
 - Darwin's Voyage effect for the building may give techs.
 - If it was a space part, corresponding information is sent around. Otherwise, the city is refreshed^{Page 60, 2} and the citizen workers are auto-arranged if the radius has changed.
 - Production is changed according to the worklist. If no worklist, then, if the building is a special one that can be built again (Gold), it is started again, otherwise new one is chosen by the advisor.
- If a unit is produced:
 - If the production can be changed (the city has not bought the former turn) and the unit is obsolete, city switches to the obsoleting unit.
 - If the city does not fulfil the units requirements (tech, improvement, or unit has no `NoBuild` flag, etc.) and the player is not Barbarian, it is notified and a signal is emitted: city survival is not checked.
 - Otherwise, if we have enough shields to build the unit:
 - * If it is a Settlers consuming last population, we have at least two cities and the city setting enabling this is on, the city is disbanded (otherwise, a notification and a signal of not being able to produce happens), its units, including the newly created one, are transferred to the nearest city, and the city processing is finished here.
 - * Otherwise, the city remembers that we have built last this turn.
 - * If some population cost is paid, the city size is reduced and the city is updated (citizens, borders, etc.) and refreshed^{Page 60, 2} with citizen workers auto-arranging if the workable tiles change.
 - * Shields are reduced on paid cost. Notifications and `unit_built` signal are emitted.

³ Units that obsolete another but have fallen not available for e.g. government change are not "downgraded" (but also are not produced, even if you have paid for them!).

⁴ A "`unit_built`" signal is not emitted if you disband a city; `city_destroyed` with nil as the destroyer parameter is instead. By the way, the unit will have the former city's `Veteran_Build` rank as a last memory of it.

- * If we have additional building slots, the unit we build does not cost pop and is not unique, it can be produced more than one time. If the city has a worklist, to use the full `City_Build_Slots` effect, the unit should be repeated at the top of the list so many times (the positions will be removed). For floor (`shield_stock / unit_cost`), similar units are built with corresponding shield stock reduction (cycle breaks only if the city is destroyed in process).
- Here it is checked if the game is over and it is time to leave the game.
- If the city is big enough and was happy before, it celebrates and gets its rapture counter up, otherwise any celebrations are cancelled and the counter is zeroed. Then, the city's "was happy" switch is updated from its feeling level⁵.
- Plague is checked.
- City gets its food surplus into its granaries.
- If the full granary size is achieved or overdone and the plague has not just struck, or the city has rapture grow this turn, the city tries to grow. If no necessary Aqueduct, it just loses some food (but granary building effect reduces this loss); any way, all that does not fit into the (new) food stock is lost. A grown city is refreshed^{Page 60, 2}.
- City claimed borders are updated.
- Food in the city is balanced. If not enough, food-upkept units are disbanded, then a citizen is lost (may destroy the city). Granary food left after a shrink is calculated from the granary size of the reduced city but with granary preserving effect of the city before shrink.
- Sell, buy and airlift counters are cleared.
- Bulbs are harvested to the player's research. (If it has negative bulbs, a tech can be lost here, but unlikely).
- Gold is harvested to the player's national treasury, then gold upkeep to buildings and units is paid. If the treasury appears in debt after this calculation, balancing happens according to gold upkeep style (not here in style 2).
- If the city is in disorder, notification is sent and disorder turns counter is increased.
- Pollution is checked (production is calculated without disorder fine)
- If you rebel enough turns to overthrow your government, it happens.
- The city is finally refreshed^{Page 60, 2} and citizen workers are auto-arranged.
- National gold upkeeps are balanced, according to the upkeep style. Nationally supported improvements (not wonders) are put on list, and a random one is sold until the gold is positive. The same happens to gold-upkept units (transporters are disbanded only after all their cargo is checked for disbanding, units that can not be "sold" do not return their upkeep into the treasury).
- If expenses exceed 150% of the national treasury, warning "LOW on FUNDS!" is sent to the player.
- Pay tech upkeep and check for obtaining/losing techs.
- *FEP*: Refresh cities vision radius.
- At this moment, the players in "dying" state leave the gameboard.
- Internet or Apollo Program revealing effects show things (depending on ruleset).
- Marco Polo's Embassy effect gives contact to other players (depending on ruleset).
- *FEP*: Phase finished *AI* function (for human players, also needed to initialize *Citizen Governor (aka Citizen Management Agent, or CMA)*).

⁵ City happiness is not immediately updated with building a unit unless it costs population.

- Now connections are unbuffered. If the game is over for this moment, other phases of the turn are not started. After done with phases, the turn begins to end.

Turn end

Function: `end_turn()`

- An `end_turn` packet is sent.
- Borders are updated over the map.
- Barbarians are summoned.
- If migrations are enabled, they happen, and all cities are sent to players.
- City disasters.
- Global warming.
- Nuclear winter.
- Diplomatic states are updated (e.g. Cease-Fires run out). Players are iterated in a two-floor loop, on both levels in their main order. This includes moving or destroying illegally positioned units.
- Historians may do their reports.
- Settings turn.
- Voting turn.
- In-game date advances.
- Timeout is updated.
- Game and players info is sent to the clients.
- Year is sent to the clients.

Then, the metaserver info is sent. The game is checked to be over by rules or stopped manually; if so, players are ranked.

Game over

Client connections are thawed, and the turn timer is cleared. The scores are calculated and go to the scorelog. Map is shown to everybody. Server resends info to the metaserver, and saves the game on gameover.

MANUALS

The Manuals category is where we place all of our varying manuals such as the game manual, server manual, advanced topics, modpack installer, and ruleset manuals.

3.1 Game Manual

The Freeciv21 Game Client (`freeciv21-client`) is the main user interface that allows one to play single player and online multiplayer games. This manual will mostly track what is in the `master` branch of the Freeciv21 GitHub Repository at <https://github.com/longturn/freeciv21/tree/master>. You can see the date that this file was last updated at the bottom of this page in the footer.

If you have not installed Freeciv21, then obviously you need to start *there*.

If you are having trouble, come find the Longturn Community on Discord at <https://discord.gg/98krqGm>. A good place to start is the `#questions-and-answers` channel.

3.1.1 Starting the Game

Depending on how you installed Freeciv21 will determine how you start it. If you installed with one of the precompiled binary packages, then you should find Freeciv21 in your OS's launcher, Start Menu, etc. If you compiled the code yourself, then you will go to the location you asked `--target install` to place the files. Double-clicking `freeciv21-client` should start it up.

Note: You may have noticed that the version string is a bit different than other games or software. The version number for Freeciv21 is broken into a few parts: `[major version].[minor version].[hash version].[version string level]-[version string]`. In [Fig. 3.1](#) the major version is 3, minor version is 0, hash version is 699369, version string level is 7 and the version string is beta. The easiest way to “speak” the version in this example is version 3.0-beta.7. The hash version is an integer derived from the hexadecimal git hash and is generally ignored when discussing the version.

The following buttons are available on the *Start Screen*:

- *Tutorial* – Quickly start the game *Tutorial*. This is a great place to start for new players.
- *Start New Game* – Start a new single player game. See *Start New Game* below.
- *Connect to Network Game* – Connect to a Longturn multiplayer game or one you host yourself. See *Connect to Network Game* below.
- *Load Saved Game* – Load a previously saved single player game. See *Load Saved Game* below.
- *Start Scenario Game* – Start a single player scenario game. See *Start Scenario Game* below.

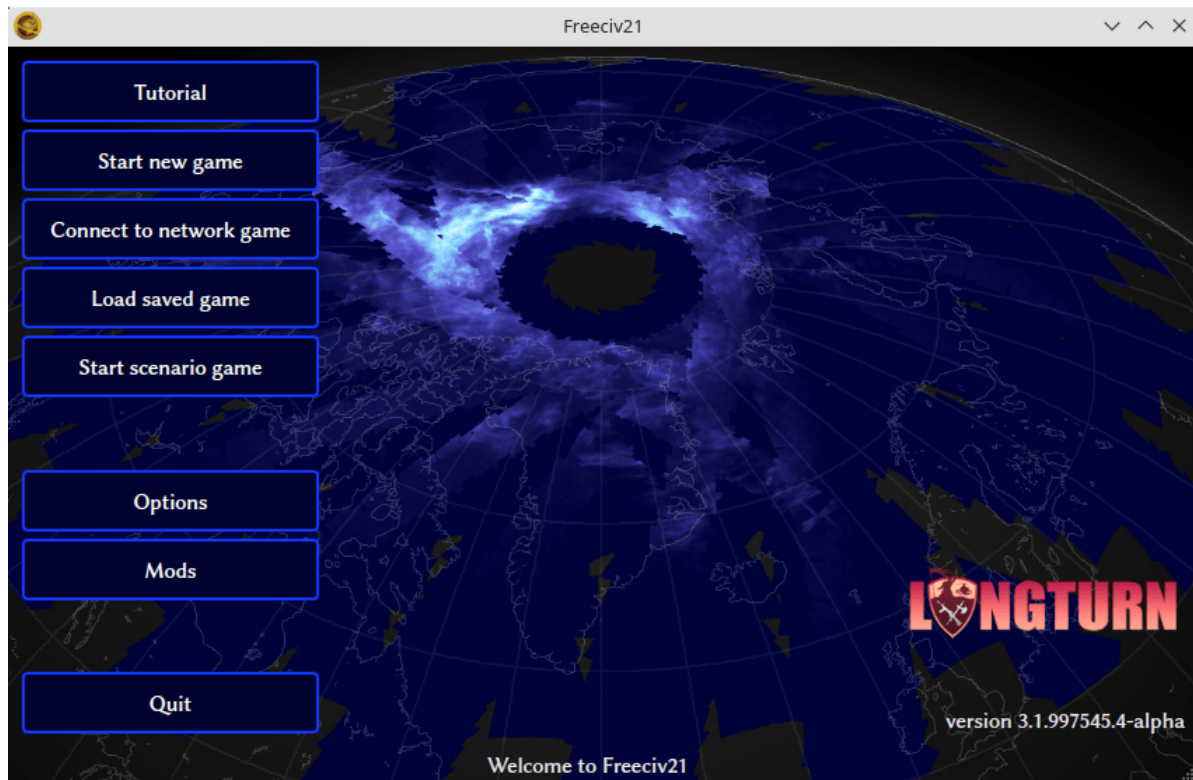


Fig. 3.1: Start Screen with NightStalker Theme

- *Options* – Set client options. See [Options](#) below.
- *Mods* – Launch the modpack installer. See [Using the Modpack Installer Utility](#).
- *Quit* – Quit Freeciv21

Note: Notice that there is not a *Help* button available. This is by design. The in-game help is compiled at run-time based on the ruleset you select and other server settings you may set.

Tutorial

After clicking *Tutorial* on the start screen, a pregame menu will show similar to [Fig. 3.2](#), below. All the settings are preconfigured. Simply click *Start* to begin the game tutorial.

Start New Game

Clicking *Start New Game* is used to start a new single player game. When clicked the following dialog will appear. From upper left to lower right, the following user interface elements are available:

- *Players List Table*
- *Nation*
- *Rules*

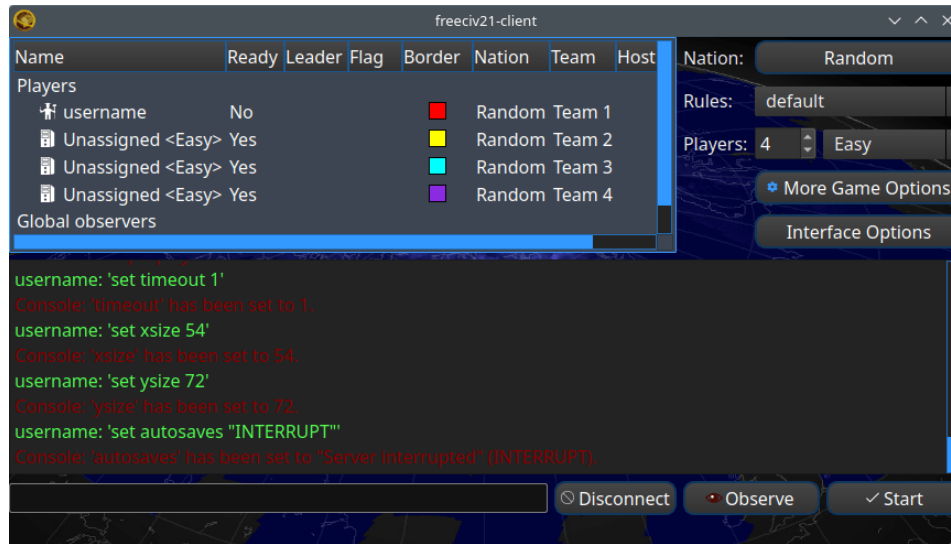


Fig. 3.2: Start New Game Dialog

- *Number of Players*
- *AI Skill Level*
- *More Game Options*
- *Interface Options*
- *Server Output Window*
- *Server Chat/Command Line*
- *Disconnect*
- *Observe*
- *Start*

Players List Table

The *Players List Table* shows information about the configured players in the game. The information shown in Fig. 3.2 is what a single player game looks like. A Longturn multiplayer game will look very similar, except that all the player's aliases will be shown as set up by the game administrator. You can right-click on a player's row to configure details about the specific player:

- *Observe* – Allows you to connect to a running game and observe that player. This is useful during Longturn multiplayer games when you want to connect and see what a player is doing, but you cannot make any actual moves for the player. This works for LAN games as well. You can also use the server chat line and issue this command: `/observe <player>`.
- *Remove Player* – Removes the player from the list.
- *Take This Player* – Allows you to claim this player as your own and then when you click *Start* you will join the game as that player. This is a required step for Longturn multiplayer games at the start of a new game. Subsequent logins to a game when you [Connect to Network Game](#) will not require another *take* action. You can also use the server chat line and issue this command: `/take <player>`
- *Pick Nation* – Allows you to choose the Nation of the player. See [Nation](#) below.

- *Set Difficulty* – Set the difficulty of the AI.
- *Put on Team* – Combine players into teams.
- *AI Toggle Player* – Toggle if the player is an AI or a human. This is needed before you can use the **take** option above as players when added are AI by default.

Nation

Clicking on the button that says *Random* as shown in Fig. 3.2 above, Freeciv21 will bring up a dialog box allowing you to pick the nation you want to play as shown in Fig. 3.3 below. Freeciv21 comes with hundreds of available nations to pick from. Each nation has a city graphics style that is automatically selected, but you can also change it. You can pick from European, Classical, Tropical, Asian, Babylonian, and Celtic. You can also change the gender of your empire's leader between male and female. Lastly you can use the built-in leader names or enter one of your choosing.



Fig. 3.3: Select Nation Dialog

Rules

Freeciv21 comes with a collection of rulesets that define the game parameters. Rulesets control all the aspects of playing a game. For more information on rulesets, you can refer to [Ruleset Modding](#).

Freeciv21 comes with the following rulesets:

- Civ1
- Civ2
- Civ2Civ3 (Default)
- Classic
- Experimental

- Multiplayer
- Royale

Number of Players

The spinner can be changed up or down to customize the number of players. The ruleset can also set the number of players, so be sure to pick the ruleset before you pick the number of players.

AI Skill Level

This box will do a mass-change for all the AIs to be the same level. If you want to customize this, then use the [Players list table](#) right-click menu.

More Game Options

Clicking on this button will bring up the *Game Options* dialog box as shown in [Fig. 3.4](#) below. From here you can customize other settings for the game before you start it. The ruleset defines many of these options, so be sure to select the ruleset you want to play before attempting to set other settings. You can hover the mouse over the entries to get an explanation of what the setting does. If you a curious person and like details, you can also read about all of them in [Server Options](#).

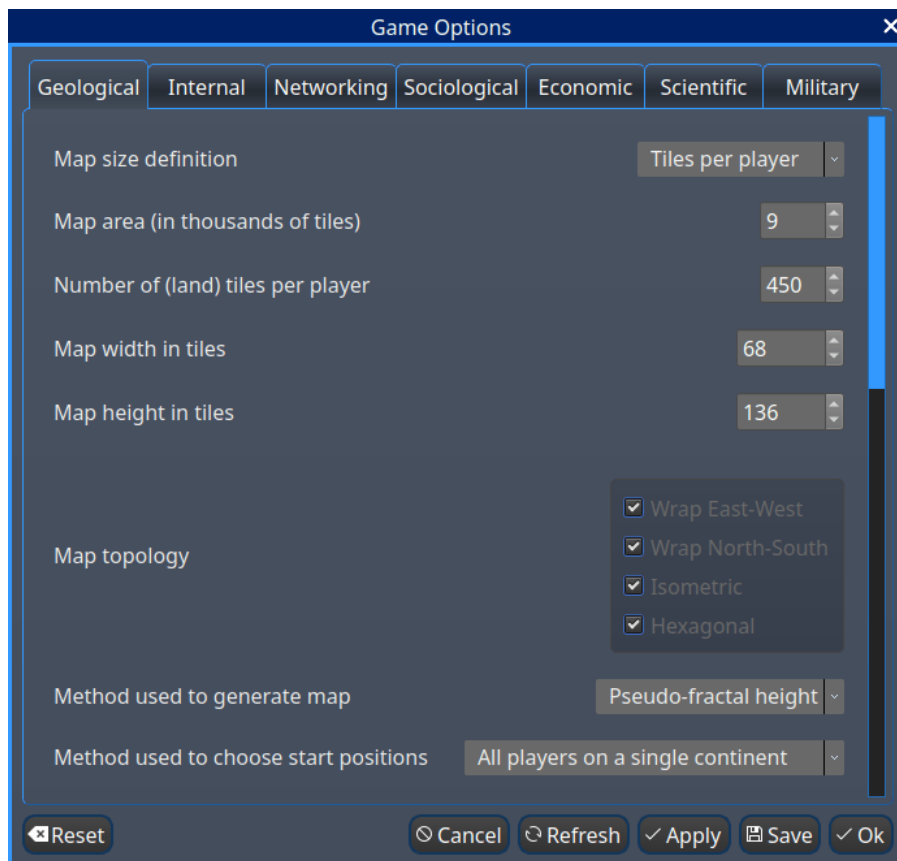


Fig. 3.4: Game Options dialog

Interface Options

Refer to the section on *Options* below.

Server Output Window

When you pick a ruleset, you will often see a bunch of output inside of this window. Also, if you make changes to the game in *More Game Options*, you will see output from those settings in this window as well. This is actually a good way to learn what the varying game options are that can be issued via the *Server chat/command line*. This window is read-only, however you can select text from it and copy it to paste in a text file if needed.

Server Chat/Command Line

The *Server Chat/Command Line* is a text box below the server output window. From here you can manually enter `/set` commands to the server directly if you know what you want to set.

Disconnect

Clicking this button takes you back to the *Start Screen* as shown in [Fig. 3.1](#). The local `freeciv21-server` instance will terminate at this time. A subsequent click of *Start New Game* will spawn a new instance of the server, however any changes made previously will be lost and you will have to start over.

Observe

This button allows you to do a **global** observe of all players. This is a special server setting and is not enabled by default, however it is available in single player games. If you are globally observing a game, you can right-click on the *Nations and Diplomacy View* button in the *Top Function Bar*, and select a particular nation to observe.

Start

When you are finished with all the settings, clicking *Start* will cause the game to start!

Connect to Network Game

When you click on the *Connect to Network Game* button, a dialog box will appear as in [Fig. 3.5](#) below.

There are three ways to connect to a server:

1. **A Local Server:** If you are hosting a local server on the same IP subnet as the client, then it will show in the *Local Server* box at the top.
2. **An Internet Server:** If the Longturn community lists running games on its internet metaserver, games will show here and can be connected to via the *Internet Servers* box in the middle.
3. **Manually:** If you simply need to connect to a remote server and you know the the *Servername* and *Port*, then this is your option.

For the first two options you will select the server in the table and ensure that your *Username* is correct and then click *Connect* to connect to the server. The *Password* box will activate when you have connected to the server. Type in your password and then click *Connect* a second time and you will join the server.

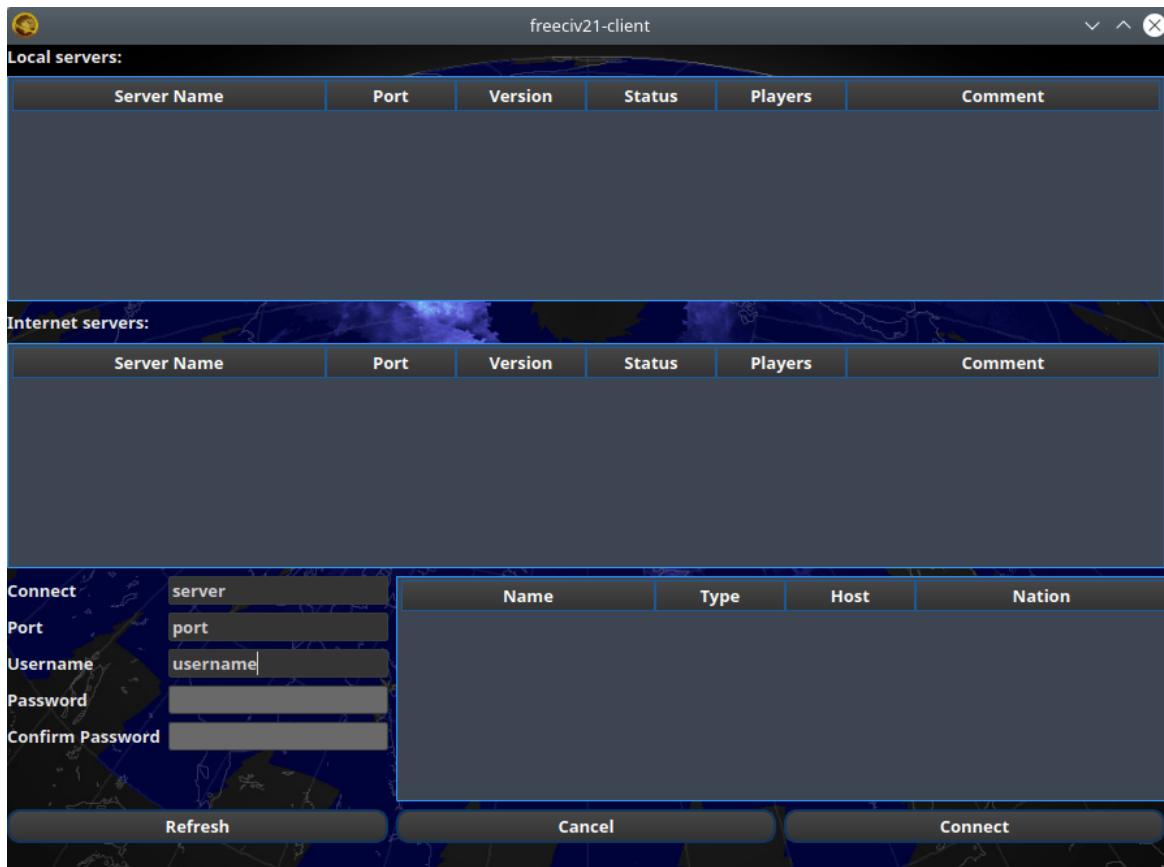


Fig. 3.5: Server Connect Dialog

For the last option, enter in the server name or IP address into the *Connect* text box and the server port in the corresponding *Port* text box. Ensure your username is correct and then click *Connect* to connect to the server. The *Password* box will activate when you have connected to the server. Type in your password and then click *Connect* a second time and you will join the server. Pretty much all Longturn online multiplayer games are connected this way.

Note: If you are hosting your own server with authentication enabled and a player has never connected before they may be prompted to confirm the password a second time in the *Confirm Password* box before being allowed to connect.

Load Saved Game

When you click on the *Load Saved Game* button, a dialog box will appear as in Fig. 3.6 below. Find the saved game you want to load and click (select) it in the table.

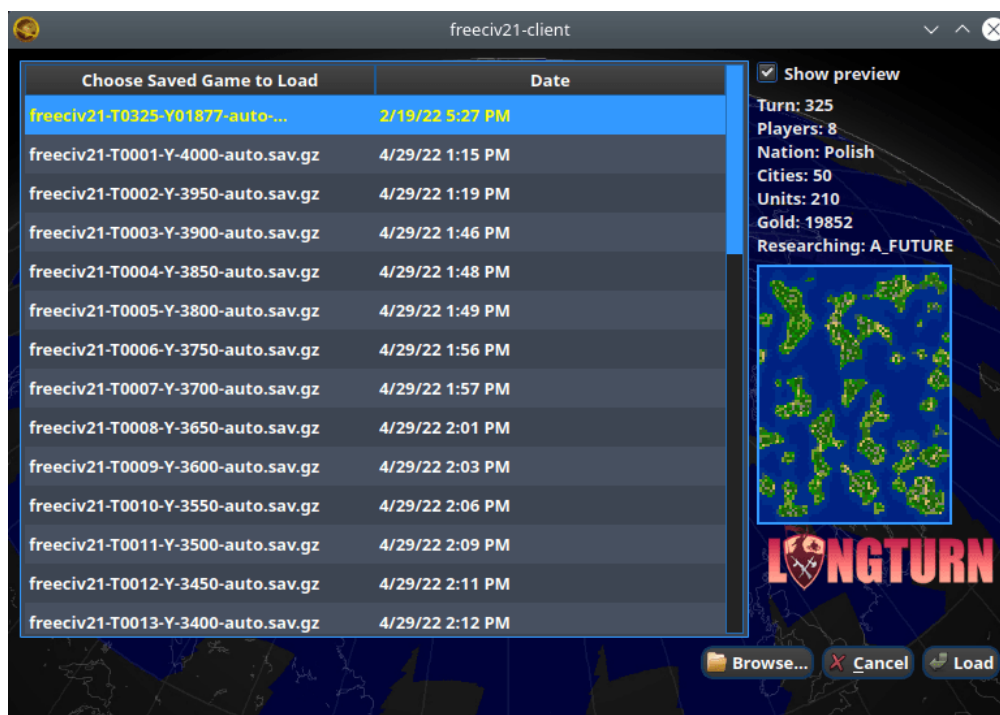


Fig. 3.6: Load Saved Game Dialog

Freeciv21 will show you a sample of the game map and some information about the game. If this is not what you were looking for, select another save from the table. When you have found the save you want to load, click on the *Load* button and you will be placed in the game at the save point.

Alternately, you can click on the *Browse* button to browse your local filesystem to pick a saved game that is not in your user profile.

Start Scenario Game

When you click on the *Start Scenario Game* button, a dialog box will appear as in Fig. 3.7 below. You can click on a scenario to select it and see information about the selected scenario on the panel to the right side. When you have found the scenario you want to run, click on the *Load Scenario* button. This will bring up the new game dialog as shown in Fig. 3.2 above. When ready, click *Start* to begin the scenario game.

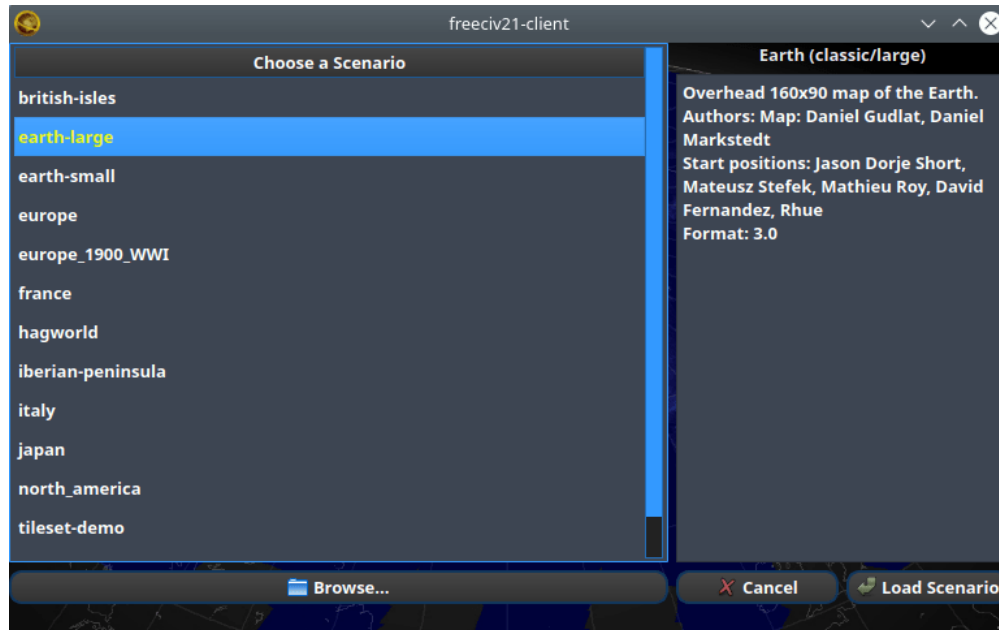


Fig. 3.7: Scenarios Dialog

Tip: If you are new to Freeciv21, the *Tutorial* scenario will walk you through about 100 turns of tips on how to get started playing a single player game.

Clicking on the *Browse* button will bring a filesystem browser that you can use to pick a scenario file in an alternate location. Clicking on *Cancel* will bring you back to the *Starting the Game* start screen.

Options

From the Start Screen, as shown in Fig. 3.1, when you click on the *Options* button, a dialog box will appear as in Fig. 3.8 below. This dialog allows you to set a myriad of options that affect the look and feel of the user interface when you are playing a game.

The interface options dialog box can also be opened while in a game via the *Game* → *Interface Options* menu as well as from the *Start New Game* dialog by clicking on the *Interface Options* button.

The Interface Options dialog is broken down into the following tabs:

- *Network*: On this tab you can save your preferred username, server and port information to be used during *Connect to Network Game*.
- *Sound*: On this tab you can set everything related to in game sound and music.
- *Interface*: On this tab you can set anything related to how you interact with the interface of the client while playing a game.

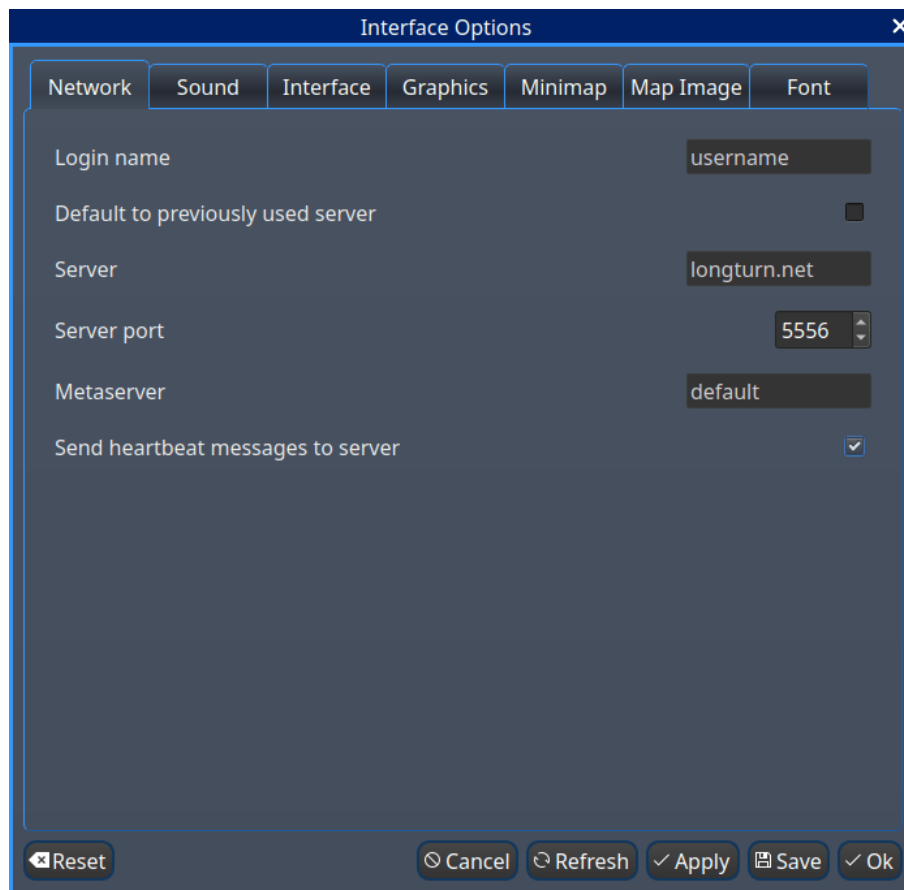


Fig. 3.8: Interface Options dialog

- *Graphics*: On this tab you can change the look and feel with a different theme. NightStalker is the default theme out of the box. You can also set the default tileset to use under different map styles as well as various things you may or may not want the client to paint (draw) on the screen.
- *Overview*: On this tab turn various features of the minimap on and off.
- *Map Image*: Freeciv21 can save summary images of the map every turn. This tab allows you to configure how you want to do that. Refer to [Game Menu](#) for more information.
- *Font*: There is a collection of font styles used by the client. This tab allows you to tailor them to your own favorites. We ship with the Libertinus font set and it is the default.

The buttons along the bottom of the dialog box are:

- *Reset*: Clicking this button will reset all the options to the out of box defaults.
- *Cancel*: Clicking this button will either return you to the [Starting the Game](#) start screen or close the dialog and drop you back to the [Map View](#) if requested from the *Game* menu.
- *Refresh*: If you used the modpack installer to add a new tileset, soundset, or musicset with the local options dialog open, you can use the *Refresh* button to reload the available choices for some of the drop down selection boxes. For more information on the modpack installer, refer to [Using the Modpack Installer Utility](#).
- *Apply*: Apply the settings as set to the client for immediate effect. This button will not do much if local options was called from the [Start New Game](#) dialog box.
- *Save*: Save the current settings.
- *OK*: Apply the settings and close the dialog box. This button does not do a save operation by default.

3.1.2 Main Game Interface

Regardless the method you used to get into a game, eventually you will make it to the main game interface as shown in [Fig. 3.9](#) below. This is known as the [Map View](#).

The game interface is broken down into a collection of sections. From the top, the sections are:

- **Menu Bar**: Like many programs, Freeciv21 has a menu bar at the top. For details see [Menu Bar](#).
- **Top Function Bar**: This top bar is a big aspect of playing any Freeciv21 game. More details are available in the [Top Function Bar](#) section.
- **The Main Map**: This is the main canvas of the game. This is where your empire will either thrive or be destroyed by your enemies. Refer to [Map View](#) for more information.
- **Unit Controls**: [Fig. 3.9](#) shows a Battleship selected and the available unit actions in a bar at the bottom middle of the interface. This is the [Unit Controls](#) bar.
- **Minimap**: In the lower right corner of the main map interface is a small version of the map known as the [Minimap](#).



Fig. 3.9: Freeciv21 Game Overview

3.1.3 Menu Bar

The Menu Bar consists of the following items:

- *Game Menu*
- *View Menu*
- *Select Menu*
- *Unit Menu*
- *Combat Menu*
- *Work Menu*
- *Multiplayer Menu*
- *Civilization Menu*
- *Help Menu*

Game Menu

The *Game* menu is used to conduct global level options on the client or the game being played. It has the following options:

Save Game

Saves the game as it is **right now** to the default save location.

Save Game As ...

Saves the game as it is **right now** to a location of your choosing.

Save Map to Image

Save a .png image file of the map to the user's pictures directory.

Interface Options

Opens the *Interface Options* dialog box as described in the [Options](#) section above.

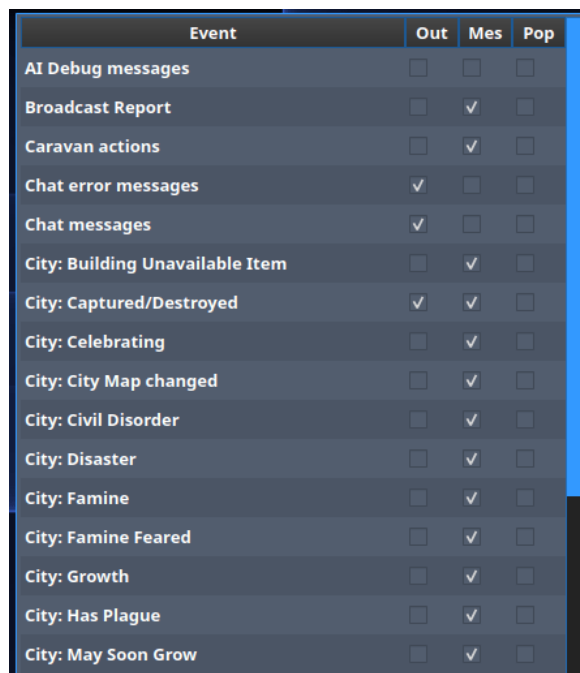
Game Options

Opens the *Game Options* dialog as described in the [More Game Options](#) section above.

Message Options

Opens the *Message Options* dialog as shown in [Fig. 3.10](#) below. The screenshot only shows a few rows of available options, many more will be found in the client. Any item with a check mark in the *Out* column will be shown in the *Server Chat/Command Line* widget. Any item with a check mark in the *Mes* column will be shown in the *Messages* widget (see [Messages](#)). Lastly, any item with a check mark in the *Pop* column will be shown in a pop up message box window.

You can go to the [Game Message Options](#) page for a complete list of all available options.



Event	Out	Mes	Pop
AI Debug messages	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Broadcast Report	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Caravan actions	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Chat error messages	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Chat messages	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
City: Building Unavailable Item	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Captured/Destroyed	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Celebrating	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: City Map changed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Civil Disorder	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Disaster	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Famine	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Famine Feared	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Growth	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Has Plague	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: May Soon Grow	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Fig. 3.10: Message Options Dialog

Shortcut Options

Opens the *Shortcuts* dialog as shown in [Fig. 3.11](#) below. The dialog is used to set the preferred keyboard shortcuts to be used in the game. The screenshot only shows a few rows of available options, many more will be found in the client. To override the default, click your mouse cursor on the box on the right and then type the shortcut you want to set the setting to. For example: You want to change the *Scroll Map* option to the *PgUp* key, you would click in the box and then press the *PgUp* key on your keyboard. To reset to defaults, you can click on the *Reset* button. Click *Save* and then *Close* when finished.

You can go to the [Game Shortcut Options](#) page for a complete list of all available options.

Load Another Tileset

Opens the *Available Tilesets* dialog as shown in [Fig. 3.12](#) below. You can select any tileset installed by clicking on the name. You may get an error message if the tileset is not compatible with the current ruleset (for example: if it lacks a unit).

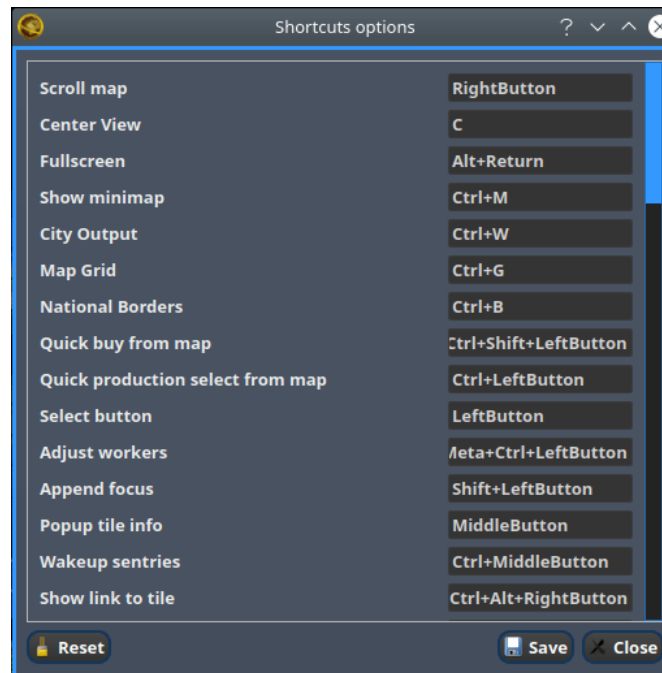


Fig. 3.11: Shortcut Options Dialog

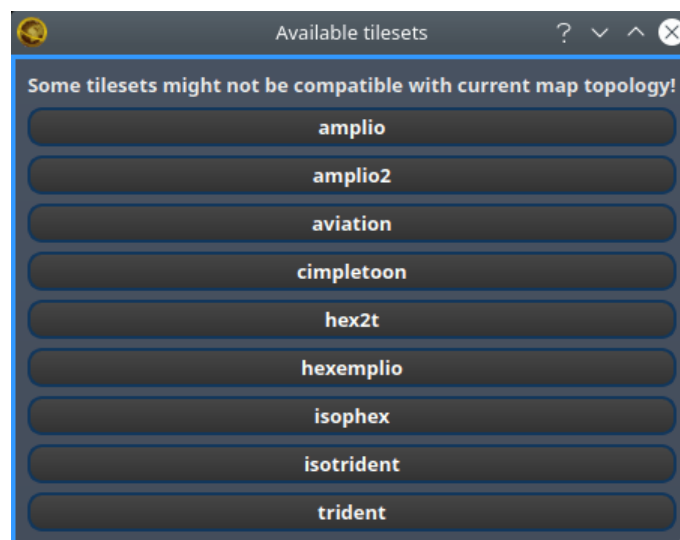


Fig. 3.12: Available Tilesets Dialog

Add Modpacks

Launches the modpack installer. This utility allows the Longturn community to create third-party content and offer it for enhanced gameplay. For more information refer to *Using the Modpack Installer Utility*.

Tileset Options

For tilesets supporting it, opens a dialog that lets you change the appearance of the map.

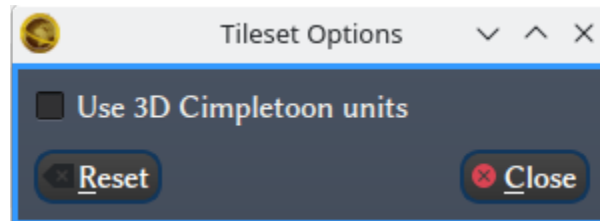


Fig. 3.13: Tileset Options Dialog

Tileset Debugger

Opens the *Tileset Debugger* dialog. This option is documented in *Tileset Debugger*

Save Options Now

Save the settings set in *Options* immediately.

Save Options on Exit

Saves the settings set in *Options* when the client is exited.

Leave Game

Allows you to leave the game and return to the start screen.

Quit

Quits the client after a confirmation dialog box.

View Menu

The *View* enables a user to manipulate what is shown on the *Map View* as well as varying degrees of zooming in and out.

The *View* menu has the following options:

Center View

With a unit selected, this menu option will place the unit in the center of the screen.

Fullscreen

Sets Freeciv21 to use the whole screen, removing the title bar and operating system Task bar.

Minimap

Shows or hides the *Minimap* in the lower right corner.

Show New Turn Information

Enables or Disables populating new turn information in a widget on the *Map View*.

Show Detailed Combat Information

Enables or Disables populating the *Battle Log* widget. When enabled you will see a widget appear on the screen (typically in the upper left corner) after combat occurs in your nation's vision. Your nation's vision is all map tiles that are visible to your nation, either natively or via shared vision treaty with an ally or team mate. The figure below gives an example of 3 combat events.

You can move the widget by click+dragging with your mouse on the plus symbol in the upper left corner. You can also scale the widget larger or smaller with the plus and minus icon buttons near the upper right corner. You



Fig. 3.14: Battle Log

can close the widget by clicking on the x symbol in the upper right corner. Lastly, if you click on the winning unit icon in a row, the client will move the map to where the combat occurred.

If you do not do anything with the *Battle Log* widget after combat occurs, it will fade from the map automatically after 20 seconds.

Lock Interface

Locks the user interface, preventing the move of objects around such as the server log/chat widget.

Zoom In

Each selection of this menu option (or corresponding keyboard shortcut or user interface button next to the *Minimap*) will zoom in on the *Map View* one level.

Zoom Default

Resets the zoom level to the default position at Zoom Level 0.

Zoom Out

Each selection of this menu option (or corresponding keyboard shortcut or user interface button next to the *Minimap*) will zoom out on the *Map View* one level.

Scale Fonts

Enables fonts to scale along with the zoom level.

City Bar Style

This menu has a sub-menu of three options: *Simple*, *Traditional*, and *Polished* as shown in Fig. 3.15, Fig. 3.16, and, Fig. 3.17, respectively.

City Outlines

Enables or Disables the display on the *Map View* of the city's vision radius or outline.

City Output

Enables or Disables the display on the *Map View* the city's food, production, and trade as shown in the *City Dialog*.

Map Grid

Enables or Disables the display on the *Map View* the tile grid. This can be useful to help differentiate individual tiles from others.



Fig. 3.15: City Bar Style - Simple



Fig. 3.16: City Bar Style - Traditional



Fig. 3.17: City Bar Style - Polished

National Borders

Enables or Disables the display on the *Map View* the national borders of your neighbors. Each nation is given a color at game start (as seen on the *Nations and Diplomacy View*).

Native Tiles

Enables or Disables the display on the *Map View* tiles that are native to the unit selected. Non-native tiles are marked with a red hash. Non-Native means that the unit cannot move there.

City Names

Enables or Disables the display on the *Map View* the names of cities in the city bar.

City Growth

Enables or Disables the display on the *Map View* the growth of cities in the city bar.

City Production Levels

Enables or Disables the display on the *Map View* the production of cities in the city bar.

City Buy Cost

Enables or Disables the display on the *Map View* the cost to buy the currently constructed item in the city bar.

City Traderoutes

Enables or Disables the display on the *Map View* trade routes between cities.

Select Menu

The *Select* is used to select units on the game map in varying degrees. It has the following options:

Single Unit (Unselect Others)

Assuming you have selected multiple units (such as the next menu for *All on Tile*) and you want to quickly undo that. This menu supports that function.

All on Tile

Quickly select all of the units on the same tile. This assumes that multiple units of different type are stacked on the same tile.

Same Type on Tile

Quickly select all of the units of the same type on the tile. This assumes that multiple units of different types are stacked on the same tile.

Same Type on Continent

Works the exact same way as the menu option above (*Same Type on Tile*) but expands the selection to the same island or continent.

Same Type Everywhere

Even larger window of unit selection to pick all of the same type, but in all locations across the *Map View*. This is useful to help you find units placed in allied cities or to conduct a mass airlift.

Wait

If you are not ready to move the currently selected unit, you can tell it to **wait** with this menu option. The rotation of unit selection will bypass this unit and will come back until you either move the unit or tell it you are done.

Done

If you have moved the unit, but still have move points left or are simply done working with the unit for you can tell the client you are finished (done) with it with this menu item.

Advanced Unit Selection

Opens the *Advanced Unit Selection* dialog box as shown in [Fig. 3.18](#) below. This dialog box gives you options to select a group of units using a more nuanced grouping method than the other *Select* menu options.

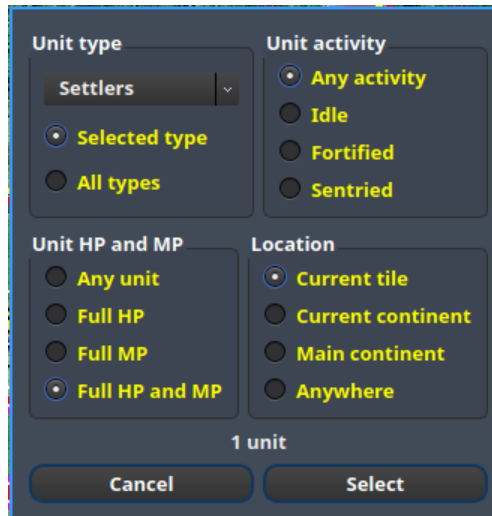


Fig. 3.18: Advanced Unit Selection Dialog

Unit Menu

The *Unit* menu is used to give units orders. It has the following options:

Go to Tile

With a unit selected, give the unit orders to go to a selected tile on the map.

Go to and ...

Similar to *Go to Tile* above, however when the unit reaches the given tile you can give the unit a specific command to do as selected from the sub-menu. This is useful, for example, for Settler units to have them go to a spot and build a city as soon as possible.

Go to Nearest City

Instruct the unit to go to the nearest city. Nearest in this context is the one that can be reached in the fewest move points (MPs).

Go to / Airlift to City...

If Airlifting is enabled in the game (and assuming you have any required city improvement(s) that are required for Airlifting), you use this menu to tell the unit to transport to a city with the Airlift capability. A dialog box will pop up asking what city you want to Airlift to.

Tip: Depending on the game rules, a player could use the *Select* menu to select many units of a similar type and then use this menu item to Airlift a great number of units all in one move very quickly.

Autoexplore

Ask the unit to automatically open up the unknown (the area of the map that is black and has not been visited by any of your units).

Patrol

Instruct a unit to make a collection of moves in a pattern as part of a patrol route.

Sentry

Ask a unit to **Sentry**. Sentry is not the same as **Fortify** as found in the *Combat* menu. A sentried unit is on lookout and will give notice if another unit from an opponent comes into its field of vision.

Note: A sentried unit does not gain a fortification bonus when outside of a city. However, sentried units inside of cities gain a default fortification bonus while in the city.

Unsentry All On Tile

Instruct a stacked set of units on a single tile to stop that activity and ask for new orders.

Load

Load a unit into a transporter, such as a Caravel, Galleon, or Transport ship.

Unload

Unload a unit from a transporter.

Unload All From Transporter

If you have many units inside of a transporter and you want all of them to disembark at the same time, then you can use this menu to make that nice and easy.

Set Home City

Transfers ownership and management (e.g. support) of a unit to the city that it is currently present in. This menu allows you to shift support of units around to help with the cost of supporting units. Refer to [Economics View](#) for more information on unit support costs.

Upgrade Unit

Upgrade a unit from one level to another. For example, Phalanx units are often upgradeable to an improved Pikemen with the discovery of **Feudalism**. The upgrade will cost gold and the client will tell you what that cost is before you agree to the spend.

Convert Unit

Similar to *Upgrade*. The convert option allows you to change a unit from one type to another. This is ruleset dependent and is not available in all rulesets.

Disband Unit

Use this menu option to eliminate (kill, destroy, fire) a unit. If performed inside of a city, then 50% of the shields used in the production of the unit is given to the city and helps build whatever is currently under construction. If a unit is disbanded outside of a city, there is no benefit other than the elimination of shield or gold upkeep depending on your form of government.

Rename Unit

Give the unit a unit name. Similar to many Naval vessels in real life, you can name your units with a special name.

Combat Menu

The *Combat* menu is used to give combat units orders. It has the following options:

Fortify Unit

Instruct the unit to **Fortify** on the tile. An icon will show on the unit signifying fortification. By fortifying, the unit is given a defensive bonus depending on the terrain it is on. See in game help for more specifics of what defense bonuses are given by terrain type. When a unit is fortifying, it is not in **Sentry** mode and will not inform you of enemy unit movement inside of its vision radius.

Build Fortress / Buoy

Some units have the ability to build forts, fortresses, and buoys. They are typically Workers or Engineers. In some rulesets, Transports can build Buoys. Forts, Pre-Forts, and Fortresses give a unit increased defensive bonuses in addition to what is provided by the base terrain. See in game help for the specifics. Buoys are used to act as sentries in the oceans around your cities and can give you opponent unit movement information.

Build Airbase

Instructs a unit to build an Airbase. This is often a requirement for Airlifting in many rulesets. They also often give the ability to heal an aircraft type unit faster while on the tile.

Build Base

This generic menu will include a sub-menu of all of the base type tile improvements that can be built as defined by the current ruleset.

Pillage

Tells a unit to remove (pillage) tile infrastructure improvements such as roads, railroad, and bases.

Do ...

A dialog box will pop up here and give you all of the actions that the unit selected can perform.

Work Menu

The *Work* menu is used to give units infrastructure work orders such as building roads, irrigation, or mines. Tile infrastructure improvements are mostly done with Workers and Engineers, however some rulesets allow other units to perform this type of work. See in game help on units for details. It has the following options:

Build City

Certain units such as Settlers can create cities. If the unit has sufficient move points available, then giving this command will build a new city where the unit is currently placed on the [Map View](#). The unit will be consumed by the action.

Auto Worker

Tell a Worker to use an in game algorithm to improve tiles. The game engine will give the Worker instructions so you do not have to.

Build Road / Railroad / Maglev

Tell a Worker to build a road. If sufficient technological knowledge is available, then a railroad and eventually a maglev may be able to be constructed at a later time during the game. The menu will change with the best available option depending on what has been done to improve the tile in the past.

Build Path

Provides a sub-menu of all of the pathing options available for the tile. This is mostly road, railroad, and maglev. Other rulesets may have different path types. See in game help for more details.

Build Irrigation / Farmland

Tell a Worker to irrigate the tile in order to improve the food output from the tile. If sufficient technological knowledge is available, then farmland may be added to the tile at a later time during the game. The menu will change with the best available option depending on what has been done to improve the tile in the past.

Cultivate to Plains

Cultivation is a multi-step process where a tile is converted from one type to another. Such as converting a swamp to plains. Not all tile terrain types can be cultivated to other types. See in game help for details.

Build Mine

Tells a Worker to build a mine on a tile to improve the shield output.

Plant Forest / Swamp

If the unit is on a grassland tile, then you can tell the Worker to plant a forest on the tile. If the unit is on a forest tile, then you can tell the Worker to convert the forest to swamp.

Connect with Road

Tell a Worker to build a road many times along a given path.

Connect with Railroad / Maglev

Tell a Worker to build a railroad or maglev many times along a given path.

Connect with Irrigation

Tell a Worker to connect many tiles together with irrigation. This is often done to get irrigation from a source of fresh water over to a city.

Transform to Hills / Swamp / Ocean

Tell an Engineer to conduct a major terraforming operation on the tile. Mountains can be cut down to hills, forests, and grassland can be converted to swamp and then the swamp can be converted to ocean. In some circumstances, an ocean tile can be converted to swamp and then the swamp can be converted to grassland. In most rulesets, only the Engineer unit can do these major operations.

Clean Pollution

Tell a Worker to clean pollution from the tile. Pollution on a tile will eliminate or severely cripple the output of a tile and contributes to global warming.

Clean Nuclear Fallout

If a Nuclear unit has been detonated nearby (e.g. attacked a city), then there will be nuclear fallout all over the place. Similar to pollution, nuclear fallout severely cripples the output of a tile and contributes to nuclear winter.

Help Build Wonder

Certain units, such as Caravan can be used to move production from one city to another and help build small and great wonders. This menu aids that function. Alternately you can simply “walk” the Caravan into a city and a pop up dialog will ask what you want to do.

Establish Trade Route

Certain units, such as Caravan can be used to establish a trade route between two cities. This menu aids that function.

As you can see, there are a number of ways that a tile can be altered with infrastructure improvements. Be sure to have a close look at the in game help on Terrain for more information.

Multiplayer Menu

The *Multiplayer* menu has a collection of functions to aid certain multiplayer games. Many of the options are specifically tailored to the MP2c and WarCiv rulesets. It has the following options:

Delayed Go To

Give a unit orders to move at a specific time in the turn. This assumes that the turn is time based.

Delayed Orders Execute

Execute an action by a unit at a specific time in the turn. This assumes that the turn is time based.

Clear Orders

Clear any delayed orders from the above two menu items.

Add All Cities to Trade Planning

Add all current cities into an advanced trade planning array. This is used by the WarCiv ruleset.

Calculate Trade Planning

Run a trade effectiveness algorithm across all of the cities in the trade plan to determine the best routes. This is used by the WarCiv ruleset.

Add / Remove City

Add or remove a city from the trade planning array.

Clear Trade Planning

Clear all trade planning to start over.

Automatic Caravan

Any Caravan units built by a city will follow the trade planning output for that city. This is used by the WarCiv ruleset.

Set / Unset Rally Point

Set or remove a rally point to easily send units, once produced, to a specific spot on the game map.

Quick Airlift

Depending on what is selected in the menu below, you can quickly airlift a unit to a destination city.

Unit Type for Quickairlifting

Select the type of unit that will be quickly airlifted by the menu above.

Default Action vs Unit

A sub-menu will show some optional actions that a unit should do by default against another unit. The default is **Ask**.

Default Action vs City

A sub-menu will show some optional actions that a unit should do by default against a city. The default is **Ask**.

Civilization Menu

The *Civilization* menu is used to gain access to many functions of your empire. You can load up pages for units, cities, nations, etc; change the form of government and see how you are doing compared to your opponents with the demographics report. It has the following options:

National Budget...

Selecting this menu item will bring up a dialog box allowing you to set the rate in percentage points for gold (taxes), science (bulbs), and luxury (goods). This is the same as clicking on the *National Budget View* button on the *Top Function Bar*. Fig. 3.19 dialog below, shows a sample screenshot. In this example, the player's nation is in Democracy, has set gold to 30%, science to 40%, and luxury to 30%.

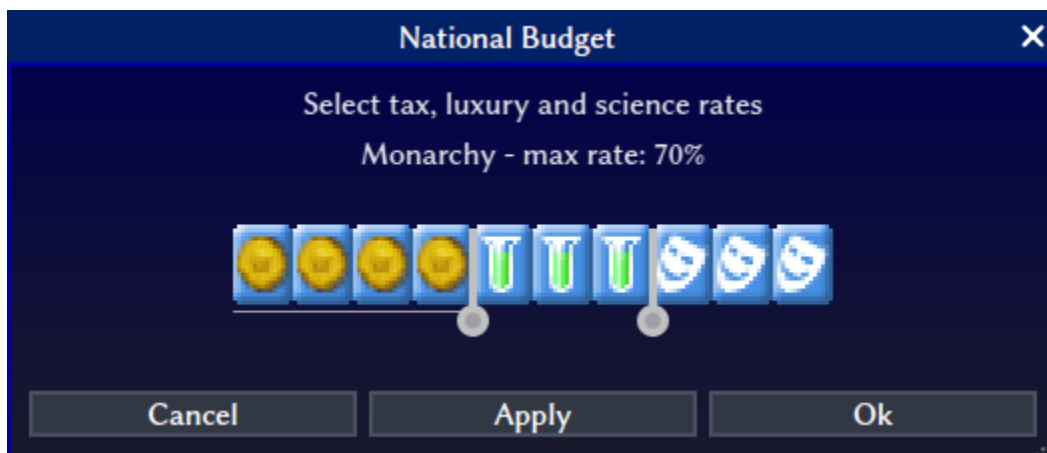


Fig. 3.19: National Budget Dialog

Note: In Freeciv21 the National Budget is simplified into three segments: gold collection, scientific output, and luxury goods.

Government

Depending on your technological progress through the game, you may be able to select a different form of government. The change is achieved from this menu item. This is the same as clicking on the *National Status View* option on the *Top Function Bar*.

Map View

Clicking this menu item will show you the main game map. This is the same as clicking on the *Map View* option on the *Top Function Bar*.

Units View

Clicking this menu item will show you the units report widget. This is the same as clicking on the *Units View* option on the *Top Function Bar*.

Nations View

Clicking this menu item will show you the nations and diplomacy view. This is the same as clicking on the *Nations and Diplomacy View* option on the *Top Function Bar*.

Cities View

Clicking this menu item will show you the cities view. This is the same as clicking on the *Cities View* option on the *Top Function Bar*.

Economy View

Clicking this menu item will show you the economics view. This is the same as clicking on the *Economics View* option on the *Top Function Bar*.

Research View

Clicking this menu item will show you the research tree view. This is the same as clicking on the *Research View* option on the *Top Function Bar*.

Wonders of the World Report

Clicking this menu item will show you a traveler's report widget on the *Map View*. The widget will give information on any cities that have constructed any of the great wonders.

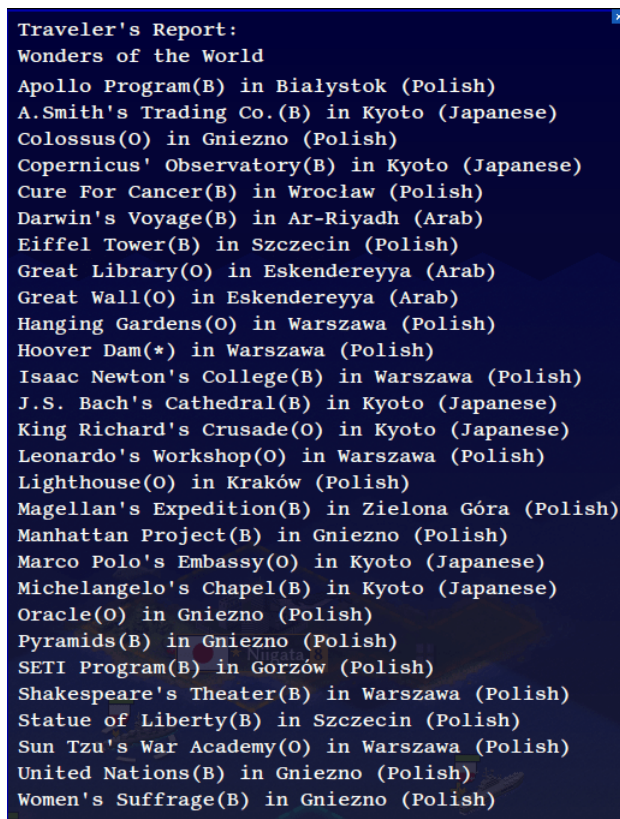
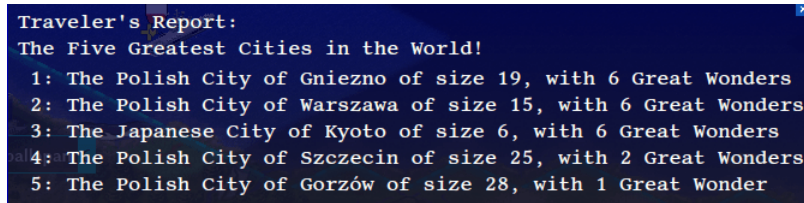


Fig. 3.20: Wonders of the World

Top Five Cities Report

Clicking this menu item will show you a traveler's report widget on the *Map View*. The widget will give information on the top five largest cities.



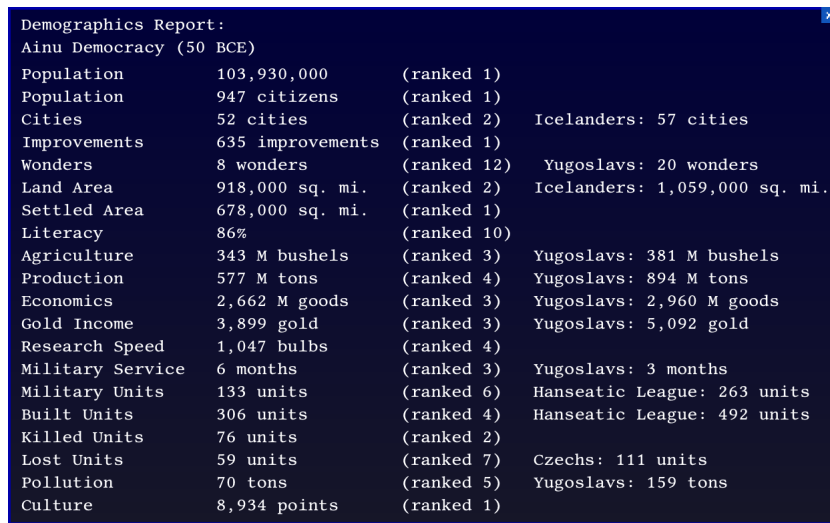
```

Traveler's Report:
The Five Greatest Cities in the World!
1: The Polish City of Gniezno of size 19, with 6 Great Wonders
2: The Polish City of Warszawa of size 15, with 6 Great Wonders
3: The Japanese City of Kyoto of size 6, with 6 Great Wonders
4: The Polish City of Szczecin of size 25, with 2 Great Wonders
5: The Polish City of Gorzów of size 28, with 1 Great Wonder
  
```

Fig. 3.21: Top Five Cities

Demographics Report

Clicking this menu item will show you a demographics report widget on the *Map View*. The widget will give information about how your nation stacks up against your opponents. If you have an embassy with your opponents in the game, the demographics report will tell you who is #1, if you are not #1.



```

Demographics Report:
Ainu Democracy (50 BCE)
Population      103,930,000      (ranked 1)
Population      947 citizens      (ranked 1)
Cities          52 cities        (ranked 2)      Icelanders: 57 cities
Improvements    635 improvements  (ranked 1)
Wonders         8 wonders       (ranked 12)     Yugoslavs: 20 wonders
Land Area       918,000 sq. mi.   (ranked 2)     Icelanders: 1,059,000 sq. mi.
Settled Area    678,000 sq. mi.   (ranked 1)
Literacy        86%              (ranked 10)
Agriculture     343 M bushels     (ranked 3)     Yugoslavs: 381 M bushels
Production      577 M tons        (ranked 4)     Yugoslavs: 894 M tons
Economics       2,662 M goods     (ranked 3)     Yugoslavs: 2,960 M goods
Gold Income     3,899 gold        (ranked 3)     Yugoslavs: 5,092 gold
Research Speed  1,047 bulbs       (ranked 4)
Military Service 6 months         (ranked 3)     Yugoslavs: 3 months
Military Units   133 units         (ranked 6)     Hanseatic League: 263 units
Built Units      306 units         (ranked 4)     Hanseatic League: 492 units
Killed Units     76 units          (ranked 2)
Lost Units       59 units          (ranked 7)     Czechs: 111 units
Pollution       70 tons           (ranked 5)     Yugoslavs: 159 tons
Culture         8,934 points      (ranked 1)
  
```

Fig. 3.22: Demographics

Spaceship View

Clicking this menu item will show you the spaceship view. The space race is a ruleset defined option and is not enabled in all rulesets. Your nation must also be very technologically advanced to build the components needs for a spaceship. See in game help for more details.

The client will automatically place the components for you as you construct them.

Achievements Report

Clicking this menu item will show you an achievements report widget on the main map. Achievements are a ruleset defined option and not enabled by default in many rulesets.

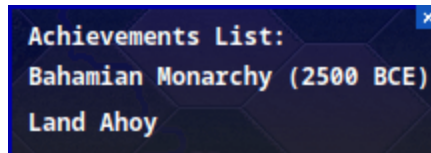


Fig. 3.23: Achievements

Help Menu

The *Help* menu gives you access to the in-game help. It has the following chapters:

- Overview
- Strategy and Tactics
- Terrain
- Economy
- Cities
- City Improvements
- Wonders of the World
- Units
- Combat
- Zones of Control
- Government
- Effects
- Diplomacy
- Technology
- Space Race
- About Current Tileset
- About Current Ruleset
- About Nations
- Connecting
- Controls
- Citizen Governor
- Chatline
- Worklist Editor
- Languages
- Copying
- About Freeciv21

Each of these options is simply a quick link to the same named section in the game help menu.

3.1.4 Top Function Bar

The *Top Function Bar* is used to get to varying views (pages) in the game without having to use the main menu, especially the *Civilization* menu. The *Top Function Bar* is broken up into 9 sections, from left to right.

- *Map View*
- *Units View*
- *Cities View*
- *Nations and Diplomacy View*
- *Research View*
- *Economics View*
- *National Budget View*
- *National Status View*
- *Messages*

Map View

This is your primary playing surface. This is the map where you build your civilization. The button for this is shown in Fig. 3.24 below. Fig. 3.9, has a good example of the *Map View*.

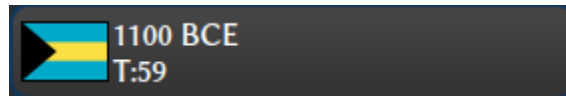


Fig. 3.24: Map View Button

If you hover your mouse over the *Map View* button, a pop up widget will appear and give you information about your nation. The pop up widget shows: Nationality, Total Population, Year (Turn Number), Gold (Surplus/Deficit), and National Budget.

To move around the map canvas, you can right-click in the main map area and the canvas will move. The further from the center of the screen, the faster the map canvas will move per mouse click. You can also use two-finger gestures on your mouse/trackpad to swipe up, down, left, and right.

One other feature of the *Map View* is the ability to middle-click on a unit (via the *Popup Tile Info Shortcut*). After a middle-click a popup widget will appear giving you some information about the tile. Fig. 3.25, gives an example of a Howitzer on Plains. The popup information widget gives a great deal of information about the coordinates of the tile, terrain type, and infrastructure improvements made to the tile. If a unit is on the tile, as in our example, you are also given details about the unit.

Tip: The client will give you some basic combat chances if you select one of your units and then use the middle-click popup tile info shortcut on the tile of an enemy unit.

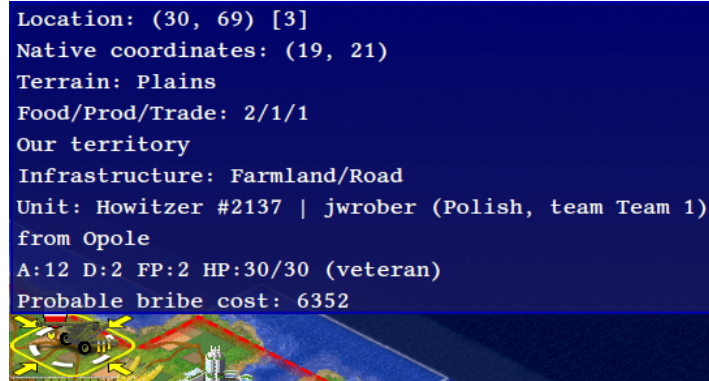


Fig. 3.25: Map View - Unit Information

Units View

The *Units View* is a separate page in a table format. When you click the button for it as shown in Fig. 3.26, the client will switch to a listing of your units.



Fig. 3.26: Units View Button

The *Units View* has two tables. For regular games without the `unitwaittime` server setting set, you get something similar to Fig. 3.27 below. If you are playing a game with `unitwaittime` set, then you will see a second table below the table shown in Fig. 3.27, such as Fig. 3.28, that displays the amount of time until the unit can move.

The *Find Nearest* button can be used to help you find a specific unit type. Select the row for the unit type you wish to find on the *Map View* and then click the button. The *Units View* page will close and the closest unit of that type will be selected and centered on the map. To quickly disband every one of a specific unit type, select the row for the unit type and then click *Disband All*. This feature is similar to the same button in the *Economics View*. Lastly, if you have sufficient funds in your national treasury, you can upgrade all units of a type that are inside your cities by selecting the row for the unit type you wish to upgrade and click the *Upgrade* button.

Nations and Diplomacy View

The *Nations and Diplomacy View* is actually two views accessed from the same place on the *Top Function Bar*. When you click the button for it as shown in Fig. 3.29, the client will switch to a list of nations that you are playing against in a table format.

Fig. 3.30 gives a sample of the *Nations View* in the client with all available columns displayed. If you right-click on the table heading, you will be given a list of column names that you can enable or disable. If you change anything, then be sure to save the settings from the *Game* menu. If any players have been killed in the game, the table will show *R.I.P.* next to the name of the player that has been destroyed.

If you have an embassy with a nation you will be able to see much more in the table than if you do not have an embassy. If you select the row of a nation you have an embassy with, you will be given some interesting intelligence at the bottom of the page. Fig. 3.31 gives an example.

On the left you will see the name of the Nation, The name (username) of the Ruler, the current form of Government, the Capital city, how much gold they have in their national treasury, the national budget ratios, research target, and culture score.

Units:

Find Nearest Disband All Upgrade

Unit Type	★ Upgradable	⚡ In Progress	✂ Active	🛡 Shield Upkeep	🍖 Food Upkeep	👑 Gold Upkeep
Archers	★	0	3	-	-	-
Explorer	-	0	1	-	-	-
Flagship Frigate	-	1	0	-	-	-
Longboat	★	0	1	-	-	-
Musketeers	-	9	32	-	-	-
Phalanx	★	0	1	-	-	-
Pikemen	★	0	5	-	-	-
Warriors	★	0	3	-	-	-
Workers	-	0	31	-	-	-
Totals:	5	10	77	0	0	0

Fig. 3.27: Units View

Units Waiting:

Unit Type	Near Location	Time left
Longboat	near Treasure Cay	30 minutes 50 seconds
Musketeers	near Tangier	30 minutes 47 seconds

Fig. 3.28: Units View with unitwaittime



Fig. 3.29: Nations and Diplomacy View Button

Name	~	Username	Flag	Nation	Border	Score	Team	AI	Attitude	Embassy	Dipl.State	Vision	State	Government	Culture	Gold	Tax	Science	Luxury
Sithu Kyawhtin		Unassigned		Shan		?	Team 6	✓	Uncooperative	No Contact	War			?		?	?	?	??
Roland Symonette		jwrober		Bahamian		316	Team 1	-	-	-	-		moving		Monarchy	731	152	70	30
Mohammed V		Unassigned		Moroccan		?	Team 4	✓	Neutral	No Contact	War			?		?	?	?	??
Maria Theresia		Unassigned		Austrian		162	Team 3	✓	Uncooperative	Contact	War			?		?	?	?	??
Juan Antonio Lavalleja		Unassigned		Uruguayan		?	Team 2	✓	Uncooperative	No Contact	War			?		?	?	?	??
Benito Mussolini		Unassigned		Italian		?	Team 5	✓	Uncooperative	No Contact	War			?		?	?	?	??

Fig. 3.30: Nations View

Nation: Aztec	War: Poles, Japanese.	Techs unknown by Aztecs: Amphibious Warfare, Combined Arms, Communism, Computers, Environmentalism, Espionage, Fusion Power, Guerilla Warfare, Labor Union, Laser, Mass Production, Miniaturization, Mobile Warfare, Nuclear Fission, Nuclear Power, Plastics, Recycling, Refrigeration, Robotics, Space Flight, Stealth, Superconductors.
Ruler: Speaker Axayacatl	Cease-fire: Koreans.	Techs unknown by you: None
Government: Democracy	Alliance: (e_o)Iroquois, (e_o)Arabs.	Known Wonders: None
Capital: Xochimilco		
Gold: 1221		
Tax: 20%		
Science: 80%		
Luxury: 0%		
Researching: Mobile Warfare (2227/3968)		
Culture: 0		

Fig. 3.31: Nations Intelligence

Note: The Capital City will show as *unknown* if you have not seen the city on the *Map View*. If it is in the unknown or has not been seen by one of your units, then you will not have knowledge of the Capital.

In the center you can see the relationship status of the nation across the game. If you see a half-moon icon next to a nation, then the nation selected has given shared vision to that nation. In Fig. 3.31 above, you can see that the Aztecs have an alliance with the Iroquois and the Arabs. The Aztecs also have shared vision with both of these nations.

On the right, you can see a comparison of technological research between your nation and the nation selected.

The *Nations and Diplomacy View* has a few buttons at the upper left. From left to right, they are: *Meet*, *Cancel Treaty*, *Withdraw Vision*, *Toggle AI Mode*, and *Active Diplomacy*. This is how you access the *Diplomacy* component of the *Nations and Diplomacy View*.

Let us talk about the buttons from right to left as *Meet* takes the longest to describe. If you have any active treaty negotiations occurring, you can click on the *Active Diplomacy* button to switch to that page. The Fig. 3.29 on the top function bar will change to a flag icon with a red dot to give you a visual reminder that there are open meetings to attend to.

Depending on the command line level you have in the game (default is *hack* for single player games), you may be able to change a player from an AI to a human after a game has started to allow a human player to come into the game. This is what the *Toggle AI Mode* button does. If the button is greyed out you cannot change the AI mode in the game.

If you have previously shared vision via a treaty from the *Diplomacy View*, you can revoke it by clicking on the *Withdraw Vision* button. Sharing vision is similar to you allowing another player to see all of your territory.

If you have a relationship with a player other than **War**, you can cancel it with the *Cancel Treaty* button. Relationship pacts can be changed with the *Diplomacy View*. Most rulesets support **Cease Fire**, **Peace**, and **Alliance**.

Note: Some forms of government will not allow you to break a **Peace** or **Alliance** treaty without changing government to Anarchy.

Lastly, clicking *Meet* will bring up a diplomacy screen where you can interact with a player that you have an embassy with. Fig. 3.32, shows a sample screen where the parties are agreeing to a cease fire.

If you wish to give gold to a player, enter in the amount in the *Gold* box. You can also add clauses to the treaty by selecting from the *Add Clause* button. Depending on what is enabled in your game, you can swap sea and world maps, trade cities, give advances, share vision, give an embassy, or agree to peace or an alliance via a pact. If you are happy with the components of the treaty you can click the *Accept Treaty* button. The client will change the thumbs-down icon to the right of your nation to a thumbs-up showing agreement. If you do not want to do anything and definitely do not want to accept the treaty, then you can click on the *Cancel Meeting* button. This will close the *Diplomacy View* and return you to the *Nations View*.

Note: The ability to trade maps, cities, or advances is ruleset dependent and may not be enabled for all games. The



Fig. 3.32: Diplomacy

other clauses such as share vision, give an embassy or change the relationship via a pact are enabled at all times.

Tip: You do not have to use the *Diplomacy View* to get an embassy with a player. You can always build a Diplomat unit and have that unit get an embassy by going to a player's city and "walk" into the city. An action dialog will show and you can establish an embassy without asking via diplomacy. See in game help for more information on using units to conduct many gameplay features besides simply establishing an embassy.

Lastly, you can see in Fig. 3.32 above that there are more than one conversations occurring. Your foreign state department is busy! If you happen to click out of the *Diplomacy View*, for example by clicking on the button for the *Map View*, the button for the *Nations and Diplomacy View* will slowly pulse, giving you a reminder to come back.

To get back to the *Diplomacy View*, you can click on the *Active Diplomacy* button from the *Nations View* described earlier to bring it back up.

Cities View

The *Cities View* is a separate page in a table format. When you click the button for it as shown in Fig. 3.33, the client will switch to a listing of your cities.



Fig. 3.33: Cities View Button

Fig. 3.34 gives an example of the *Cities View* in the client with the default columns displayed. If you right-click on the table heading, you will be given a list of other columns you may want to show. If you change anything, then be sure to

save the settings from the *Game* menu.

Name	Sz	State	Workers H/C/U/A	Grow (Have/Need)	Surplus F/P/T	Economy G/L/S	Production Turns/Buy	Currently Building (Have/Need)
JELEŃIA GÓRA	22	Celebrating	17/3/0/0	--- (148/230)	0/69/57	35/70/0	2/250	Solar Plant (195/320)(worklist)
YOKOHAMA	12	Celebrating	7/5/0/0	--- (72/130)	0/27/28	2/28/0	4/180	Mfg. Plant (130/220)(worklist)
BYDGOSZCZ	26	Celebrating	20/0/0/0	-168 (167/270)	-1/65/105	66/130/81	*	Coinage (65)
ŁÓDŹ	22	Celebrating	14/6/0/0	--- (156/230)	0/61/57	9/56/27	3/286	Solar Plant (177/320)(worklist)
BIAŁYSTOK	24	Celebrating	15/5/0/0	-166 (165/250)	-1/74/49	-4/60/54	*	Coinage (74)
HIROSHIMA	8	Peace	3/4/1/0	6 (73/90)	3/4/24	8/12/0	25/200	Airport (20/120)(worklist)
KOKURA	15	Celebrating	14/1/0/0	--- (98/160)	0/41/56	17/56/0	2/150	Cathedral (5/ 80)(worklist)
KAWASAKI	16	Celebrating	11/5/0/0	4 (159/170)	3/27/45	16/44/0	2/68	Supermarket (46/ 80)(worklist)

Fig. 3.34: Cities

If you double-click on a city row, the game will switch to the *Map View* and open the *City Dialog* for that city. When you close the *City Dialog*, the client will not bring you back to the *Cities View*. If you right-click on a city's row, a pop-up menu will appear. From this menu you can change what the city is producing, set a citizen governor, sell a city improvement, conduct an advanced selection of cities, buy what is currently being produced, and center the city on the map. If you choose to center the city on the map, the client will close the *Cities View* and open the *Map View* and place the city in the center of the screen.

Economics View

The *Economics View* is a separate page set in a table format. When you click the button for it as shown in Fig. 3.35, the client will switch to a listing of your nation's economy. A nation's economy is mostly about city improvement, unit support, and maintenance.

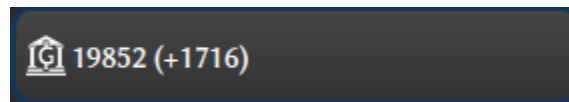


Fig. 3.35: Economics View Button

If you hover your mouse over the button, a pop up widget will appear and give you information about your nation's economy.

Fig. 3.36 below shows a sample *Economics View*. Notice that you can see city improvements and units in a table format giving you the number produced, how much gold in upkeep per turn each consumes, total gold upkeep per turn for all of them, and if any are redundant. A redundant improvement is one that has been overcome by events; typically by a new technological advancement. You want to sell redundant items as they are costing you gold and giving nothing back in return. The *Economics View* will not tell you what city the item is redundant in, you will have to go find that yourself. This is a good use case for the advanced select option in the *Cities View*.

The *Economics View* has a few buttons in the upper left: *Disband*, *Sell All*, and *Sell Redundant*. The *Disband* button will disband all the units of a type that has been selected in the view. The *Sell All* button does the same for a city improvement that has been selected in the view. Lastly, the *Sell Redundant* button will only sell redundant city improvements in those cities for the city improvement that has been selected in the view. The *Messages* view will tell you what was sold where.

Note: You can only sell one city improvement at a time per turn, so you might not be able to do all the things you want every turn.

Income: 211 Total Costs: 89					
<div>Disband</div> <div>Sell All</div> <div>Sell Redundant</div>					
Type	Name	Redundant	Count	Cost	Total Upkeep
	Aqueduct	0	6	1	6
	Aqueduct, Lake	0	2	0	0
	Aqueduct, River	0	5	0	0
	Barracks II	0	9	1	9
	Amphitheater	0	3	2	6
	Courthouse	0	21	1	21
	Granary	0	25	1	25
	Marketplace	0	6	0	0
	Temple	0	22	1	22
	Workers	0	25	1	0
	Warriors	0	6	1	0
	Phalanx	0	2	1	0
	Pikemen	0	9	1	0
	Musketeers	0	18	1	0
	Longboat	0	1	1	0

Fig. 3.36: Economics View

Research View

The *Research View* is a separate page showing the technology research tree. This is the page where you instruct your scientists which technologies to research. When you click the button for it as shown in Fig. 3.37, the client will switch to your research tree.

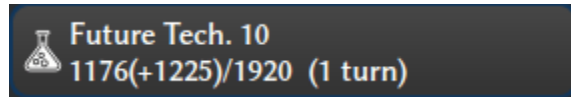


Fig. 3.37: Research View Button

Fig. 3.38 below shows a sample of a *Research Tree*. In this picture the player is researching Sanitation.

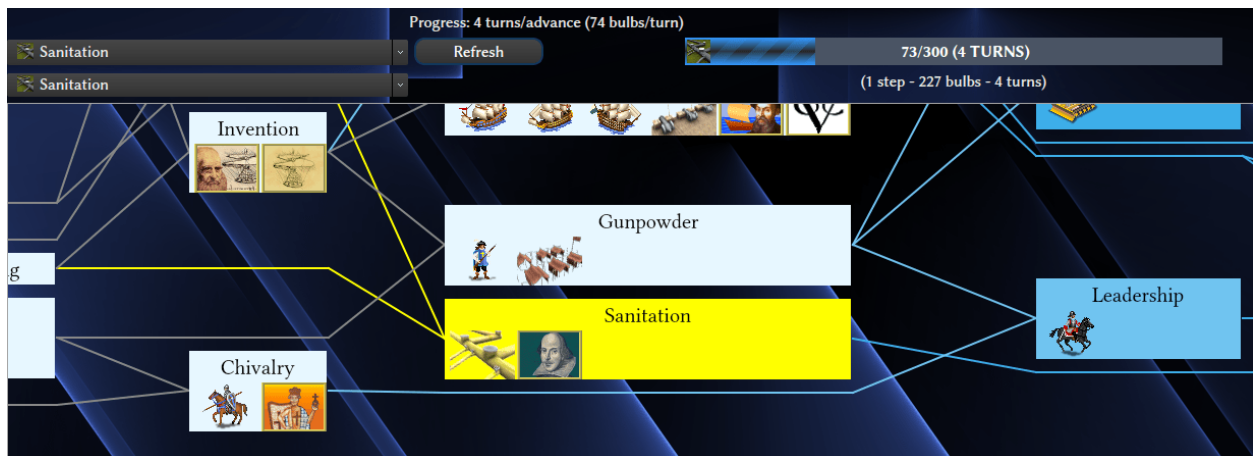


Fig. 3.38: Research Tree

If nothing is being researched, the *Research View* button will pulse to bring attention to it.

On the *Research Tree*, the top left drop down box is where you can pick from a menu of what technology advance you want your scientists to concentrate (*Current Research*). The bottom left drop down box is where you can set a future target (*Research Target*). The game will work through the list of technologies as turns progress in order of dependency.

To the right of the *Current Research* drop down box, you will find a *Refresh* button. In some circumstances a player can find that the current researched bulbs is greater than the bulbs needed to complete the task. The *Refresh* button will be enabled in this situation and the player can press to refresh the research on the current advance to complete the task.

The progress bar on the right will show you how many bulbs you are producing each turn, how many more bulbs you have left to finish the research target and, if enabled, how many bulbs are being used for technology upkeep.

If you do not want to use the drop down boxes to pick current and target technologies, you can left-click on the box for the technology in the *Research Tree* view.

If you hover your mouse over the icons in the *Research Tree*, a pop-up widget will appear giving you information pulled from the in game help. Only so much information is displayed, so you may be prompted to go to the in game help for more information.

National Budget View

The *National Budget View* on the *Top Function Bar* shows what percentage of gold, science, and luxury goods your nation is set at. Fig. 3.39 shows an example of the *National Budget View* button.

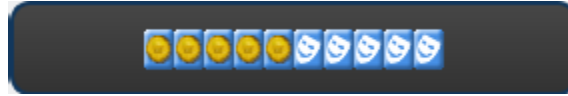


Fig. 3.39: National Budget View Button

Clicking on the *National Budget View* will bring up the *National Budget* dialog as shown in Fig. 3.19 in the *Civilization Menu* section.

National Status View

The *National Status View* on the *Top Function Bar* shows various information about your nation and the world via icons. The four icons from left to right are: Research Progress, Global Warming Chance, Nuclear Winter Chance, and Government. The Research Progress, Global Warming Chance, Nuclear Winter Chance icons will change depending on the rate and current status.



Fig. 3.40: National Status View

If you hover your mouse over the *National Status View*, a pop up widget will appear and give you information about your nation's status. The pop up widget shows: Population, Year, Turn Number, Total Gold, Net Income, National Budget, Research along with progress, Bulbs per Turn, Culture Score, Global Warming Chance, Nuclear Winter Chance, Current form of Government. Some of this information is a duplicate of what is shown on the *Map View*, *National Budget View*, and *Research View*. The values for Nuclear Winter and Global Warming chance give a good indication of what the icon looks like.

Messages

The *Messages* button on the *Top Function Bar* is used to toggle the message log widget. Fig. 3.41 below shows an example of the *Messages* button.



Fig. 3.41: Messages Button

Fig. 3.42 below shows an example of the *Messages* widget. If you double-click on a message for unit movement, city production and a few other message types the client will take you to the city or the unit on the *Map View*.

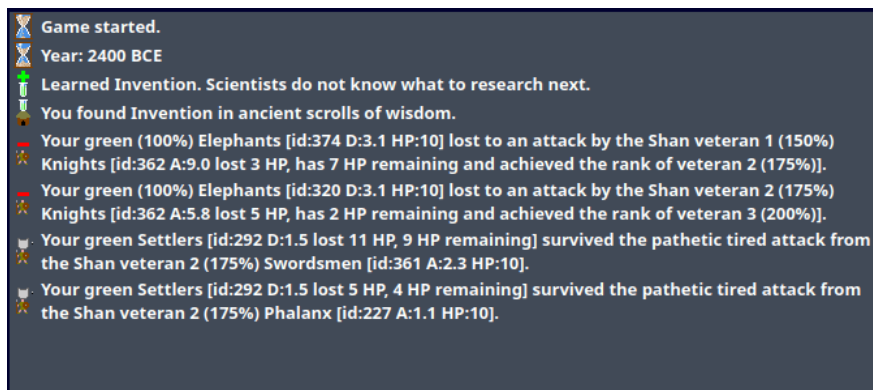


Fig. 3.42: Messages Widget

3.1.5 Unit Controls

When a unit has been selected on the *Map View*, a control widget will appear at the bottom center of the screen. The *Unit Controls* widget will give you some information about the unit, the terrain it is on and then some buttons corresponding to actions that the unit can take from the *Unit Menu*, *Combat Menu*, or the *Work Menu* respectively. Fig. 3.43 shows a Workers that has been selected.



Fig. 3.43: Unit Controls Widget

You can see that the Workers is selected because it has a white selection ring around its base. Looking at the dialog, in the header, you can see that this unit is ID # 111, has 4 3/9 Move Points (MPs), and 10 of 10 Hit Points (HPs).

From left to right you can see an image of the unit with MPs overlaid, the terrain it is on with infrastructure improvements shown, and then the actions that this unit can take. In this example the actions available are: Plant to Forest/River, Build Road, Go to Tile, Sentry, Auto Worker, Wait, and Done.

Depending on the type of unit selected, the available actions will change, but the other information will remain the same. If you rename a unit (from the *Unit Menu*), the name will appear in quotes after the Unit ID value.

If the unit selected is not in your field of vision on the map, then you can click on the icon for the unit on the left side and the game map will center on the unit for you. As with other widgets in Freeciv21, you can click+drag the widget to move it by using the plus symbol in the upper left corner.

3.1.6 Minimap

There is a *Minimap* widget in the lower right corner of the *Map View*. Fig. 3.44 below shows an example of the *Minimap* widget for a single player game. Fig. 3.45 shows an example of the *Minimap* widget for a Longturn multiplayer game with a fixed turn length. The difference between the two is the single player *Turn Done* button is replaced with a countdown timer. On the right hand side of the *Minimap* are zoom buttons similar to what you would find in the *View Menu*. Inside of the *Minimap* you will see an area inside a white border. This is the portion of the whole *Map View* that is displayed on the screen. Right-clicking inside the *Minimap* will cause the *Map View* to move to that spot. The *Minimap* provides a quick way to move around a large map in this manner.

You can also resize the *Minimap* widget by dragging from the top or left sides.

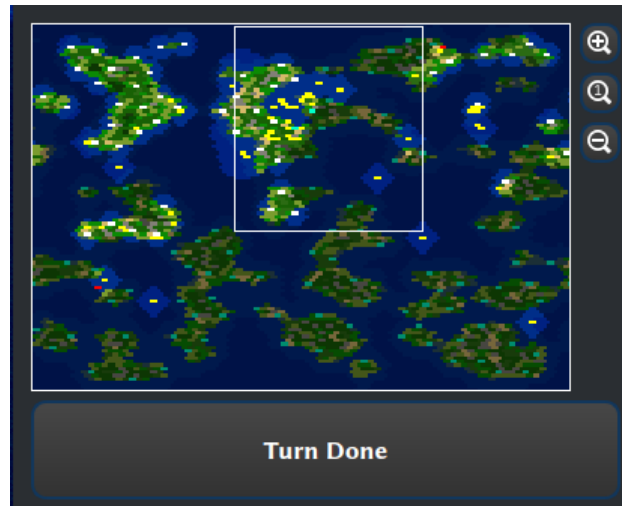


Fig. 3.44: Minimap in Single Player Mode

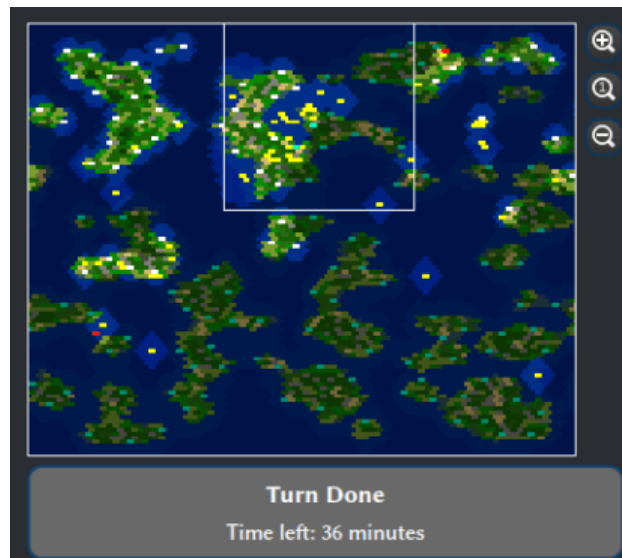


Fig. 3.45: Minimap in Multiplayer Mode

3.1.7 City Dialog

Getting to know the *City Dialog* is a major aspect of playing Freeciv21. As a player you will spend a great amount of time using this dialog box. The *City Dialog* is accessed by clicking on a city in the *Map View* or by double-clicking a city from the table in the *Cities View*. Fig. 3.46 shows a sample of the *City Dialog*. The dialog box is broken up into 5 major segments:

1. City information (top left)
2. Production and city supported items (left)
3. City citizen tile output (center)
4. General, Citizens, and Governor tabs (right)
5. Present units (bottom center).



Fig. 3.46: City Dialog

Let us start at the top left as highlighted in Fig. 3.47. In this segment of the dialog box is some general information about the city. The name of the city is in the header near the top. If you click on the city name, a dialog box will appear and allows you to rename the city to something else. You will see navigation buttons for previous and next city as well as a close button on the very top. Below the city name, the dialog will show how large the city is. Each rectangle icon is equivalent to 1 citizen (10,000 population). At the bottom of this segment is information about the city's status. From left to right you will see net food, net production, net gold, net science, net trade, and turns to grow to the next city size. If you hover your mouse over any of these icons, the client will show you detailed information on the calculation for the net value displayed.

Note: The city array will change over time as you gain and lose cities. It generally follows a left to right, top to bottom pattern on the map.

Moving to the segment highlighted in Fig. 3.48. You will see all the items that the city supports, starting with Wonders, Improvements and then Units. Next to this you will see the production queue of items the city will work on. You can

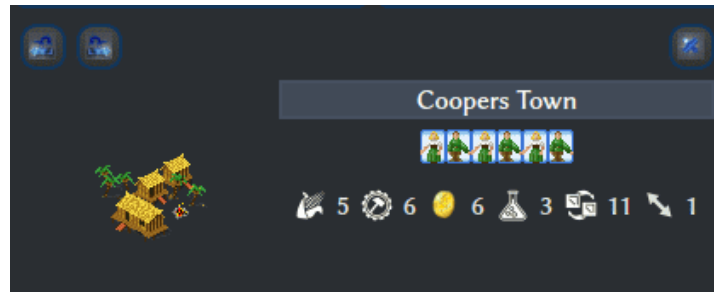


Fig. 3.47: City Dialog - Top Left

see how much gold it will cost to buy the current item being produced. If you click on the *Buy* button, a confirmation dialog will appear. In Fig. 3.48's example a Temple is being constructed for a total cost of 25 shields (production). The city has produced net 24 of 25 needed shields and at its current rate of net +6 production will take 1 turns to complete. The player for this example has also added multiple items to the work list. When the Temple is finished, the city will start production on Aqueduct, River. At this point, the city needs 1 shield to finish the Temple. At +6 shields per turn the city will produce 5 more shields that is needed. The surplus shields will transfer to the Aqueduct, River when the Temple is complete. If the production rate did not have extra shields left over, then no shields would go towards the Aqueduct, River, when the Temple is complete.

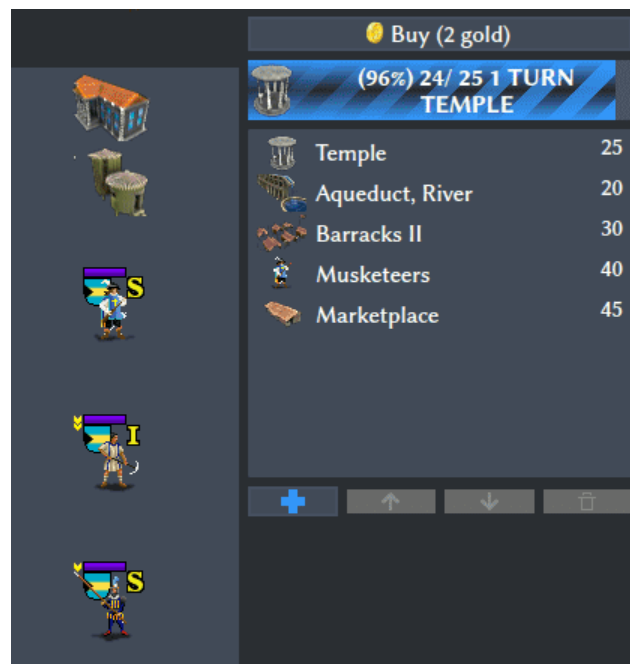


Fig. 3.48: City Dialog - Production

At the bottom of the segment are four buttons: *Add*, *Move Up*, *Move Down*, and *Delete*. Clicking on *Add*, will open a pop up menu showing all of the items that the city can produce. Clicking on the item will add it to the bottom of the work list above. If you wish to move this new item or another item in the work list up, click (select) it from the work list and click the *Move Up* button. Same action for the *Move Down* button. To remove the item from the work list, select an item in the work list and click on *Delete*. You can also double-click on an item in the work list and it will be removed from the work list.

City Production Work Lists

One more feature of the *Production* segment is the work list editor. You can save work lists for later use. To get started, populate the work list with things you want to build. This can be a collection of city improvements, units, and wonders. Once you have the list configured the way you like it, right-click on the work list canvas and a pop up menu will show. Start by selecting *Save Worklist*. A dialog box will pop up allowing you to give the list a name. If you clear out the work list on the *Production* segment and then right-click on the work list canvas, you can pick *Insert Worklist* or *Change Worklist* from the sub-menu. *Insert Worklist* will add the items in the saved work list to the main work list. *Change Worklist* will clear what is in the main work list and replace it with the saved work list.

Moving to the center, we can see the city citizen tile output segment and the city's full vision radius highlighted over the main map as shown in Fig. 3.49. This example shows a size 6 city, which means 6 tiles can be managed by the citizens. Each citizen can be assigned to work one tile, extracting food, production and trade from it (the numbers shown are in the same order). In addition, the city tile is always worked for free. In this example, there are irrigated grassland (3/0/1), irrigated grassland with river (3/0/2), and mined hills with roads and wine (1/3/4) tile being managed by a citizen. You also see the city center is on a forest river tile and gives 1/3/1 output. You can click on the city center and the game will automatically pick the best tiles for net food to aid city growth. You can also click on a tile to remove the citizen from the tile and then click another tile to have the citizen manage another tile. This is commonly referred to as city micromanagement. If you remove a citizen from managing a tile, take a look at the top left segment. The citizens icon bar will show one entertainer specialist. If you want to change the entertainer to a scientist or a taxman, you can click on it in the icon bar to change.



Fig. 3.49: City Dialog - City Center

Moving right, we can see the *General* tab on panel as highlighted in Fig. 3.50. This tab shows information similar to what is shown on the top left segment. Hovering your mouse over many of the items at the top of the *General* tab will show detailed calculation on how the net value is calculated. Hovering your mouse over the citizens value will give you information on the happiness of the city's citizens.

On the right side is the *Citizens* tab, as highlighted in Fig. 3.51. The *Citizens* tab shows you information about the happiness and nationality of the citizens. Happiness is broken down into segments: Cities (Empire Size), Luxury Goods, Buildings, Nationality, Units, and Wonders.

General	Citizens	Governor
Food:	17 (+5)	
Granary:	23/26	
Citizens:	6 (Content)	
Change in:	1 turn	
Production:	6 (+6)	
Waste:	0	
Trade:	12 (+11)	
Corruption:	1	
Gold:	8 (+6)	
Science:	3 (+3)	
Luxury:	0 (+0)	
Culture:	0	
Pollution:	0	
Plague risk:	0.6%	
Tech Stolen:	Not stolen	
Airlift:	s: 0 d: ∞	
Improvements:	upkeep 2	
Supported units:	3	

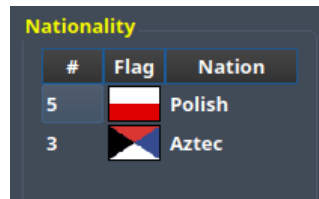
Fig. 3.50: City Dialog - General

General	Citizens	Governor
Happiness		
Cities:		
Luxuries:		
Buildings:		
Nationality:		
Units:		
Wonders:		

Fig. 3.51: City Dialog - Citizens

Overall happiness of the citizens in your cities depends heavily on all of these factors. Certain forms of government have varying degrees of empire size penalties and as your empire grows you will have to deal with the negative consequences of managing a large empire and the unhappiness it creates. The amount of luxury goods you are producing as part of your [National Budget View](#) will aid this problem. Certain city improvements can improve happiness, as well as units in the city (martial law), along with wonders. If units are in the field in battle against your enemies, they can cause unhappiness. Hovering your mouse over the appropriate row will give you more information about it.

Fig. 3.52 shows what it looks like when you have mixed nationality in your cities. Mixed nationality can cause unhappiness and occurs when you conquer an opponent's city.





#	Flag	Nation
5		Polish
3		Aztec

Fig. 3.52: City Dialog - Citizens Nationality

The *Governor* tab, as highlighted in Fig. 3.53, gives information on the Citizen Governor for this city. For more information on how to use the Citizen Governor refer to [Citizen Governor \(aka Citizen Management Agent, or CMA\)](#).

Moving to the bottom center, you will see a list of the units that are present in the city, as highlighted in Fig. 3.54. There can be units present in a city that the city is not supporting, or from your allies. If you right-click on a unit, a pop up box with action commands you can give to the unit will be displayed. If you double-click on a unit, the *City Dialog* will close and the unit will be selected. See [Unit Controls](#) for more information on what you can do with units. If you wish to select more than one unit to give a command to, you can do that by holding the ctrl key on your keyboard and then left-clicking on the units you want to select. When finished, right-click on one of the selected units and pick the option you want. This is a great way to activate a collection of units all at once. Depending on the number of units in the city, this widget will expand left and right to the width of the screen.

3.1.8 Game Message Options

As discussed in the [Message Options](#) section of the game manual, there is a large collection of message output options available. This page lists them in detail. Here is a sample of the top of the message options dialog.

Note: Messages are essentially a visual response to an event that occur during a turn or turn change in a game.

The column headings do the following:

- **Out** – The event message will be placed into the *Server Chat/Command Line* widget on the main map.
- **Mes** – The event message will be placed into the *Messages* widget as described in [Messages](#).
- **Pop** – The event message will be displayed in a pop-up message box.

The following is a detailed view of all of the message options available in Freeciv21. Each entry provides a short description of the message, the default setting and the name of the event.

AI Debug messages

This event outputs AI calculations and other messages related to the in-game AI (e.g. the computer running one or more nations in a game).

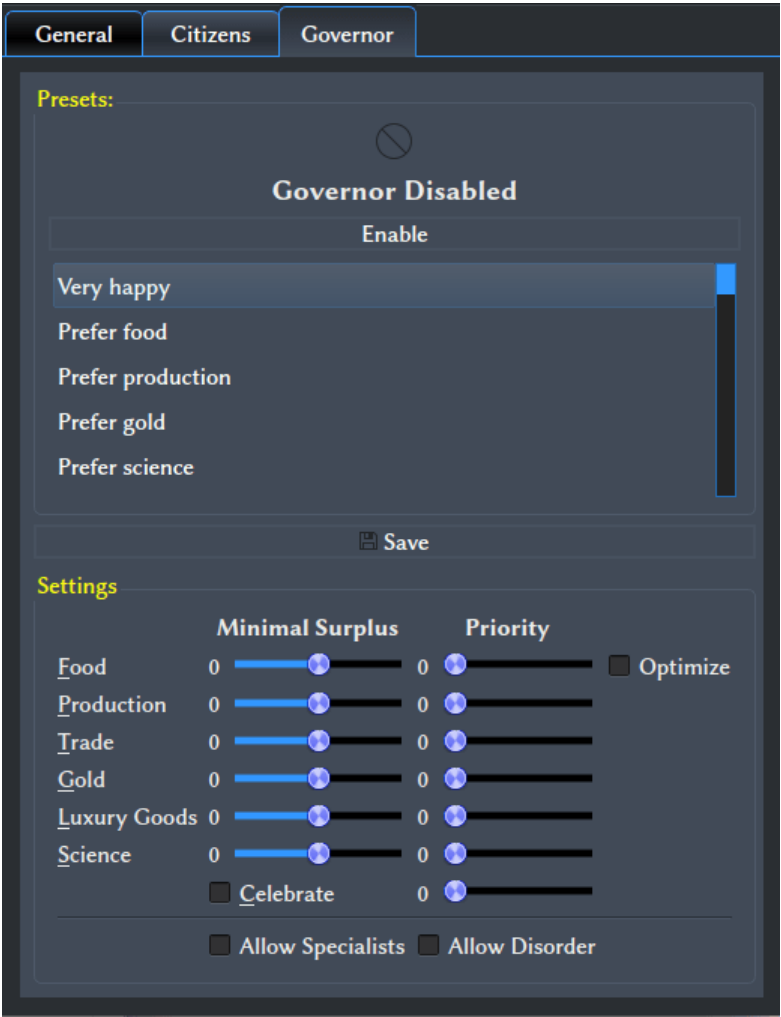


Fig. 3.53: City Dialog - Governor

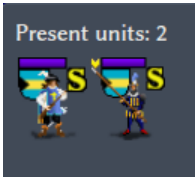


Fig. 3.54: City Dialog - Present Units

Event	Out	Mes	Pop
AI Debug messages	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Broadcast Report	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Caravan actions	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Chat error messages	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Chat messages	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
City: Building Unavailable Item	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Captured/Destroyed	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Celebrating	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: City Map changed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Civil Disorder	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Disaster	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Famine	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Famine Feared	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Growth	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: Has Plague	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
City: May Soon Grow	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Fig. 3.55: Message Options Dialog

- Default: Off/None
- Event Name: E_AI_DEBUG

Broadcast Report

This event outputs a historian report at given turn intervals. Historian reports are often a widget on the main map, however there is an associated message generated at the same time.

- Default: Mes
- Event Name: E_BROADCAST_REPORT

Caravan actions

This event outputs information on an action a Caravan has taken. Examples include establishing a trade route or helping to build a wonder.

- Default: Mes
- Event Name: E_CARAVAN_ACTION

Chat error messages

This event is associated with any kind of error message related to commands entered in the *Server Chat/Command Line* widget on the main map.

- Default: Out
- Event Name: E_CHAT_ERROR

Chat messages

This is simply chat messages between the public (all players on a server) and allies (all players on the same team or allied together).

- Default: Out
- Event Name: E_CHAT_MSG

City: Building Unavailable Item

You will receive this message if you attempt to build something in a city, but cannot build it for some reason. There are a number of reasons when this event can come up. Some examples are: you added future targets to your city worklist, but do not have the technology available to build the item; you already have the building in your city and accidentally asked the city to build it again; you already have a unique unit built and you ask the city to build another one.

- Default: Mes
- Event Name: E_CITY_CANTBUILD

City: Captured/Destroyed

The city named in the message has either been captured by your enemy or has been destroyed and turned into ruins.

- Default: Mes
- Event Name: E_CITY_LOST

City: Celebrating

The named city's citizens are so happy that they are celebrating in your honor! Some tilesets show a fireworks effect over the city when this event occurs to help you identify the city.

- Default: Mes
- Event Name: E_CITY_LOVE

City: City Map Changed

The working radius of the named city has changed. This can happen when certain city improvements are constructed in the city or with a certain technology being researched, which are often ruleset dependent.

- Default: Mes
- Event Name: E_CITY_RADIUS_SQ

City: Civil Disorder

The citizens of the named city are so angry that the city is in disorder. Most tilesets show a raised fist icon over the city when this event occurs to help you identify the city.

- Default: Mes
- Event Name: E_CITY_DISORDER

City: Disaster

The named city has experienced some kind of disaster. Some examples include: a fire has destroyed a city improvement; a flood has destroyed all food saved in the city's granary; an explosion in a Factory has caused some damage.

- Default: Mes
- Event Name: E_DISASTER

City: Famine

The named city has experienced famine. Famine occurs when the city is producing less food than is required to maintain (feed) the citizens within the city and the granary is empty. When this event occurs, the city will be reduced in size by one.

- Default: Mes
- Event Name: E_CITY_FAMINE

City: Famine Feared

Very similar to the message item above. If a city is close to experiencing famine, this message will alert you to the impending event. You will have time to act on the named city to prevent famine if possible when you see this message.

- Default: Mes

- Event Name: E_CITY_FAMINE_FEARED

City: Growth

The named city has grown! This event occurs when the accumulated food in the city's granary has met or exceeded the amount needed for the next food box size. You can see this information in the *City Dialog* box on the general tab. Look for the granary stats and the food surplus.

- Default: Mes

- Event Name: E_CITY_GROWTH

City: Has Plague

The named city has experienced a plague. A plague can cause population loss or prevent the city from growing to the next size. Some rulesets allow for city improvements that can reduce the chance of plague in a city.

- Default: Mes

- Event Name: E_CITY_PLAGUE

City: May Soon Grow

The named city is close to filling its granary and will grow to the next size.

- Default: Off/None

- Event Name: E_CITY_MAY_SOON_GROW

City: Needs Improvement to Grow

The named city is trying to grow, but cannot due to the lack of an improvement. In many rulesets, the Aqueduct and Sewer System are needed for cities to grow beyond certain sizes.

- Default: Mes

- Event Name: E_CITY_IMPROVEMENT

City: Needs Improvement to Grow, Being Built

The named city will soon grow and needs the city improvement that is currently being produced. The message may show many turns in advance, giving you ample time to determine if it needs to be rush bought or can finish on its own.

- Default: Mes

- Event Name: E_CITY_IMPROVEMENT_BLDG

City: Normal

A city that was previously in disorder is now no longer in disorder.

- Default: Mes

- Event Name: E_CITY_NORMAL

City: Nuked

The named city has been hit with a Nuclear bomb or other similar type of unit.

- Default: Mes

- Event Name: E_CITY_NUKED

City: Production changed

The named city has changed what is at the top of the worklist.

- Default: Off/None

- Event Name: E_CITY_PRODUCTION_CHANGED

City: Released from citizen governor

The named city was previously under control of the *Citizen Governor* (aka *Citizen Management Agent, or CMA*) and can no longer fulfill the requirements.

- Default: Mes

- Event Name: E_CITY_CMA_RELEASE

City: Suggest Growth Throttling

The named city is producing a Granary and may grow before the improvement is complete. You will want to ensure that the Granary is completed before the turn when the city will grow, or you will lose the benefits of the improvement for one city growth cycle.

- Default: Mes

- Event Name: E_CITY_GRAN_THROTTLE

City: Transfer

The named city has been transferred as part of a diplomatic agreement.

- Default: Mes

- Event Name: E_CITY_TRANSFER

City: Was Built

The named city has been founded by Settlers.

- Default: Off/None

- Event Name: E_CITY_BUILD

City: Worklist Events

The named city has had some kind of worklist change. This often occurs when you change the type of item being produced such as going from an improvement to a unit.

- Default: Off/None

- Event Name: E_WORKLIST

Connect/disconnect messages

Outputs when users connect and disconnect from a game server. These are often seen on Longturn multiplayer games.

- Default: Out

- Event Name: E_CONNECTION

Deprecated Modpack syntax warnings

An installed Modpack uses syntax that may stop working in future versions of the game.

- Default: Mes and Pop

- Event Name: E_DEPRECATION_WARNING

Diplomat Action: Bribe

Your Diplomat or Spy was successful in bribing an enemy unit.

- Default: Mes

- Event Name: E_MY_DIPLOMAT_BRIBE

Diplomat Action: Caused Incident

Your Diplomat or Spy was successful in causing an incident in a targeted city.

- Default: Mes

- Event Name: E_DIPLOMATIC_INCIDENT

Diplomat Action: Embassy

Your Diplomat or Spy was successful in establishing an embassy with another nation.

- Default: Mes

- Event Name: E_MY_DIPLOMAT_EMBASSY

Diplomat Action: Escape

Your Diplomat or Spy was successful in escaping detection from the enemy nation.

- Default: Mes

- Event Name: E_MY_DIPLOMAT_ESCAPE

Diplomat Action: Failed

Your Diplomat or Spy was unsuccessful in the named action taken.

- Default: Mes

- Event Name: E_MY_DIPLOMAT_FAILED

Diplomat Action: Gold Theft

Your Spy was successful in stealing gold from a targeted city.

- Default: Mes

- Event Name: E_MY_SPY_STEAL_GOLD

Diplomat Action: Incite

Your Diplomat or Spy was successful in inciting a targeted city to revolt.

- Default: Mes

- Event Name: E_MY_DIPLOMAT_INCITE

Diplomat Action: Map Theft

Your Spy was successful in stealing maps from an enemy nation.

- Default: Mes

- Event Name: E_MY_SPY_STEAL_MAP

Diplomat Action: Poison

Your Diplomat or Spy was successful in poisoning the citizens of a targeted city.

- Default: Mes

- Event Name: E_MY_DIPLOMAT_POISON

Diplomat Action: Sabotage

Your Diplomat or Spy was successful in sabotaging the production of a targeted city.

- Default: Mes

- Event Name: E_MY_DIPLOMAT_SABOTAGE

Diplomat Action: Suitcase Nuke

Your Spy was successful in deploying a suitcase tactical nuclear device in a targeted city.

- Default: Mes

- Event Name: E_MY_SPY_NUKE

Diplomat Action: Theft

Your Diplomat or Spy was successful in stealing a technology advance from an enemy nation.

- Default: Mes

- Event Name: E_MY_DIPLOMAT_THEFT

Diplomatic Message

This message appears when some kind diplomatic event has occurred. Examples include: accepting or canceling a diplomatic meeting; in-game allied AI asks for assistance; in-game AI threatens to kill you.

- Default: Mes

- Event Name: E_DIPLOMACY

Enemy Diplomat: Bribe

An enemy's Diplomat or Spy was successful in bribing one of your units.

- Default: Mes

- Event Name: E_ENEMY_DIPLOMAT_BRIBE

Enemy Diplomat

[Embassy] An enemy's Diplomat or Spy was successful in establishing an embassy with your nation.

- Default: Mes

- Event Name: E_ENEMY_DIPLOMAT_EMBASSY

Enemy Diplomat: Failed

An enemy's Diplomat or Spy was unsuccessful in the named action taken.

- Default: Mes

- Event Name: E_ENEMY_DIPLOMAT_FAILED

Enemy Diplomat: Gold Theft

An enemy's Spy was successful in stealing gold from a targeted city.

- Default: Mes

- Event Name: E_ENEMY_SPY_STEAL_GOLD

Enemy Diplomat: Incite

An enemy's Diplomat or Spy was successful in inciting a targeted city to revolt.

- Default: Mes

- Event Name: E_ENEMY_DIPLOMAT_INCITE

Enemy Diplomat: Map Theft

An enemy's Spy was successful in stealing your maps.

- Default: Mes

- Event Name: E_ENEMY_SPY_STEAL_MAP

Enemy Diplomat: Poison

An enemy's Diplomat or Spy was successful in poisoning the citizens of a targeted city.

- Default: Mes

- Event Name: E_ENEMY_DIPLOMAT_POISON

Enemy Diplomat: Sabotage

An enemy's Diplomat or Spy was successful in sabotaging the production of a targeted city.

- Default: Mes

- Event Name: E_ENEMY_DIPLOMAT_SABOTAGE

Enemy Diplomat: Suitcase Nuke

An enemy's Spy was successful in deploying a suitcase tactical nuclear device in a targeted city.

- Default: Mes

- Event Name: E_ENEMY_SPY_NUKE

Enemy Diplomat: Theft

An enemy's Diplomat or Spy was successful in stealing a technology advance from you.

- Default: Mes

- Event Name: E_ENEMY_DIPLOMAT_THEFT

Error message from bad command

This message appears when any kind of incorrect command you give the game occurs.

- Default: Mes

- Event Name: E_BAD_COMMAND

Extra Appears or Disappears

This message appears when you use Workers or Engineers to terraform terrain that had a special “extra” on the tile or when you terraform it back to the original terrain.

- Default: Mes

- Event Name: E_SPONTANEOUS_EXTRA

Game Ended

The game has ended. The final player report is shown.

- Default: Mes

- Event Name: E_GAME_END

Game Started

The game has started.

- Default: Off/None

- Event Name: E_GAME_START

Global: Eco-Disaster

Global Warming or Nuclear Winter has occurred.

- Default: Mes

- Event Name: E_GLOBAL_ECO

Global: Nuke Detonated

A player has detonated a Nuclear device on the map. Coordinates are given in the message.

- Default: Mes

- Event Name: E_NUKE

Help for beginners

Messages to aid new players.

- Default: Mes

- Event Name: E_BEGINNER_HELP

Hut: Barbarians in a Hut Roused

One of your units has entered a hut on the map and roused Barbarians.

- Default: Mes

- Event Name: E_HUT_BARB

Hut: City Founded from Hut

One of your units has entered a hut on the map and founded a city for you at that location.

- Default: Mes

- Event Name: E_HUT_CITY

Hut: Gold Found in Hut

One of your units has entered a hut on the map and found gold inside. The message will contain the amount of gold found.

- Default: Mes

- Event Name: E_HUT_GOLD

Hut: Killed by Barbarians in a Hut

One of your units has entered a hut on the map and was killed by Barbarians.

- Default: Mes

- Event Name: E_HUT_BARB_KILLED

Hut: Mercenaries Found in Hut

One of your units has entered a hut on the map and mercenaries were found that join your nation. Mercenaries are often the best attacking unit that you have the technology for.

- Default: Mes

- Event Name: E_HUT_MERC

Hut: Settler Found in Hut

One of your units has entered a hut on the map and found Settlers inside that can be used to build a city at a location of your choice.

- Default: Mes

- Event Name: E_HUT_SETTLER

Hut: Tech Found in Hut

One of your units has entered a hut on the map and found *scrolls of wisdom* containing a technology advance.

- Default: Mes

- Event Name: E_HUT_TECH

Hut: Unit Spared by Barbarians

One of your units has entered a hut on the map and was not killed by a band of Barbarians.

- Default: Mes

- Event Name: E_HUT_BARB_CITY_NEAR

Improvement: Bought

You have rush bought with gold a city improvement in the named city.

- Default: Off/None

- Event Name: E_IMP_BUY

Improvement: Built

The named city has completed construction of the listed city improvement.

- Default: Mes

- Event Name: E_IMP_BUILD

Improvement: Forced to Sell

Your national treasury did not have enough gold to maintain the upkeep of all of your city improvements at turn change, so the game sold one or more of them.

- Default: Mes

- Event Name: E_IMP_AUCTIONED

Improvement: New Improvement Selected

You did not tell a city to build a specific city improvement, so the in-game *Advisor* selected one for you.

- Default: Mes

- Event Name: E_IMP_AUTO

Improvement: Sold

You manually sold a named city improvement.

- Default: Off/None

- Event Name: E_IMP_SOLD

Message from server operator

The server operator has sent a broadcast message to all players. Longturn multiplayer games will use this feature sometimes.

- Default: Mes and Pop

- Event Name: E_MESSAGE_WALL

Nation Selected

You have selected (taken) a nation. This message typically comes up during Longturn multiplayer games when you are taking control of a nation either by picking up a idle player or acting as the regent for a team mate. In single player games, you can also take control of any of the in-game AI players as well and this message will show at that time too.

- Default: Out

- Event Name: E_NATION_SELECTED

Nation: Achievements

Your nation has crossed an achievement boundary. Different rulesets have varying types of achievements available.

- Default: Mes

- Event Name: E_ACHIEVEMENT

Nation: Barbarian Uprising

There has been a Barbarian uprising in the game. Ensure you have sufficient defensive units in your cities as they will attack when they find you.

- Default: Mes

- Event Name: E_UPRISING

Nation: Civil War

Your nation has been broken apart due to Civil War. Some of your cities have broken away from your nation and formed a new nation controlled by an in-game AI.

- Default: Mes

- Event Name: E_CIVIL_WAR

Nation: Collapse to Anarchy

Too many cities are in disorder and you can no longer maintain a functioning government. Your nation has fallen into anarchy.

- Default: Mes

- Event Name: E_ANARCHY

Nation: First Contact

One of your units has come into contact with the first enemy nation in the game.

- Default: Mes

- Event Name: E_FIRST_CONTACT

Nation: Learned New Government

Your scientists have learned a new technology advance that also allows for a new form of government. A good example is learning Republic allows you to form a new government of the same name.

- Default: Mes

- Event Name: E_NEW_GOVERNMENT

Nation: Low Funds

Your national treasury is low in gold. If you do not correct the issue, city improvements will be sold at turn change.

- Default: Mes

- Event Name: E_LOW_ON_FUNDS

Nation: Multiplier changed

Certain effects in a ruleset can create bonuses (e.g. multipliers). This message occurs when one or more multipliers has changed values. Details are given as part of the message.

- Default: Mes

- Event Name: E_MULTIPLIER

Nation: Pollution

The named city's production has caused pollution on a tile in its working radius.

- Default: Mes

- Event Name: E_POLLUTION

Nation: Revolution Ended

You started a revolution to form a new government and it is now over. The new form of government is given in the message.

- Default: Mes

- Event Name: E_REVOLT_DONE

Nation: Revolution Started

You have started a revolution to form a new government and are now in anarchy.

- Default: Mes

- Event Name: E_REVOLT_START

Nation: Spaceship Events

One or more of your cities has constructed a Space Component, Space Module, or Space Structural for your spaceship. This message will also come up when you launch your spaceship, and when the spaceship arrives at Alpha Centauri, or is destroyed along the way and does not make it.

- Default: Mes

- Event Name: E_SPACESHIP

Player Destroyed

Either you or another player has completely destroyed a player in the game.

- Default: Mes

- Event Name: E_DESTROYED

Report

You have asked for a non-modal report such as Demographics or Top Five Cities.

- Default: Mes

- Event Name: E_REPORT

Scenario/ruleset script message

This is a message from a Lua script inside of a scenario or a ruleset. The game tutorial uses these extensively.

- Default: Mes and Pop

- Event Name: E_SCRIPT

Server Aborting

There is a very bad error occurring on the server and it is aborting/shutting down.

- Default: Mes and Pop

- Event Name: E_LOG_FATAL

Server Problems

The server is experiencing some problems that are not fatal.

- Default: Out

- Event Name: E_LOG_ERROR

Server settings changed

The server settings have changed. The game admins of Longturn multiplayer games will sometimes have to alter the settings after a game has started.

- Default: Out

- Event Name: E_SETTING

Technology: Acquired New Tech

Your nation has acquired a new named technology advance. This can be through a diplomatic agreement, or from a Great Wonder such as the Great Library.

- Default: Mes

- Event Name: E_TECH_GAIN

Technology: Learned New Tech

Your scientists have researched a new named technology advance for you.

- Default: Mes

- Event Name: E_TECH_LEARNED

Technology: Lost a Tech

Your scientists are not able to maintain enough research (bulbs) to maintain knowledge and have now forgotten/lost a named technology advance.

- Default: Mes

- Event Name: E_TECH_LOST

Technology: Other Player Gained/Lost a Tech

Your embassy with another player relays a message that the player has gained or lost a named technology advance.

- Default: Mes

- Event Name: E_TECH_EMBASSY

Technology: Selected new Goal

You have given your scientists a new goal to research on the technology tree.

- Default: Mes

- Event Name: E_TECH_GOAL

Treaty: Alliance

You have formed an alliance pact with the named player.

- Default: Mes

- Event Name: E_TREATY_ALLIANCE

Treaty: Broken

You have broken a named diplomatic pact with a given player. For example you break a peace treaty and go to war.

- Default: Mes

- Event Name: E_TREATY_BROKEN

Treaty: Cease-fire

You have entered into a cease-fire pact with the named player.

- Default: Mes

- Event Name: E_TREATY_Ceasefire

Treaty: Embassy

You have established an embassy with the named player. This can occur via a Diplomat or from a diplomatic meeting once contact has been established.

- Default: Mes

- Event Name: E_TREATY_EMBASSY

Treaty: Peace

You have entered into a peace pact with the named player.

- Default: Mes

- Event Name: E_TREATY_PEACE

Treaty: Shared Vision

You have granted shared vision with the named player.

- Default: Mes

- Event Name: E_TREATY_SHARED_VISION

Turn Bell

This event gives a message of the turn number and year when the turn changes.

- Default: Mes

- Event Name: E_TURN_BELL

Unit: Action Failed

A named action by a unit has failed.

- Default: Mes

- Event Name: E_UNIT_ACTION_FAILED

Unit: Attack Failed

You tried to attack an enemy unit and your unit has been destroyed in the process. The game will give a detailed results message that looks like this:

```
Your attacking {veteran level} {unit name} [id:{number} D:{defense}
HP:{hit points}] failed against the {enemy nation} {veteran level}
{unit name} [id:{number} lost {hit points} HP, {hit points} HP
remaining]!
```

- Default: Out

- Event Name: E_UNIT_LOST_ATT

Unit: Attack Succeeded

Your unit attacked an enemy unit and won the battle. The game will give a detailed results message that looks like this:

```
Your attacking {veteran level} {unit name} [id:{number} A:{attack}
lost {hit points} HP, has {hit points} remaining] succeeded against
the {enemy nation} {veteran level} {unit name} [id:{number}
HP:{hit points remaining}].
```

- Default: Out

- Event Name: E_UNIT_WIN_ATT

Unit: Bought

You have rush bought with gold a unit in the named city.

- Default: Off/None

- Event Name: E_UNIT_BUY

Unit: Built

The named city has completed construction of the listed unit.

- Default: Mes

- Event Name: E_UNIT_BUILT

Unit: Built unit with population cost

The named city has completed construction of the listed unit that also cost city population. This is often Settlers or Migrants.

- Default: Mes

- Event Name: E_UNIT_BUILT_POP_COST

Unit: Defender Destroyed

Your unit has been attacked by an enemy player and while acting as a defender has been destroyed in the process. The game will give a detailed results message that looks like this:

```
Your {veteran level} {unit name} [id:{number} D:{defense}
HP:{hit points}] lost to an attack by the {enemy nation}
{veteran level} {unit name} [id:{number} A:{attack} lost
{hit points} HP, has {hit points} HP remaining].
```

- Default: Mes

- Event Name: E_UNIT_LOST_DEF

Unit: Defender Survived

Your unit has been attacked by an enemy player and while acting as a defender has survived. The game will give a detailed results message that looks like this:

```
Your {veteran level} {unit name} [id:{number} D:{defense} lost
{hit points} HP, {hit points} HP remaining] survived the pathetic
attack from the {enemy nation} {veteran level} {unit name}
[id:{number} A:{attack} HP:{hit points remaining}].
```

- Default: Mes

- Event Name: E_UNIT_WIN_DEF

Unit: Did Expel

You have successfully expelled an enemy unit to its nation's capital city.

- Default: Mes

- Event Name: E_UNIT_DID_EXPEL

Unit: Lost outside battle

This message name can be a bit misleading. You can lose a unit in varying scenarios that do not involve direct conflict. Examples include: you leave a unit inside the borders of a nation that you entered into a peace pact with; you transfer a city to another player, which also includes any units that are supported by that city; a unit is on a transporter unit such as a Galleon and the ship was sunk in an attack.

- Default: Mes

- Event Name: E_UNIT_LOST_MISC

Unit: Orders / goto events

This event occurs when you give units advanced orders using the *Unit* → *Goto and...* menu option.

- Default: Mes

- Event Name: E_UNIT_ORDERS

Unit: Production Upgraded

The named city is producing a unit that has been obsoleted by a technology advance. The newer unit is now being constructed. For example: a city is building a Phalanx, however your nation has recently discovered Feudalism. The city will change to producing Pikemen instead of Phalanx.

- Default: Mes

- Event Name: E_UNIT_UPGRADED

Unit: Promoted to Veteran

One of your units has been promoted to a higher veteran level. The message will give the veteran level.

- Default: Mes

- Event Name: E_UNIT_BECAME_VET

Unit: Relocated

One or more of your units has been relocated on the map. This is often caused when you use Engineers loaded on a Transport to terraform ocean to swamp. Global Warming can also cause units to be relocated.

- Default: Mes

- Event Name: E_UNIT_RELOCATED

Unit: Sentries awaken

A unit you have sentried has observed an enemy unit in its vision radius. The message will give details on the enemy unit that was observed.

- Default: Mes

- Event Name: E_UNIT_WAKE

Unit: Unit did

Currently unused.

- Default: Mes

- Event Name: E_UNIT_ACTION_TARGET_OTHER

Unit: Unit did heal

A unit was healed, e.g. gained hit points.

- Default: Mes

- Event Name: E_MY_UNIT_DID_HEAL

Unit: Unit did to you

An enemy unit has taken an action against city. For example, when an enemy Spy sabotages production of a city improvement.

- Default: Mes
- Event Name:

E_UNIT_ACTION_TARGET_HOSTILE

Unit: Unit escaped

An enemy unit has escaped.

- Default: Mes

- Event Name: E_UNIT_ESCAPED

Unit: Unit illegal action

This message will appear when you attempt to take an action with a unit that is not possible. For example, trying to capture a unit that is not able to be captured.

- Default: Mes

- Event Name: E_UNIT_ILLEGAL_ACTION

Unit: Unit was healed

One of your units has been completely healed, e.g. 100% of its hit points has been restored.

- Default: Mes

- Event Name: E_MY_UNIT_WAS_HEALED

Unit: Was Expelled

One of your units was expelled by an enemy nation and has been returned to your capital.

- Default: Mes

- Event Name: E_UNIT_WAS_EXPELLED

Unit: Your unit did

A unit of yours was able to take an action against another unit. This message is typically related to Diplomat and Spy units.

- Default: Mes
- Event Name:

E_UNIT_ACTION_ACTOR_SUCCESS

Unit: Your unit failed

A unit of yours was not able to take an action against another unit. This message is typically related to Diplomat and Spy units.

- Default: Mes
- Event Name:

E_UNIT_ACTION_ACTOR_FAILURE

Vote: New vote

The in-game voting process has been activated and you are asked to vote on a topic.

- Default: Out

- Event Name: E_VOTE_NEW

Vote: Vote canceled

The player who initiated the in-game voting process has canceled the vote on a topic.

- Default: Out

- Event Name: E_VOTE_ABORTED

Vote: Vote resolved

The in-game voting process has completed.

- Default: Out

- Event Name: E_VOTE_RESOLVED

Wonder: Finished

The named Great Wonder has been completed by the listed player.

- Default: Mes

- Event Name: E_WONDER_BUILD

Wonder: Made Obsolete

One of your named wonders (both Great and Small) has had its effect removed due to becoming obsolete. This often occurs when a new technology advance has been discovered or another wonder is constructed.

- Default: Mes

- Event Name: E_WONDER_OBSOLETE

Wonder: Started

Construction of the named Great Wonder has been started by the listed player.

- Default: Mes

- Event Name: E_WONDER_STARTED

Wonder: Stopped

Construction of the named Great Wonder has been stopped by the listed player.

- Default: Mes

- Event Name: E_WONDER_STOPPED

Wonder: Will Finish Next Turn

Construction of the named Great Wonder will be finished at the end of the turn by the listed player. If you are also building the same wonder, it will become obsolete and you cannot built it.

- Default: Mes

- Event Name: E_WONDER_WILL_BE_BUILT

Year Advance

This event shows a message that the year has advanced at turn change.

- Default: Off/None

- Event Name: E_NEXT_YEAR

3.1.9 Game Shortcut Options

As discussed in the *Shortcut Options* section of the game manual, there is a large collection of keyboard and mouse shortcut options available. This page lists them in detail. Here is a sample of the top of the shortcut options dialog.

To override the default (e.g. change a shortcut setting), click your mouse cursor on the box on the right and then type the shortcut you want to set the setting to. For example: You want to change the *Scroll Map* option to the *PgUp* key, you would click in the box and then press the *PgUp* key on your keyboard. To reset to defaults, you can click on the *Reset* button. Click *Save* and then *Close* when finished.

In most cases, the keyboard shortcut will also show on an associated menu entry depending on the type of shortcut.

Here are the available shortcuts:

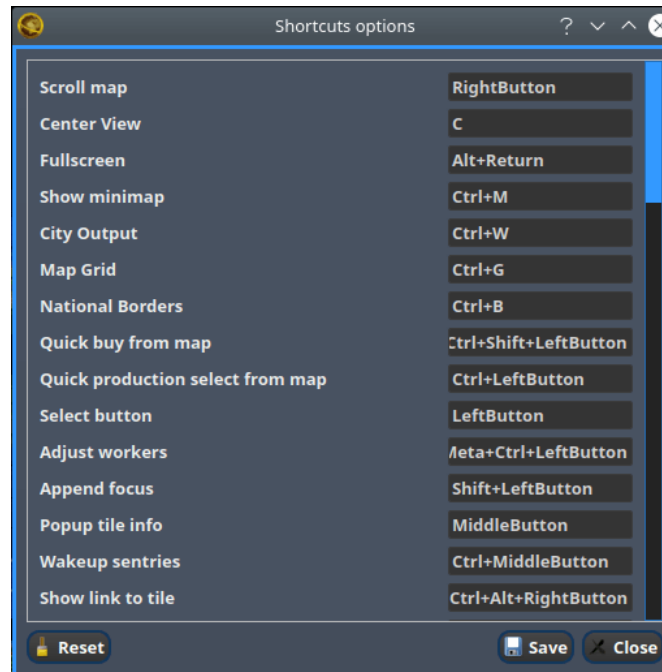


Fig. 3.56: Shortcut Options Dialog

Scroll Map

Scroll the map view. You can also use two-finger swipe on a trackpad if you have one (such as on a laptop) in all four directions to scroll the map.

Default: Right mouse button

Center View

With a unit selected, pressing this shortcut will center the unit on the screen. Similar to the option on the *View* menu.

Default: C

Fullscreen

Toggle fullscreen mode of the game interface.

Default: Alt+Return

Show minimap

Toggle the display of the minimap.

Default: Ctrl+M

City Output

Toggle the display of the city citizen managed tile output on the main map.

Default: Ctrl+W

Map Grid

Toggle the display of the map grid on the main map.

Default: Ctrl+G

National Borders

Toggle the display of the national borders of players on the main map.

Default: Ctrl+B

Quick buy from map

Quick rush buy production without opening up the city dialog. When used, whatever is selected from the menu will be immediately purchased from your national treasury.

Default: Ctrl+Shift plus Left mouse button

Quick production select from map

Quick production change for a city without opening up the city dialog. When used, whatever is selected from the menu will be immediately added to the top of the work list.

Default: Ctrl plus Left mouse button

Select button

Select an item on the map, such as a unit or a city.

Default: Left mouse button

Adjust workers

Auto adjust the citizens managing tiles in a city without opening up the city dialog. This is the same as opening a city in the city dialog and clicking on the city center tile.

Default: Meta+Ctrl plus Left mouse button (for example, Meta is the windows key for Windows keyboards).

Append focus

When applied to units, allow you to move a unit with a goto order without having to enter the *Unit* menu.

Default: Shift plus Left mouse button

Popup tile info

Pop up a widget of information about the tile, unit(s) on the tile, and other information related to the tile.

Default: Middle mouse button (different OSs handle this differently and is often a press of the scroll wheel).

Wakeup sentries

Quickly wake up all units that are sentried.

Default: Ctrl plus Middle mouse button (different OSs handle this differently and is often a press of the scroll wheel).

Show link to tile

Copies a link to a tile to the *Server Chat/Command Line* widget on the main map.

Default: Ctrl+Alt plus Right mouse button

Paste Production

Pastes a previously copied production item to the top of a selected city's worklist. See Copy Production below.

Default: Ctrl+Shift plus Right mouse button

Copy Production

Copy the item being produced in the selected city into a memory buffer. See Paste Production above.

Default: Shift plus Right mouse button

Show/hide workers

Toggle the display of Workers or Engineers on the main map.

Default: Alt+Shift plus Right mouse button

Units selection (for tile under mouse position)

Opens a widget to allow you to select a unit from a stack for movement.

Default: Ctrl+Spacebar

City Traderoutes

Toggles the display of traderoutes for your cities.

Default: Ctrl+D

City Production Levels

Toggles the display of city production in the city bar.

Default: Ctrl+P

City Names

Toggles the display of the city name in the city bar.

Default: Ctrl+N

Done Moving

Tell the selected unit that you are finished with it, even if it still has move points available.

Default: Spacebar

Go to/Airlift to City...

Instruct a selected unit to go to or airlift to a city. Go to can happen on any tile on the main map. Airlift requires the unit to be inside a city that has an Airport.

Default: T

Auto Explore

Instruct a unit to have the computer control where it explores around the map.

Default: X

Patrol

Tell a unit to patrol. When used, you will give the unit a patrol route with the mouse.

Default: Q

Unsentry All on Tile

Unsentry all sentried units on the tile under the mouse pointer.

Default: Ctrl+Shift+D

Do...

Instruct the unit to do some action. Be sure to have the *Messages* widget open when using this shortcut as there are often instructions given there.

Default: D

Upgrade

Upgrade the selected unit to a new type. The unit must be in one of your cities and you need to have sufficient gold in your national treasury.

Default: Ctrl+U

Set Home City

Changes a unit's supporting home city to a new home city. The unit must be inside of the city that will become its new supporting home city.

Default: H

Build Mine

Instruct a Workers or Engineers to build a mine on the tile.

Default: M

Plant

Instruct a Workers or Engineers to plant on the tile.

Default: Shift+M

Build Irrigation

Instruct a Workers or Engineers to build irrigation on the tile.

Default: I

Cultivate

Instruct a Workers or Engineers to cultivate on the tile.

Default: Shift+I

Build Road

Instruct a Workers or Engineers to build a road/railroad/maglev on the tile.

Default: R

Build City

Instruct a Settlers to build a new city on the tile.

Default: B

Sentry

Instruct a unit to sentry on the tile.

Default: S

Fortify

Instruct a unit to fortify on the tile.

Default: F

Go to Tile

Instruct a unit to go to a selected tile with the Left mouse button.

Default: G

Wait

Tell a unit to wait as you are not ready to give it orders yet.

Default: W

Transform

Instruct a Workers or Engineers to transform a tile.

Default: O

Explode Nuclear

Explode a Nuclear bomb on the tile where the unit is located. Assumes you have moved the unit to the tile where you want to explode it.

Default: Shift+N

Load

Load a unit on a transporter, such as a Transport.

Default: L

Unload

Unload a unit from a transporter, such as a Transport.

Default: U

Quick buy current production from map

Quick rush buy the current item being produced without opening up the city dialog. When used, the item will be immediately purchased from your national treasury.

Default: BackButton (Different OSs handle this differently. Often there is a similar named button on your mouse.)

Lock/Unlock interface

Lock the placement of the widgets on the screen so they cannot be accidentally moved.

Default: Ctrl+Shift+L

Auto Worker

Instruct a Workers or Engineers to allow the computer to control their work.

Default: A

Paradrop/clean pollution

If you have a Paratroopers selected you can send it to a tile of your choice with the Left mouse button. Alternately, if you have a Workers or Engineers selected, you can instruct them to clean pollution on the tile they are on. In some rulesets, Transport units can also remove pollution from ocean tiles.

Default: P

Popup combat information

When used just before a combat round, will output the combat results into the *Server Chat/Command Line* widget on the main map.

Default: Ctrl+F1

Reload Theme

Reloads the current theme.

Default: Ctrl+Shift+F5

Reload Tileset

Reloads the current tileset.

Default: Ctrl+Shift+F6

Toggle city full bar visibility

Toggles the display of the city bar.

Default: Ctrl+F

Zoom In

Zoom in on the map.

Default: + (plus sign)

Zoom Out

Zoom out on the map.

Default: - (minus sign)

Load Lua Script

Load a pre-written Lua script file and runs it.

Default: Ctrl+Shift+J

Load last loaded Lua Script

Re-run the previously selected Lua script file.

Default: Ctrl+Shift+K

Reload tileset with default scale

Reload the current tileset and go to the default zoom level.

Default: `Ctrl+Backspace`

Go And Build City

Instruct a Settlers to go to a tile and as soon as possible build a city there.

Default: `Shift+B`

Go and Join City

Instruct a Settlers or Migrants to go to a selected city as as soon as possible add to the city's population.

Default: `Shift+J`

Pillage

Instruct a unit to pillage infrastructure improvements on a tile.

Default: `Shift+P`

Clean Nuclear Fallout

Instruct a unit to clean nuclear fallout on a tile.

Default: `N`

3.2 Server Manual

The Freeciv21 Game Server (`freeciv21-server`) is the engine that powers all games that are played. This manual will mostly track what is in the `master` branch of the Freeciv21 GitHub Repository at <https://github.com/longturn/freeciv21/tree/master>. You can see the date that this file was last updated at the bottom of this page in the footer.

Note: This manual is for advanced users of Freeciv21. Most players simply install the game and either play local single-player games or join online Longturn multi-player games and do not really worry at all about the server. If you are good at Linux administration, interested in hosting your own server, or better understand how the Longturn team goes about creating the online games, then read on!

If you are having trouble, come find the Longturn Community on Discord at <https://discord.gg/98krqGm>. A good place to start is the `#questions-and-answers` or `#technicalities` channels.

3.2.1 Server Introduction

In order to play Freeciv21 over a network, one machine needs to act as the *server*: the state of the game lives on the server and players can connect to play. There are many ways to operate such a server. This page gives an introduction to the topic, guided by the experience gathered by running the [Longturn](#) website.

In order to run a server, you need a computer that other players can reach over the network. If you are planning to play at home, this can usually be any machine connected to your WiFi or local cabled network. Otherwise, the easiest is to rent a small server from a hosting provider. We strongly recommend that you choose a Linux-based server, as that is what we have experience with. Linux is also the only platform that currently support *user authentication*. You will need the ability to run your own programs on the server, so SSH access is a must. Apart from that, Freeciv21 is quite light on resources, so you will hardly hit the limits of even the cheapest options.

Whether you choose to use your own machine or to rent one, the basic principle of operating a server is the same: you need to run a program called `freeciv21-server` for as long as the game will last. This program will wait for players to connect and handle their moves in the exact same way as in a single player game. In fact, Freeciv21 uses an internal server for single player games!

Installation

Installing Freeciv21 on a server is done the normal way, as documented in *Installing Freeciv21*. Because the official packages come with the complete game, they will pull a number of dependencies that are normally not used on servers (for instance, a display server). This software is installed for packaging reasons, but will not be used.

Starting the Server

Once Freeciv21 is installed, it is immediately ready to run a simple server, which one can do by running the following command:

```
$ ./freeciv21-server
```

This command will start a server on the default port for Freeciv21, which is 5556. You can provide a custom port number by passing it to the `-p` argument:

```
$ ./freeciv21-server -p 5000
```

A server started with this command can be reached by pointing the game to port 5000 of your server machine's IP address or hostname. We suggest to start the server from within a terminal multiplexer such as `tmux` or `screen`, which will let it run independently of an active terminal window.

User Permissions

Freeciv21 supports several access levels for players connected to a server, restricting which commands they are allowed to run:

none

The user may not issue any command.

info

The user may use informational commands only.

basic

The user may use informational commands as well as commands affecting the game. Commands affecting the game start a vote if more than one user is connected.

ctrl

Same as **basic**, but the vote is bypassed for commands affecting the game.

admin

May use any command, except for `quit`, `rfcstyle`, and `write`. This includes potentially destructive commands such as `save` and `fcdb` – use with care.

hack

May use all commands without restriction.

By default, users connected to your server have access level **basic**. This can be changed using the `cmdlevel` command:

```
cmdlevel info new
```

This command grants access level **info** to any newly connecting player. A few more options are available; please see `help cmdlevel` from the server command line or the game server/chatline for more information.

Note: The `take` and `observe` commands require access level `info` only. Their use can be restricted using the `allowtake` server option or, in more advanced setups, using the `user_take` [Authentication and Database Support \(fddb\)](#) hook.

Authentication

By default, Freeciv21 servers are open to anyone without any sort of authentication. It is possible to make Freeciv21 ask for a password. There is also a built-in registration procedure with which one can create an account and set a password.

Authentication needs vary greatly depending on what you do with your server. If you use passwords merely to prevent players from peeking on each other, you may be fine with a very basic setup. If you are planning to run a full-scale server allowing registration through a website, you will need much more advanced settings. Freeciv21 ought to support all use cases within a unified interface. To allow for the necessary flexibility, authentication support is implemented using a Lua script.

See the [fddb documentation](#) for more information about setting up authentication.

3.2.2 Server Command Line Options

The `freeciv21-server` program has a collection of command-line options that can be used to control the behavior of the server when run. To get a listing of all the options, you can use the `--help` option, such as:

```
$ ./freeciv21-server --help
```

Tip: It is generally considered a best practice to write a `.sh` script to run your server. This way you do not have to remember all the command line options to use every time you run your server.

-A, --Announce <PROTO>

Announce game in LAN using protocol PROTO (IPv4/IPv6/none). Default is IPv4. Options include: IPv4, IPv6, or none.

-B, --Bind-meta <ADDR>

Connect to metaserver from this address. Default will be your public IP address assigned from your internet service provider.

Note: With regard to the command line arguments concerning the metaserver, Freeciv21 does not have its own metaserver at this time and the legacy Freeciv metaserver does not support Freeciv21. This means, right now, that all commands related to the metaserver are held over from the fork from legacy Freeciv until the Longturn community creates a custom metaserver.

-b, --bind <ADDR>

Listen for clients on ADDR. Default is the IP address of the server computer.

-d, --debug <LEVEL>

Set debug log level (fatal/critical/warning/info/debug). Default log level is `info`.

-e, --exit-on-end

When a game ends, exit instead of restarting.

- F, --Fatal**
Raise a signal on failed assertion. An assertion is a code calculation error. With this set, the server process will SEGFAULT instead of issuing a warning message to the terminal console.
- f, --file <FILE>**
Load saved game FILE. Useful when wanting to restart a game.
- i, --identity <ADDR>**
Be known as ADDR at metaserver or LAN client.
- k, --keep**
Keep updating game information on metaserver even after failure. Default is not to keep updating.
- l, --log <FILE>**
Use FILE as logfile. Generally a very good idea when running a server.
- M, --Metaserver <ADDR>**
Set ADDR as metaserver address. Allows you to point to a specific metaserver instead of the default: <https://meta.freeciv.org/>
- m, --meta**
Notify metaserver and send server's info.
- p, --port <PORT>**
Listen for clients on port PORT. Default is 5556 as assigned by [IANA](#)
- q, --quitidle <TIME>**
Quit if no players for TIME seconds.
- R, --Ranklog <FILE>**
Use FILE as ranking logfile. Generally a very good idea when running a server.
- r, --read <FILE>**
Read startup file FILE.
- S, --Serverid <ID>**
Sets the server id to ID.
- s, --saves <DIR>**
Save games to directory DIR. Generally a very good idea to save games at least every turn and depending on how long the turns are set to, to save within a turn. In the case of a server crash, restarting from a save comes in very handy.
- t, --timetrack**
Prints stats about elapsed time on misc tasks. Typically used to test code performance.
- w, --warnings**
Warn about deprecated modpack constructs.
- ruleset <RULESET>**
Load ruleset RULESET. Default is the Civ2Civ3 ruleset.
- scenarios <DIR>**
Save scenarios to directory DIR.
- a, --auth**
Enable database authentication (requires -Database).
- D, --Database <FILE>**
Enable database connection with configuration from FILE.
- G, --Guests**
Allow guests to login if auth is enabled. See

-N, --Newusers

Allow new users to login if auth is enabled.

-h, --help

Display help on command line options and quits.

--help-all

Display help including Qt specific options and quits.

-v, --version

Display version information and quits.

3.2.3 Server Commands

After a server has been started from the command-line, an administrator can issue a set of commands to the server's own command-line. This server command-line is separate from the OS terminal command-line.

/start

This command starts the game. When starting a new game, it should be used after all human players have connected, and *AI* players have been created (if required), and any desired changes to initial server options have been made. After **/start**, each human player will be able to choose their nation, and then the game will begin. This command is also required after loading a savegame for the game to recommence. Once the game is running this command is no longer available, since it would have no effect.

/help

With no arguments gives some introductory help. With argument “commands” or “options” gives respectively a list of all commands or all options. Otherwise the argument is taken as a command name or option name, and help is given for that command or option. For options, the help information includes the current and default values for that option. The argument may be abbreviated where unambiguous.

/list colors

List the player colors.

/list connections

Gives a list of connections to the server.

/list delegations

List of all player delegations.

/list ignored users

List of a player's ignore list.

/list map image definitions

List of defined map images.

/list players

The list of the players in the game.

/list rulesets

List of the available rulesets (for **/read** command).

/list scenarios

List of the available scenarios.

/list nationsets

List of the available nation sets in this ruleset.

/list teams

List of the teams of players.

/list votes

List of the running votes.

/quit

Quit the game and shutdown the server.

/cut <connection-name>

Cut specified client's connection to the server, removing that client from the game. If the game has not yet started that client's player is removed from the game, otherwise there is no effect on the player. Note that this command now takes connection names, not player names. See `/list connections`.

/explain <option-name>

The `/explain` command gives a subset of the functionality of `/help`, and is included for backward compatibility. With no arguments it gives a list of options (such as `/help options`), and with an argument it gives help for a particular option (such as `/help <option-name>`).

/show all|vital|situational|rare|changed|locked|rulesetdir

With no arguments, shows vital server options (or available options, when used by clients). With an option name argument, show only the named option, or options with that prefix. With `all`, it shows all options. With `vital`, `situational`, or `rare`, a set of options with this level. With `changed`, it shows only the options which have been modified from the ruleset defaults. While with `locked` all settings locked by the ruleset will be listed. With `ruleset`, it will show the current ruleset directory name.

/wall <message>

For each connected client, pops up a window showing the message entered.

/connectmsg <message>

Set message to send to clients when they connect. Empty message means that no message is sent.

/vote yes|no|abstain [vote number]

A player with basic *level access* issuing a control level command starts a new vote for the command. The `/vote` command followed by “yes”, “no”, or “abstain”, and optionally a vote number, gives your vote. If you do not add a vote number, your vote applies to the latest vote. You can only suggest one vote at a time. The vote will pass immediately if more than half of the voters who have not abstained vote for it, or fail immediately if at least half of the voters who have not abstained vote against it.

Note: Voting is not a feature that is used very often, but does come in handy.

/cancelvote <vote number>

With no arguments this command removes your own vote. If you have an admin access level, you can cancel any vote by vote number, or all votes with the `all` argument.

/debug diplomacy|ferries|tech|city|units|unit|timing|info

Print *AI* debug information about given entity and turn continuous debugging output for this entity on or off.

- debug diplomacy <player>
- debug ferries
- debug tech <player>
- debug city <x> <y>
- debug units <x> <y>
- debug unit <id>
- debug timing
- debug info

/set <option-name> <value>

Set an option on the server. The syntax and legal values depend on the option. See the help for each option. Some options are “bitwise”, in that they consist of a choice from a set of values. Separate these with |, for instance, /set topology wrapx|iso. For these options, use syntax like /set topology "" to set no values.

/team <player> <team>

A team is a group of players that start out allied, with shared vision, embassies, and fight together to achieve team victory with averaged individual scores. Each player is always a member of a team (possibly the only member). This command changes which team a player is a member of. Use "" if names contain whitespace.

/rulesetdir <directory>

Choose new ruleset directory or modpack.

/metamessage <meta-line>

Set user defined metaserver info line. If parameter is omitted, previously set metamessage will be removed. For most of the time user defined metamessage will be used instead of automatically generated messages, if it is available.

Note: Freeciv21 does not have its own metaserver at this time and the legacy Freeciv metaserver does not support Freeciv21. This means, right now, that all commands related to the metaserver are held over from the fork from legacy Freeciv until the Longturn community creates a custom metaserver.

/metapatches <meta-line>

Set metaserver patches line. See Note about Freeciv21 metaserver above.

/metaconnection up|down|persistent|?

/metaconnection ? reports on the status of the connection to the metaserver. /metaconnection down or /metac d brings the metaserver connection down. /metaconnection up or /metac u brings the metaserver connection up. /metaconnection persistent or /metac p is like ‘up’, but keeps trying after failures. See Note about Freeciv21 metaserver above.

/metaserver <address>

Set address (URL) for metaserver to report to. Same as --Metaserver on the command-line. See NOTE about Freeciv21 metaserver above.

/aitoggle <player-name>

Toggle *AI* status of player. By default, new players are AI.

/take <player-name>

Only the console and connections with cmdlevel *hack* can force other connections to take over a player. If you are not one of these, only the <player-name> argument is allowed. If - is given for the player name and the connection does not already control a player, one is created and assigned to the connection. The /allowtake *option* controls which players may be taken and in what circumstances.

For example, if you have cmdlevel *hack* and are connected to a server, you can issue /take <player-name> - to take over any player. If you do not have cmdlevel *hack*, then the /allowtake *option* must be properly set as well as a proper /delegate *command* by the player wishing to delegate is completed first. Then a player can use /take to take the player while the delegation is in place.

/observe <player-name>

Only the console and connections with cmdlevel *hack* can force other connections to observe a player. If you are not one of these, only the <player-name> argument is allowed. If the console gives no player-name or the connection uses no arguments, then the connection is attached to a global observer. The /allowtake *option* controls which players may be observed and in what circumstances.

For example, if you have cmdlevel *hack* and are connected to a server, you can issue /observe with no <player-name> parameter. The server will change your connection to a global observer, able to view all nations. A global observer can make no changes and can only see information. If a user with cmdlevel *hack* issues

`/observer <player-name>`, then they can only observe that particular nation only. To restore to original connection, you issue `/take <player-name>` for your own username.

`/detach <connection-name>`

Only the console and connections with `cmdlevel hack` can force other connections to detach from a player.

This rarely used command essentially forces a connected client to disconnect from a server. To see the connections, issue a `/list connections` command as noted *above*.

`/create <player-name> [ai type]`

With the `/create` command a new player with the given name is created. If `player-name` is empty, a random name will be assigned when the game begins. Until then the player will be known by a name derived from its type. The `ai type` parameter can be used to select which *AI* module will be used for the created player. This requires that the respective module has been loaded or built in to the server. If the game has already started, the new player will have no units or cities. Also, if no free player slots are available, the slot of a dead player can be reused (removing all record of that player from the running game).

`/away`

Toggles away mode for your nation. In away mode, the *AI* will govern your nation but make only minimal changes.

Note: The term *minimal changes* is not well understood at this time. The server help does not provide more details. An enterprising enthusiast could read the *AI* code to determine what the term means and provide more details. Any real player is not going to want the AI to run their nation and will *delegate* instead.

`/handicapped <player-name>`

With no arguments, sets all *AI* players to skill level *Handicapped*, and sets the default level for any new AI players to *Handicapped*. With an argument, sets the skill level for the specified player only. This skill level has the same features as *Novice*, but may suffer additional ruleset-defined penalties.

- Does not build offensive diplomatic units.
- Gets reduced bonuses from huts.
- Prefers defensive buildings and avoids close diplomatic relations.
- Can see through *FOW*.
- Does not build air units.
- Has complete map knowledge, including unexplored territory.
- Naive at diplomacy.
- Limits growth to match human players.
- Believes its cities are always under threat.
- Always offers cease-fire on first contact.
- Does not bribe worker or city founder units.
- Has erratic decision-making.
- Research takes 250% as long as usual.
- Has reduced appetite for expansion.

`/novice <player-name>`

With no arguments, sets all *AI* players to skill level *Novice*, and sets the default level for any new AI players to *Novice*. With an argument, sets the skill level for the specified player only.

- Does not build offensive diplomatic units.

- Gets reduced bonuses from huts.
- Prefers defensive buildings and avoids close diplomatic relations.
- Can see through *FOW*.
- Does not build air units.
- Has complete map knowledge, including unexplored territory.
- Naive at diplomacy.
- Limits growth to match human players.
- Believes its cities are always under threat.
- Always offers cease-fire on first contact.
- Does not bribe worker or city founder units.
- Has erratic decision-making.
- Research takes 250% as long as usual.
- Has reduced appetite for expansion.

/easy <player-name>

With no arguments, sets all *AI* players to skill level Easy, and sets the default level for any new AI players to Easy. With an argument, sets the skill level for the specified player only.

- Does not build offensive diplomatic units.
- Gets reduced bonuses from huts.
- Prefers defensive buildings and avoids close diplomatic relations.
- Can see through *FOW*.
- Does not build air units.
- Has complete map knowledge, including unexplored territory.
- Naive at diplomacy.
- Limits growth to match human players.
- Always offers cease-fire on first contact.
- Does not bribe worker or city founder units.
- Can change city production type without penalty.
- Has erratic decision-making.
- Has reduced appetite for expansion.

/normal <player-name>

With no arguments, sets all *AI* players to skill level Normal, and sets the default level for any new AI players to Normal. With an argument, sets the skill level for the specified player only.

- Does not build offensive diplomatic units.
- Can see through *FOW*.
- Has complete map knowledge, including unexplored territory.
- Can skip anarchy during revolution.
- Always offers cease-fire on first contact.

- Does not bribe worker or city founder units.
- Can change city production type without penalty.

/hard <player-name>

With no arguments, sets all *AI* players to skill level **Hard**, and sets the default level for any new AI players to **Hard**. With an argument, sets the skill level for the specified player only.

- Has no restrictions on national budget.
- Can target units and cities in unseen or unexplored territory.
- Knows the location of huts in unexplored territory.
- Can see through *FOW*.
- Has complete map knowledge, including unexplored territory.
- Can skip anarchy during revolution.
- Can change city production type without penalty.

/cheating <player-name>

With no arguments, sets all *AI* players to skill level **Cheating**, and sets the default level for any new AI players to **Cheating**. With an argument, sets the skill level for the specified player only.

- Can target units and cities in unseen or unexplored territory.
- Knows the location of huts in unexplored territory.
- Can see through *FOW*.
- Has complete map knowledge, including unexplored territory.
- Can skip anarchy during revolution.
- Can change city production type without penalty.

/experimental <player-name>

With no arguments, sets all *AI* players to skill level **Experimental**, and sets the default level for any new AI players to **Experimental**. With an argument, sets the skill level for the specified player only. **THIS IS ONLY FOR TESTING OF NEW AI FEATURES!** For ordinary servers, this level is no different to **Hard**.

- Has no restrictions on national budget.
- Can target units and cities in unseen or unexplored territory.
- Knows the location of huts in unexplored territory.
- Can see through *FOW*.
- Has complete map knowledge, including unexplored territory.
- Can skip anarchy during revolution.
- Can change city production type without penalty.

/cmdlevel none|info|basic|ctrl|admin|hack

The command access level controls which server commands are available to users via the client chatline. The available levels are:

- **none**: no commands
- **info**: informational or observer commands only
- **basic**: commands available to players in the game
- **ctrl**: commands that affect the game and users

- **admin:** commands that affect server operation
- **hack:** *all* commands - dangerous!

With no arguments, the current command access levels are reported. With a single argument, the level is set for all existing connections, and the default is set for future connections. If **new** is specified, the level is set for newly connecting clients. If **first come** is specified, the **first come** level is set. It will be granted to the first client to connect, or if there are connections already, the first client to issue the **/first** command. If a connection name is specified, the level is set for that connection only. Command access levels do not persist if a client disconnects, because some untrusted person could reconnect with the same name. Note that this command now takes connection names, not player names.

/first

If there is none, become the game organizer with increased permissions.

/timeoutshow

Shows information about the timeout for the current turn, for instance how much time is left.

/timeoutset <time>

This command changes the remaining time for the current turn. Passing a value of **0** ends the turn immediately. The time is specified as hours, minutes, and seconds using the format **hh:mm:ss** (minutes and hours are optional).

/timeoutadd <time>

This increases the timeout for the current turn, giving players more time to finish their actions. The time is specified as hours, minutes, and seconds using the format **hh:mm:ss** (minutes and hours are optional). Negative values are allowed

/timeoutincrease <turn> <turninc> <value> <valuemult>

Every **<turn>** turns, add **<value>** to the timeout timer, then add **<turninc>** to **<turn>** and multiply **<value>** by **<valuemult>**. Use this command in concert with the option **/timeout**. Defaults are **0 0 0 1**.

/ignore [type=]<pattern>

The given pattern will be added to your ignore list. You will not receive any messages from users matching this pattern. The type may be either **user**, **host**, or **ip**. The default type (if omitted) is to match against the username. The pattern supports unix glob style wildcards, i.e., ***** matches zero or more character, **?** exactly one character, **[abc]** exactly one of a, b, or c, etc. To access your current ignore list, issue **/list ignore**.

/unignore <range>

The ignore list entries in the given range will be removed. You will be able to receive messages from the respective users. The range argument may be a single number or a pair of numbers separated by a dash **-**. If the first number is omitted, it is assumed to be 1. If the last is omitted, it is assumed to be the last valid ignore list index. To access your current ignore list, issue **/list ignore**.

/playercolor <player-name> <color>

This command sets the color of a specific player, overriding any color assigned according to the **plrcolormode** setting. The color is defined using hexadecimal notation (hex) for the combination of Red, Green, and Blue color components (RGB), similarly to HTML. For each component, the lowest (darkest) value is **0** (in hex: **00**), and the highest value is 255 (in hex: **FF**). The color definition is simply the three hex values concatenated together (**RRGGBB**). For example, the following command sets Caesar to pure red: **playercolor Caesar ff0000**. Before the game starts, this command can only be used if the **plrcolormode** setting is set to **PLR_SET**. A player's color can be unset again by specifying **reset**. Once the game has started and colors have been assigned, this command changes the player color in any mode; **reset** cannot be used. To list the player colors, use **/list colors**.

/playernation <player-name> [nation] [is-male] [leader] [style]

This command sets the nation, leader name, style, and gender of a specific player. The string "random" can be used to select a random nation. The gender parameter should be 1 for male, **0** for female. Omitting any of the player settings will reset the player to defaults. This command may not be used once the game has started.

/endgame

End the game immediately in a draw.

/surrender

This tells everyone else that you concede the game, and if all but one player (or one team) have conceded the game in this way then the game ends.

/remove <player-name>

This *completely* removes a player from the game, including all cities and units etc. Use with care!

/save <file-name>

Save the current game to file <file-name>. If no file-name argument is given saves to <auto-save name prefix><year>m.sav[.gz]. To reload a savegame created by /save, start the server with the command-line argument: --file <filename> or -f <filename> and use the /start command once players have reconnected.

/scensave <file-name>

Save the current game to file <file-name> as a scenario. If no file-name argument is given saves to <auto-save name prefix><year>m.sav[.gz]. To reload a savegame created by /scensave, start the server with the command-line argument: --file <filename> or -f <filename> and use the /start command once players have reconnected.

/load <file-name>

Load a game from <file-name>. Any current data including players, rulesets and server options are lost.

/read <file-name>

Process server commands from file.

/write <file-name>

Write current settings as server commands to file.

/reset game|ruleset|script|default

Reset all settings if it is possible. The following levels are supported:

- game: using the values defined at the game start.
- ruleset: using the values defined in the ruleset.
- script: using default values and rereading the start script.
- default: using default values.

/default <option name>

Reset the option to its default value. If the default ever changes in a future version, the option's value will follow that change.

/lua cmd <script line>

Evaluate a line of Freeciv21 script or a Freeciv script file in the current game. Variations are:

- lua cmd <script line>
- lua unsafe-cmd <script line>
- lua file <script file>
- lua unsafe-file <script file>

The unsafe prefix runs the script in an instance separate from the ruleset. This instance does not restrict access to Lua functions that can be used to hack the computer running the Freeciv21 server. Access to it is therefore limited to the console and connections with cmdlevel hack.

/kick <user>

The connection given by the user argument will be cut from the server and not allowed to reconnect. The time the user would not be able to reconnect is controlled by the kicktime setting.

/delegate to <username>

Delegation allows a user to nominate another user who can temporarily take over control of their player while they are away. Variations are:

- `/delegate to <username>`: Allow <username> to `delegate take` your player.
- `/delegate cancel`: Nominated user can no longer take your player.
- `/delegate take <player-name>`: Take control of a player who has been delegated to you. Behaves like `/take`, except that the `/allowtake` restrictions are not enforced.
- `/delegate restore`: Relinquish control of a delegated player (opposite of `/delegate take`) and restore your previous view, if any. This also happens automatically if the player's owner reconnects.
- `/delegate show`: Show who control of your player is currently delegated to, if anyone.

The `[player-name]` argument can only be used by connections with `cmdlevel admin` or above to force the corresponding change of the delegation status.

/aicmd <player> <command>

Execute a command in the context of the [AI](#) for the given player.

/fcdB lua <script>

The argument `reload` causes the database script file to be re-read after a change, while the argument `lua` evaluates a line of Lua script in the context of the Lua instance for the database.

/mapping define <mapdef>

Create image files of the world/player map. Variations are:

- `mapping define <mapdef>`
- `mapping show <id>|all`
- `mapping create <id>|all`
- `mapping delete <id>|all`
- `mapping colortest`

This command controls the creation of map images. Supported arguments:

- `define <mapdef>`: define a map image; returns numeric <id>.
- `show <id>|all`: list map image definitions or show a specific one.
- `create <id>|all`: manually save image(s) for current map state.
- `delete <id>|all`: delete map image definition(s).
- `colortest`: create test image(s) showing all colors.

Multiple definitions can be active at once. A definition <mapdef> consists of colon-separated options:

Table 3.1: mapdef options

Option	(Default)	Description
format=<[tool]]format>	(ppm ppm)	file format
show=<show>	(all)	which players to show
plrname=<name>		player name
plrid=<id>		numeric player id
plrbv=<bit vector>		see example; first char = id 0
turns=<turns>	(1)	save image each <turns> turns (0=no autosave, save with create)
zoom=<zoom>	(2)	magnification factor (1-5)
map=<map>	(bcku)	which map layers to draw

<[tool]]format> = use image format <format>, optionally specifying toolkit <tool>. The following toolkits and formats are compiled in:

- 0: ppm

<show> determines which players are represented and how many images are saved by this definition:

- none: no players, only terrain.
- each: one image per player.
- human: one image per human player.
- all: all players on a single image.
- plrname: just the player named with plrname.
- plrid: just the player specified with plrid.
- plrbv: one image per player in plrbv.

<map> can contain one or more of the following layers:

- a: show area within borders of specified players.
- b: show borders of specified players.
- c: show cities of specified players.
- f: show fog of war (single-player images only).
- k: show only player knowledge (single-player images only).
- t: full display of terrain types.
- u: show units of specified players.

Examples of <mapdef>:

- zoom=1:map=tcub:show=all:format=ppm|ppm
- zoom=2:map=tcub:show=each:format=png
- zoom=1:map=tcub:show=plrname:plrname=Otto:format=gif
- zoom=3:map=cu:show=plrbv:plrbv=010011:format=jpg
- zoom=1:map=t:show=none:format=magick|jpg

/rfcstyle

Switch server output between ‘RFC-style’ and normal style.

/serverid

Simply returns the id of the server.

3.2.4 Server Options

Server Options are a collection of flags that can be set from the server command-line. The *Game Manual* refers to a *menu setting* called *Game Options*. In reality, the dialog box allows a user to use a graphical interface to set specific gameplay options. This page provides a detailed description of all of the options that a game admin can set from the server command line.

Refer to *Server Settings File* for information on how to place all the options in a settings file to call from the *Server Command Line Options*.

To place a setting value on any of these settings use the `/set <option-name> <value> server command`.

Note: The default value in the “Value (min, max)” section below is based on the Classic ruleset. The default value is often set by the ruleset, but can be overridden before the game starts.

aifill

Default Value (Min, Max): 5 (0, 500)

Description: If set to a positive value, then AI players will be automatically created or removed to keep the total number of players at this amount. As more players join, these AI players will be replaced. When set to zero, all AI players will be removed.

airliftingstyle

Default Value: empty value / not set

Description: This setting affects airlifting units between cities. It can be a set of the following values:

- FROM_ALLIES: Allows units to be airlifted from allied cities.
- TO_ALLIES: Allows units to be airlifted to allied cities.
- SRC_UNLIMITED: Unlimited units from source city. **Note** that airlifting from a city does not reduce the airlifted counter, but still needs airlift capacity of at least 1.
- DEST_UNLIMITED: Unlimited units to destination city. **Note** that airlifting to a city does not reduce the airlifted counter, and does not need any airlift capacity.

allowtake

Default Value: “HAhadOo”

Description: This should be a string of characters, each of which specifies a type or status of a civilization (player). Clients will only be permitted to take or observe those players which match one of the specified letters. This only affects future uses of the `/take` or `/observe` commands. It is not retroactive. The characters and their meanings are:

- o,O = Global observer
- b = Barbarian players
- d = Dead players
- a,A = AI players
- h,H = Human players

The first description on this list which matches a player is the one which applies. Thus **d** does not include dead barbarians, **a** does not include dead **AI** players, and so on. Upper case letters apply before the game has started, lower case letters afterwards.

Each character above may be followed by one of the following numbers to allow or restrict the manner of connection:

- (none) = Controller allowed, observers allowed, can displace connections. (Displacing a connection means that you may take over a player, even when another user already controls that player.)
- 1 = Controller allowed, observers allowed, cannot displace connections.
- 2 = Controller allowed, no observers allowed, can displace connections.
- 3 = Controller allowed, no observers allowed, cannot displace connections.
- 4 = No controller allowed, observers allowed

alltemperate

Default Value: disabled

Description: If this setting is enabled, the temperature will be equivalent everywhere on the map. As a result, the poles will not be generated.

animals

Default Value (Min, Max): 20 (0, 500)

Description: Number of animals initially created on terrains defined for them in the ruleset (if the ruleset supports it). The server variable's scale is animals per thousand tiles.

aqueductloss

Default Value (Min, Max): 0 (0, 100)

Description: If a city would expand, but it cannot because it lacks some prerequisite (traditionally an Aqueduct or Sewer System), this is the base percentage of its foodbox that is lost each turn. The penalty may be reduced by buildings or other circumstances, depending on the ruleset.

autoattack

Default Value: disabled

Description: If set to on, units with moves left will automatically consider attacking enemy units that move adjacent to them.

autosaves

Default Value: TURN | GAMEOVER | QUITIDLE | INTERRUPT

Description: This setting controls which autosave types get generated:

- TURN: Save when turn begins, once every `save turns` turns.
- GAMEOVER: Final save when game ends.
- QUITIDLE: Save before server restarts due to lack of players.
- INTERRUPT: Save when server quits due to interrupt.
- TIMER: Save every `save frequency` minutes.

autotoggle

Default Value: disabled

Description: If enabled, **AI** status is turned off when a player connects, and on when a player disconnects.

barbarians

Default Value: NORMAL

Description: This setting controls how frequently the Barbarians appear in the game. See also the `onsetbarbs` setting. Possible values:

- `DISABLED`: No barbarians.
- `HUTS_ONLY`: Only in huts.
- `NORMAL`: Normal rate of appearance.
- `FREQUENT`: Frequent barbarian uprising.
- `HORDES`: Raging hordes.

borders

Default Value: `ENABLED`

Description: If this is not disabled, then any land tiles around a city or border-claiming extra (like the classic ruleset's Fortress base) will be owned by that nation. Possible values:

- `SEE_INSIDE`: See everything inside borders.
- `EXPAND`: Borders expand to unknown, revealing tiles.
- `ENABLED`: Will, in some rulesets, grant the same visibility if certain conditions are met.
- `DISABLED`: Disabled
- `ENABLED`: Enabled

caravan_bonus_style

Default Value: `CLASSIC`

Description: The formula for the bonus when a Caravan enters a city. Possible values:

- `CLASSIC`: Bonuses are proportional to distance and trade of source and destination with multipliers for overseas and international destinations.
- `LOGARITHMIC`: Bonuses are proportional to $\log^2(\text{distance} + \text{trade})$.
- `LINEAR`: Bonuses are similar to `CLASSIC`, but (like `LOGARITHMIC`) use the max trade of the city rather than current.
- `DISTANCE`: Bonuses are proportional only to distance.

citymindist

Default Value (Min, Max): 2 (1, 11)

Description: Minimum distance between cities. When a player attempts to found a new city, it is prevented if the distance from any existing city is less than this setting. For example, when this setting is 3, there must be at least two clear tiles in any direction between all existing cities and the new city site. A value of 1 removes any such restriction on city placement.

citynames

Default Value: `PLAYER_UNIQUE`

Description: Allowed city names. Possible values:

- `NO_RESTRICTIONS`: No restrictions. Players can have multiple cities with the same names.
- `PLAYER_UNIQUE`: Unique to a player. One player cannot have multiple cities with the same name.
- `GLOBAL_UNIQUE`: Globally unique. All cities in a game have to have different names.
- `NO_STEALING`: No city name stealing. Like “Globally unique”, but a player is not allowed to use a default city name of another nation unless it is a default for their nation also.

civilwarsize

Default Value (Min, Max): 10 (2, 1000)

Description: Minimum number of cities for civil war. A civil war is triggered when a player has at least this many cities and the player's capital is captured. If this option is set to the maximum value, civil wars are turned off altogether.

compresstype

Default Value: XZ

Description: Compression library to use for savegames. Possible values:

- PLAIN: No compression.
- LIBZ: Using zlib (gzip format).
- BZIP2: Using bzip2 (deprecated).
- XZ: Using xz.

conquercost

Default Value (Min, Max): 0 (0, 100)

Description: Penalty when getting tech from conquering. For each technology you gain by conquering an enemy city, you lose research points equal to this percentage of the cost to research a new technology. If this is non-zero, you can end up with negative research points.

contactturns

Default Value (Min, Max): 20 (0, 100)

Description: Turns until player contact is lost. Players may meet for diplomacy this number of turns after their units have last met, even when they do not have an embassy. If set to zero, then players cannot meet unless they have an embassy.

demography

Default Value: "NASRLPEMOCqrb"

Description: What is shown in the Demographics report. This should be a string of characters, each of which specifies the inclusion of a line of information in the Demographics report. The characters and their meanings are:

- s = include Score
- z = include League Score
- N = include Population
- n = include Population in Citizen Units
- c = include Cities
- i = include Improvements
- w = include Wonders
- A = include Land Area
- S = include Settled Area
- L = include Literacy
- a = include Agriculture
- P = include Production
- E = include Economics

- g = include Gold Income
- R = include Research Speed
- M = include Military Service
- m = include Military Units
- u = include Built Units
- k = include Killed Units
- l = include Lost Units
- O = include Pollution
- C = include Culture

Additionally, the following characters control whether or not certain columns are displayed in the report:

- q = display “quantity” column
- r = display “rank” column
- b = display “best nation” column

The order of characters is not significant, but their capitalization is.

diplbulbcost

Default Value (Min, Max): 0 (0, 100)

Description: Penalty when getting tech from treaty. For each technology you gain from a diplomatic treaty, you lose research points equal to this percentage of the cost to research a new technology. If this is non-zero, you can end up with negative research points.

diplchance

Default Value (Min, Max): 80 (40, 100)

Description: Base chance for diplomats and spies to succeed. The base chance of a Spy returning from a successful mission and the base chance of success for Diplomat and Spy units.

diplgoldcost

Default Value (Min, Max): 0 (0, 100)

Description: Penalty when getting gold from treaty. When transferring gold in diplomatic treaties, this percentage of the agreed sum is lost to both parties. It is deducted from the donor, but not received by the recipient.

diplomacy

Default Value: ALL

Description: Ability to do diplomacy with other players. This setting controls the ability to do diplomacy with other players. Possible values:

- ALL: Enabled for everyone.
- HUMAN: Only allowed between human players.
- AI: Only allowed between AI players.
- NOAI: Only allowed when human involved.
- NOMIXED: Only allowed between two humans, or two AI players.
- TEAM: Restricted to teams.
- DISABLED: Disabled for everyone.

disasters

Default Value (Min, Max): 10 (0, 1000)

Description: Frequency of disasters. Affects how often random disasters happen to cities, if any are defined by the ruleset. The relative frequency of disaster types is set by the ruleset. Zero prevents any random disasters from occurring and higher values create more opportunities for disasters to occur.

dispersion

Default Value (Min, Max): 0 (0, 10)

Description: Area where initial units are located. This is the radius within which the initial units are dispersed at game start.

ec_chat

Default Value: enabled

Description: Save chat messages in the event cache. If turned on, chat messages will be saved in the event cache.

ec_info

Default Value: disabled

Description: Print turn and time for each cached event. If turned on, all cached events will be marked by the turn and time of the event like (T2 - 15:29:52).

ec_max_size

Default Value (Min, Max): 256 (10, 20000)

Description: Size of the event cache. This defines the maximal number of events in the event cache.

ec_turns

Default Value (Min, Max): 1 (0, 32768)

Description: Event cache for this number of turns. Event messages are saved for this number of turns. A value of 0 deactivates the event cache.

endspaceship

Default Value: enabled

Description: Should the game end if the spaceship arrives? If this option is turned on, the game will end with the arrival of a spaceship at Alpha Centauri.

endturn

Default Value (Min, Max): 5000 (1, 32767)

Description: Turn the game ends. The game will end at the end of the given turn.

first_timeout

Default Value (Min, Max): -1 (-1, 8639999)

Description: First turn timeout. If greater than 0, T1 will last for `first_timeout` seconds. If set to 0, T1 will not have a timeout. If set to -1, the special treatment of T1 will be disabled. See also `timeout`.

fixedlength

Default Value: disabled

Description: Fixed-length turns play mode. If this is turned on the game turn will not advance until the timeout has expired, even after all players have clicked on *Turn Done*.

flatpoles

Default Value (Min, Max): 100 (0, 100)

Description: How much the land at the poles is flattened. Controls how much the height of the poles is flattened during map generation, preventing a diversity of land terrain there. 0 is no flattening, 100 is maximum flattening. Only affects the RANDOM and FRACTAL map generators.

foggedborders**Default Value:** disabled

Description: Whether fog of war applies to border changes. If this setting is enabled, players will not be able to see changes in tile ownership if they do not have direct sight of the affected tiles. Otherwise, players can see any or all changes to borders as long as they have previously seen the tiles.

fogofwar**Default Value:** enabled

Description: Whether to enable fog of war. If this is enabled, only those units and cities within the vision range of your own units and cities will be revealed to you. You will not see new cities or terrain changes in tiles not observed.

foodbox**Default Value (Min, Max):** 100 (1, 10000)

Description: Food required for a city to grow. This is the base amount of food required to grow a city. This value is multiplied by another factor that comes from the ruleset and is dependent on the size of the city.

freecost**Default Value (Min, Max):** 0 (0, 100)

Description: Penalty when getting a free tech. For each technology you gain “for free” (other than covered by `dipcost` or `conquercost`: for instance, from huts or from Great Library effects), you lose research points equal to this percentage of the cost to research a new technology. If this is non-zero, you can end up with negative research points.

fulltradesize**Default Value (Min, Max):** 1 (1, 50)

Description: Minimum city size to get full trade. There is a trade penalty in all cities smaller than this value. The penalty is 100% (no trade at all) for sizes up to `notradesize`, and decreases gradually to 0% (no penalty except the normal corruption) for `size` = `fulltradesize`. See also `notradesize`.

gameseed**Default Value (Min, Max):** 0 (0, 2147483647)

Description: Game random seed. For zero (the default) a seed will be chosen based on the current time.

generator**Default Value:** RANDOM

Description: Method used to generate map. Specifies the algorithm used to generate the map. If the default value of the `startpos` option is used, then the chosen generator chooses an appropriate `startpos` setting. Otherwise, the generated map tries to accommodate the chosen `startpos` setting.

- **SCENARIO:** Scenario map. Indicates a pre-generated map. By default, if the scenario does not specify start positions, they will be allocated depending on the size of continents.
- **RANDOM:** Fully random height. Generates maps with a number of equally spaced, relatively small islands. By default, start positions are allocated depending on continent size.
- **FRACTAL:** Pseudo-fractal height. Generates Earthlike worlds with one or more large continents and a scattering of smaller islands. By default, players are all placed on a single continent.
- **ISLAND:** Island-based. Generates *fair* maps with a number of similarly-sized and -shaped islands, each with approximately the same ratios of terrain types. By default, each player gets their own island.
- **FAIR:** Fair islands. Generates the exact copy of the same island for every player or every team.

- **FRACTURE:** Fracture map. Generates maps from a fracture pattern. Tends to place hills and mountains along the edges of the continents. If the requested generator is incompatible with other server settings, the server may fall back to another generator.

globalwarming

Default Value: enabled

Description: Global warming. If turned off, global warming will not occur as a result of pollution. This setting does not affect pollution.

globalwarming_percent

Default Value (Min, Max): 100 (1, 10000)

Description: Global warming percent. This is a multiplier for the rate of accumulation of global warming.

gold

Default Value (Min, Max): 50 (0, 50000)

Description: Starting gold per player. At the beginning of the game, each player is given this much gold.

happyborders

Default Value: NATIONAL

Description: Units inside borders cause no unhappiness. If this is set, units will not cause unhappiness when inside your borders, or even allies borders, depending on value. Possible values:

- **DISABLED:** Borders are not helping.
- **NATIONAL:** Happy within own borders.
- **ALLIED:** Happy within allied borders.

homecaughtunits

Default Value: enabled

Description: Give caught units a homecity. If unset, caught units will have no homecity and will be subject to the `killunhomed` option.

huts

Default Value (Min, Max): 15 (0, 500)

Description: Amount of huts (bonus extras). Huts are tile extras that usually may be investigated by units. The server variable's scale is huts per thousand tiles.

incite_gold_capt_chance

Default Value (Min, Max): 0 (0, 100)

Description: Probability of gold capture during inciting revolt. When the unit trying to incite a revolt is eliminated and loses its gold, there is chance that this gold would be captured by city defender. Ruleset defined transfer tax would be applied, though. This setting is irrelevant, if `incite_gold_loss_chance` is zero.

incite_gold_loss_chance

Default Value (Min, Max): 0 (0, 100)

Description: Probability of gold loss during inciting revolt. When the unit trying to incite a revolt is eliminated, half of the gold (or quarter, if unit was caught), prepared to bribe citizens, can be lost or captured by enemy.

kicktime

Default Value (Min, Max): 1800 (0, 86400)

Description: Time before a kicked user can reconnect. Gives the time in seconds before a user kicked using the `/kick` server [command](#) may reconnect. Changing this setting will affect users kicked in the past.

killcitizen

Default Value: enabled

Description: Reduce city population after attack. This flag indicates whether a city's population is reduced after a successful attack by an enemy unit. If this is disabled, population is never reduced. Even when this is enabled, only some units may kill citizens.

killstack

Default Value: enabled

Description: Do all units in tile die with defender? If this is enabled, each time a defender unit loses in combat, and is not inside a city or suitable base, all units on the same tile are destroyed along with the defender. If this is disabled, only the defender unit is destroyed.

killunhomed

Default Value (Min, Max): 0 (0, 100)

Description: Slowly kill units without home cities (e.g., starting units). If greater than 0, then every unit without a home city will lose *HP* each turn. The number of hitpoints lost is given by `killunhomed` percent of the HP of the unit type. At least one HP is lost every turn until the death of the unit.

landmass

Default Value (Min, Max): 30 (15, 85)

Description: Percentage of the map that is land. This setting gives the approximate percentage of the map that will be made into land.

mapseed

Default Value (Min, Max): 0 (0, 2147483647)

Description: Map generation random seed. The same seed will always produce the same map. For zero (the default) a seed will be chosen based on the time to give a random map.

mapsize

Default Value: FULLSIZE

Description: Map size definition. Chooses the method used to define the map size. Other options specify the parameters for each method.

- FULLSIZE: Number of tiles. Map area (option `size`).
- PLAYER: Tiles per player. Number of (land) tiles per player (option `tilesperplayer`).
- XYSIZE: Width and height. Map width and height in tiles (options `xsize` and `ysize`).

maxconnectionsperhost

Default Value (Min, Max): 4 (0, 1024)

Description: Maximum number of connections to the server per host. New connections from a given host will be rejected if the total number of connections from the very same host equals or exceeds this value. A value of 0 means that there is no limit, at least up to the maximum number of connections supported by the server.

maxplayers

Default Value (Min, Max): 500 (1, 500)

Description: Maximum number of players. The maximal number of human and *AI* players who can be in the game. When this number of players are connected in the pregame state, any new players who try to connect will be rejected. When playing a scenario which defines player start positions, this setting cannot be set to greater than the number of defined start positions.

metamessage

Default Value: ""

Description: Set user defined metaserver info line. If parameter is omitted, previously set `metamessage` will be removed. For most of the time user defined `metamessage` will be used instead of automatically generated messages, if it is available.

mgr_distance

Default Value (Min, Max): 0 (-5, 6)

Description: Maximum distance citizens may migrate. This setting controls how far citizens may look for a suitable migration destination when deciding which city to migrate to. The value is added to the candidate target city's radius and compared to the distance between the two cities. If the distance is lower or equal, migration is possible. So with a setting of 0, citizens will only consider migrating if their city's center is within the destination city's working radius. This setting has no effect unless migration is enabled by the `migration` setting.

mgr_foodneeded

Default Value: enabled

Description: Whether migration is limited by food. If this setting is enabled, citizens will not migrate to cities which would not have enough food to support them. This setting has no effect unless migration is enabled by the `migration` setting.

mgr_nationchance

Default Value (Min, Max): 50 (0, 100)

Description: Percent probability for migration within the same nation. This setting controls how likely it is for citizens to migrate between cities owned by the same player. Zero indicates migration will never occur, 100 means that migration will always occur if the citizens find a suitable destination. This setting has no effect unless migration is activated by the `migration` setting.

mgr_turninterval

Default Value (Min, Max): 5 (1, 100)

Description: Number of turns between migrations from a city. This setting controls the number of turns between migration checks for a given city. The interval is calculated from the founding turn of the city. So for example if this setting is 5, citizens will look for a suitable migration destination every five turns from the founding of their current city. Migration will never occur the same turn that a city is built. This setting has no effect unless migration is enabled by the `migration` setting.

mgr_worldchance

Default Value (Min, Max): 10 (0, 100)

Description: Percent probability for migration between foreign cities. This setting controls how likely it is for migration to occur between cities owned by different players. Zero indicates migration will never occur, 100 means that citizens will always migrate if they find a suitable destination. This setting has no effect if migration is not enabled by the `migration` setting.

migration

Default Value: disabled

Description: Whether to enable citizen migration. This is the master setting that controls whether citizen migration is active in the game. If enabled, citizens may automatically move from less desirable cities to more desirable ones. The *desirability* of a given city is calculated from a number of factors. In general larger cities with more income and improvements will be preferred. Citizens will never migrate out of the capital, or cause a wonder to be lost by disbanding a city.

minplayers

Default Value (Min, Max): 1 (0, 500)

Description: Minimum number of players. There must be at least this many players (connected human players) before the game can start.

multiresearch

Default Value: disabled

Description: Allow researching multiple technologies. Allows switching to any technology without wasting old research. Bulbs are never transferred to new technology. Techpenalty options are ineffective after enabling that

option.

nationset

Default Value: ""

Description: Set of nations to choose from. Controls the set of nations allowed in the game. The choices are defined by the ruleset. Only nations in the set selected here will be allowed in any circumstances, including new players and civil war. Small sets may thus limit the number of players in a game. If this is left blank, the ruleset's default nation set is used. See `/list nationsets` for possible choices for the currently loaded ruleset.

naturalcitynames

Default Value: enabled

Description: Whether to use natural city names. If enabled, the default city names will be determined based on the surrounding terrain. See *Nation Sets and Flags*.

netwait

Default Value (Min, Max): 4 (0, 20)

Description: Max seconds for network buffers to drain. The server will wait for up to the value of this parameter in seconds, for all client connection network buffers to unblock. Zero means the server will not wait at all.

notradesize

Default Value (Min, Max): 0 (0, 49)

Description: Maximum size of a city without trade. Cities do not produce any trade at all unless their size is larger than this amount. The produced trade increases gradually for cities larger than `notradesize` and smaller than `fulltradesize`. See also `fulltradesize`.

nuclearwinter

Default Value: enabled

Description: Nuclear winter. If turned off, nuclear winter will not occur as a result of nuclear fallout.

nuclearwinter_percent

Default Value (Min, Max): 100 (1, 10000)

Description: Nuclear winter percent. This is a multiplier for the rate of accumulation of nuclear winter.

occupychance

Default Value (Min, Max): 0 (0, 100)

Description: Chance of moving into tile after attack. If set to 0, combat is Civ1/2-style (when you attack, you remain in place). If set to 100, attacking units will always move into the tile they attacked when they win the combat (and no enemy units remain in the tile). If set to a value between 0 and 100, this will be used as the percent chance of "occupying" territory.

onsetbarbs

Default Value (Min, Max): 60 (1, 32767)

Description: Barbarian onset turn. Barbarians will not appear before this turn.

persistentready

Default Value: DISABLED

Description: When the Readiness of a player gets autotoggled off. In pre-game, usually when new players join or old ones leave, those who have already accepted game to start by toggling "Ready" get that autotoggled off in the changed situation. This setting can be used to make readiness more persistent. Possible values:

- DISABLED: Disabled.
- CONNECTED: As long as connected.

phasemode

Default Value: ALL

Description: Control of simultaneous player/team phases. This setting controls whether players may make moves at the same time during a turn. Change in setting takes effect next turn. Currently, at least to the end of this turn, mode is “All players move concurrently”. Possible values:

- ALL: All players move concurrently.
- PLAYER: All players alternate movement.
- TEAM: Team alternate movement.

pingtime

Default Value (Min, Max): 20 (1, 1800)

Description: Seconds between PINGs. The server will poll the clients with a PING request each time this period elapses.

pingtimeout

Default Value: 60 (60, 1800)

Description: Time to cut a client. If a client does not reply to a PING in this time the client is disconnected.

plrcolormode

Default Value: PLR_ORDER

Description: How to pick player colors. This setting determines how player colors are chosen. Player colors are used in the *Nations View*, for national borders on the map, and so on.

- PLR_ORDER: Per-player, in order. Colors are assigned to individual players in order from a list defined by the ruleset.
- PLR_RANDOM: Per-player, random. Colors are assigned to individual players randomly from the set defined by the ruleset.
- PLR_SET: Set manually. Colors can be set with the `/playercolor command` before the game starts. These are not restricted to the ruleset colors. Any players for which no color is set when the game starts get a random color from the ruleset.
- TEAM_ORDER: Per-team, in order. Colors are assigned to teams from the list in the ruleset. Every player on the same team gets the same color.
- NATION_ORDER: Per-nation, in order. If the ruleset defines a color for a player’s nation, the player takes that color. Any players whose nations don’t have associated colors get a random color from the list in the ruleset.

Regardless of this setting, individual player colors can be changed after the game starts with the `/playercolor` command.

rapturedelay

Default Value (Min, Max): 1 (1, 99)

Description: Number of turns between rapture effect. Sets the number of turns between rapture growth of a city. If set to n a city will grow after celebrating for $n + 1$ turns.

razechance

Default Value (Min, Max): 20 (0, 100)

Description: Chance for conquered building destruction. When a player conquers a city, each city improvement has this percentage chance to be destroyed.

restrictinfra

Default Value: disabled

Description: Restrict the use of the infrastructure for enemy units. If this option is enabled, the use of roads and rails will be restricted for enemy units.

revealmap

Default Value: empty value / not set

Description: Reveal the map. Possible values:

- **START:** Reveal map at game start. The initial state of the entire map will be known to all players from the start of the game, although it may still be fogged (depending on the `fogofwar` setting).
- **DEAD:** Unfog map for dead players. Dead players can see the entire map, if they are alone in their team.

revolen

Default Value (Min, Max): 5 (1, 20)

Description: Length of revolution. When changing governments, a period of anarchy will occur. Value of this setting, used the way `revolentype` setting dictates, defines the length of the anarchy.

revolentype

Default Value: RANDOM

Description: Way to determine revolution length. Which method is used in determining how long period of anarchy lasts when changing government. The actual value is set with `revolen` setting. The **quickening** methods depend on how many times any player has changed to this type of government before, so it becomes easier to establish a new system of government if it has been done before. Possible values:

- **FIXED:** Fixed to `revolen` turns.
- **RANDOM:** Randomly 1-‘revolen’ turns.
- **QUICKENING:** First time ‘revolen’, then always quicker.
- **RANDQUICK:** Random, max always quicker.

savefrequency

Default Value (Min, Max): 15 (2, 1440)

Description: Minutes per auto-save. How many minutes elapse between automatic game saves. Unlike other save types, this save is only meant as backup for computer memory, and it always uses the same name, older saves are not kept. This setting only has an effect when the `autosaves` setting includes `TIMER`.

savename

Default Value: “freeciv”

Description: Definition of the save file name. Within the string the following custom formats are allowed:

- `%R` = <reason>
- `%S` = <suffix>
- `%T` = <turn-number>
- `%Y` = <game-year>

Example: `freeciv-T%04T-Y%05Y-%R` returns `freeciv-T0100-Y000001-manual`

Be careful to use at least one of `%T` and `%Y`, else newer savegames will overwrite old ones. If none of the formats is used `-T%04T-Y%05Y-%R` is appended to the value of `savename` setting.

savepalace

Default Value: enabled

Description: Rebuild palace whenever capital is conquered. If this is turned on, when the capital is conquered the palace is automatically rebuilt for free in another randomly chosen city. This is significant because the technology

requirement for building a palace will be ignored. In some rulesets, buildings other than the palace are affected by this setting.

saveturns

Default Value (Min, Max): 1 (1, 200)

Description: Turns per auto-save. How many turns elapse between automatic game saves. This setting only has an effect when the autosaves setting includes `NEW TURN`.

sciencebox

Default Value (Min, Max): 100 (1, 10000)

Description: Technology cost multiplier percentage. This affects how quickly players can research new technology. All tech costs are multiplied by this amount (as a percentage). The base tech costs are determined by the ruleset or other game settings.

scorefile

Default Value: "freeciv-score.log"

Description: Name for the score log file. The default name for the score log file is `freeciv-score.log`.

scorelog

Default Value: disabled

Description: Whether to log player statistics. If this is turned on, player statistics are appended to the file defined by the option `scorefile` every turn. These statistics can be used to create power graphs after the game.

scoreloglevel

Default Value: ALL

Description: Scorelog level. Whether scores are logged for all players including *AI*'s, or only for human players. Possible values:

- ALL: All players.
- HUMANS: Human players only.

separatepoles

Default Value: enabled

Description: Whether the poles are separate continents. If this setting is disabled, the continents may attach to poles.

shieldbox

Default Value (Min, Max): 100 (1, 10000)

Description: Multiplier percentage for production costs. This affects how quickly units and buildings can be produced. The base costs are multiplied by this value (as a percentage).

singlepole

Default Value: disabled

Description: Whether there is just one pole generated. If this setting is enabled, only one side of the map will have a pole. This setting has no effect if the map wraps both directions.

size

Default Value (Min, Max): 4 (0, 2048)

Description: Map area (in thousands of tiles). This value is used to determine the map area. Size = 4 is a normal map of 4,000 tiles (default). Size = 20 is a huge map of 20,000 tiles. For this option to take effect, the "Map size definition" option (`mapsize`) must be set to "Number of tiles" (`FULLSIZE`).

spaceship_travel_time

Default Value (Min, Max): 100 (50, 1000)

Description: Percentage to multiply spaceship travel time by. This percentage is multiplied onto the time it will take for a spaceship to arrive at Alpha Centauri.

specials

Default Value (Min, Max): 250 (0, 1000)

Description: Amount of “special” resource tiles for the game. Special resources improve the basic terrain type they are on. The server variable’s scale is parts per thousand.

startcity

Default Value: disabled

Description: Whether player starts with a city. If this is set, the game will start with player’s first city already founded to starting location.

startpos

Default Value: DEFAULT

Description: The method used to choose where each player’s initial units start on the map. For scenarios which include pre-set start positions, this setting is ignored. Possible values:

- **DEFAULT:** Generator’s choice. The start position placement will depend on the map generator chosen. See the **generator** setting above.
- **SINGLE:** One player per continent. One player is placed on each of a set of continents of approximately equivalent value (if possible).
- **2or3:** Two or three players per continent. Similar to **SINGLE** except that two players will be placed on each continent, with three on the *best* continent if there is an odd number of players.
- **ALL:** All players on a single continent. All players will start on the *best* available continent.
- **VARIABLE:** Depending on size of continents. Players will be placed on the *best* available continents such that, as far as possible, the number of players on each continent is proportional to its value. If the server cannot satisfy the requested setting due to there being too many players for continents, it may fall back to one of the others. However, map generators try to create the right number of continents for the choice of this **startpos** setting and the number of players, so this is unlikely to occur.

startunits

Default Value: “ccwwx”

Description: List of players’ initial units. This should be a string of characters, each of which specifies a unit role. The first character must be native to at least one “Starter” terrain. The case-sensitive characters and their meanings are:

- **c** = City founder (eg., Settlers)
- **w** = Terrain worker (eg., Engineers)
- **x** = Explorer (eg., Explorer)
- **k** = Gameloss (eg., Leader)
- **s** = Diplomat (eg., Diplomat)
- **f** = Ferryboat (eg., Trireme)
- **d** = Ok defense unit (eg., Warriors)
- **D** = Good defense unit (eg., Phalanx)
- **a** = Fast attack unit (eg., Horsemen)
- **A** = Strong attack unit (eg., Catapult)

steepness

Default Value (Min, Max): 30 (0, 100)

Description: Amount of hills or mountains on the map. Small values give flat maps, while higher values give a steeper map with more hills and mountains.

team_pooled_research

Default Value: enabled

Description: If this setting is turned on, then the team mates will share the science research. Else, every player of the team will have to make its own.

teampacement

Default Value: CLOSEST

Description: Method used for placement of team mates. After start positions have been generated thanks to the `startpos` setting, this setting controls how the start positions will be assigned to the different players of the same team. Possible Values:

- **DISABLED:** The start positions will be randomly assigned to players, regardless of teams.
- **CLOSEST:** As close as possible. Players will be placed as close as possible, regardless of continents.
- **CONTINENT:** On the same continent. If possible, place all players of the same team onto the same island/continent.
- **HORIZONTAL:** Horizontal placement. Players of the same team will be placed horizontally.
- **VERTICAL:** Vertical placement. Players of the same team will be placed vertically.

techleak

Default Value (Min, Max): 100 (0, 300)

Description: The rate of the tech leakage. As other nations learn new technologies, other players that have not learned the same technology advance will have the number of bulbs reduced.

techlevel

Default Value (Min, Max): 0 (0, 100)

Description: Number of initial techs per player. At the beginning of the game, each player is given this many technologies. The technologies chosen are random for each player. Depending on the value of `tech_cost_style` in the ruleset, a big value for `techlevel` can make the next techs really expensive.

techlossforgiveness

Default Value (Min, Max): -1 (-1, 200)

Description: Research point (bulbs) debt threshold for losing a tech. When you have negative research points, and your shortfall is greater than this percentage of the cost of your current research, you forget a technology you already knew. The special value -1 prevents loss of technology regardless of research points.

techlossrestore

Default Value (Min, Max): 50 (-1, 100)

Description: Research points (bulbs) restored after losing a tech. When you lose a technology due to a negative research balance (see `techlossforgiveness`), this percentage of its research cost is credited to your research balance (this may not be sufficient to make it positive). The special value -1 means that your research balance is always restored to zero, regardless of your previous shortfall.

techlost_donor

Default Value (Min, Max): 0 (0, 100)

Description: Chance to lose a technology while giving it to another player. The chance that your civilization will lose a technology if you teach it to someone else by treaty, or if it is stolen from you.

techlost_recv

Default Value (Min, Max): 0 (0, 100)

Description: Chance to lose a technology while receiving it from another player. The chance that learning a technology by treaty or theft will fail.

techpenalty

Default Value (Min, Max): 100 (0, 100)

Description: Percentage penalty when changing technology research. If you change your current research technology, and you have positive research points (bulbs), you lose this percentage of those research points. This does not apply when you have just gained a technology this turn.

temperature

Default Value (Min, Max): 50 (0, 100)

Description: Average temperature of the planet. Small values will give a cold map, while larger values will give a warmer map.

- 100 means a very dry and hot planet with no polar arctic zones, only tropical and dry zones.
- 70 means a hot planet with little polar ice.
- 50 means a temperate planet with normal polar, cold, temperate, and tropical zones; a desert zone overlaps tropical and temperate zones.
- 30 means a cold planet with small tropical zones.
- 0 means a very cold planet with large polar zones and no tropics.

threaded_save

Default Value: disabled

Description: Whether to do saving in separate thread. If this is turned on, compressing and saving the actual file containing the game situation takes place in the background while game otherwise continues. This way users are not required to wait for the save to finish.

tilesperplayer

Default Value (Min, Max): 100 (1, 1000)

Description: Number of (land) tiles per player. This value is used to determine the map dimensions. It calculates the map size at game start based on the number of players and the value of the setting `landmass`. For this option to take effect, the “Map size definition” option (`mapsize`) must be set to “Tiles per player” (`PLAYER`).

timeaddenemymove

Default Value (Min, Max): 0 (0, 8639999)

Description: Timeout at least n seconds when enemy moved. Any time a unit moves while in sight of an enemy player, the remaining timeout is increased to this value. This setting helps with *RTS*.

timeout

Default Value (Min, Max): 0 (-1, 8639999)

Description: Maximum seconds per turn. If all players have not hit *Turn Done* before this time is up, then the turn ends automatically. Zero means there is no timeout. In servers compiled with debugging, a timeout of -1 sets the autogame test mode. Only connections with hack level access may set the timeout to fewer than 30 seconds. Use this with the command `timeoutincrease` to have a dynamic timer. The first turn is treated as a special case and is controlled by the `first_timeout` setting.

tinyisles

Default Value: disabled

Description: Presence of 1x1 islands. This setting controls whether the map generator is allowed to make islands of one only tile size.

topology

Default Value: Wrap East-West and Isometric (WRAPX|ISO)

Description: Freeciv21 maps are always two-dimensional. They may wrap at the north-south and east-west directions to form a flat map, a cylinder, or a torus (donut). Individual tiles may be rectangular or hexagonal, with either an overhead (“classic”) or isometric alignment. To play with a particular topology, clients will need a matching tileset.

Possible values (option can take any number of these):

- WRAPX: Wrap East-West
- WRAPY: Wrap North-South
- ISO: Isometric
- HEX: Hexagonal

trade_revenue_style

Default Value: CLASSIC

Description: The formula for the trade a city receives from a trade route. Possible values:

- CLASSIC: Revenues depend on distance and trade with multipliers for overseas and international routes.
- SIMPLE: Revenues are proportional to the average trade of the two cities.

trademindist

Default Value (Min, Max): 9 (1, 999)

Description: Minimum distance (tiles) for trade routes. In order for two cities in the same civilization to establish a trade route, they must be at least this far apart on the map. For square grids, the distance is calculated as *Manhattan distance*, that is, the sum of the displacements along the *x* and *y* directions. For hexagonal tiles, the distance is calculated as *Absolute distance*, that is, the sum of the absolute value between the *x* and *y* directions.

tradeworldrelpct

Default Value (Min, Max): 50 (0, 100)

Description: How largely trade distance is relative to world size. When determining trade between cities, the distance factor can be partly or fully relative to world size. This setting determines how big percentage of the bonus calculation is relative to world size, and how much only absolute distance matters.

trading_city

Default Value: enabled

Description: City trading via treaty. If turned off, trading cities in the diplomacy dialog is not allowed.

trading_gold

Default Value: enabled

Description: Gold trading via treaty. If turned off, trading gold in the diplomacy dialog is not allowed.

trading_tech

Default Value: enabled

Description: Technology trading via treaty. If turned off, trading technologies in the diplomacy dialog is not allowed.

traitdistribution

Default Value: FIXED

Description: *AI* trait distribution method. Possible values:

- FIXED: Fixed
- EVEN: Even

turnblock**Default Value:** enabled**Description:** Turn-blocking game play mode. If this is turned on, the game turn is not advanced until all players have finished their turn, including disconnected players.**unitwaittime****Default Value (Min, Max):** 0 (0, 8639999)**Description:** Minimum time between unit actions over turn change. This setting gives the minimum amount of time in seconds between unit moves and other significant actions (such as building cities) after a turn change occurs. For example, if this setting is set to 20 and a unit moves 5 seconds before the turn change, it will not be able to move or act in the next turn for at least 15 seconds. This value is limited to a maximum value of two-thirds of timeout. *unitwaittime* (*UWT*) is a tool to help reduce *RTS* around *TC* in Longturn games.**unitwaittime_extended****Default Value:** disabled**Description:** *unitwaittime* also applies to newly-built and captured/bribed units. If set, newly-built units are subject to *unitwaittime* so that the moment the city production was last touched counts as their last *action*. Also, getting captured/bribed counts as action for the victim.**unitwaittime_style****Default Value:** empty value/not set**Description:** This setting affects *unitwaittime* and effectively unused as it only has one option to set:

- **ACTIVITIES:** Units moved less than *unitwaittime* seconds from turn change will not complete activities such as pillaging and building roads during turn change, but during the next turn when their wait expires.

unreachableprotects**Default Value:** enabled**Description:** Does unreachable unit protect reachable ones. This option controls whether tiles with both unreachable and reachable units can be attacked. If disabled, any tile with reachable units can be attacked. If enabled, tiles with an unreachable unit in them cannot be attacked. Some units in some rulesets may override this, never protecting reachable units on their tile.**victories****Default Value:** SPACERACE|ALLIED**Description:** What kinds of victories are possible for the game. This setting controls how a game can be won. One can always win by conquering the entire planet, but other victory conditions can be enabled or disabled:

- **SPACERACE:** Spaceship is built and travels to Alpha Centauri.
- **ALLIED:** After defeating enemies, all remaining players are allied.
- **CULTURE:** Player meets ruleset defined cultural domination criteria.

wetness**Default Value (Min, Max):** 50 (0, 100)**Description:** Amount of water on the landmasses. Small values mean lots of dry, desert-like land. Higher values give a wetter map with more swamps, jungles, and rivers.**xsize****Default Value (Min, Max):** 64 (16, 128000)**Description:** Map width in tiles. Defines the map width. For this option to take effect, the “Map size definition” option (*mapsize*) must be set to “Width and height” (*XYSIZE*).**ysize****Default Value (Min, Max):** 64 (16, 128000)

Description: Map height in tiles. Defines the map height. For this option to take effect, the “Map size definition” option (mapsize) must be set to “Width and height” (XYSIZE).

3.2.5 Server Settings File

Freeciv21 servers can use a specially formatted plain text `.serv` file. This server settings file allows you to start a server with consistent settings. This way you can customize the way the server loads at startup instead of with the defaults from the default ruleset (Classic).

For some examples, refer to the [Longturn games repository](#). Every Longturn game has its own directory and within is a `.serv` file for that game. It is recommended to look at more recent games such as LT75 or later.

For more information on the varying commands you can place in a `.serv` file, you can access help via the server command prompt:

```
$ freeciv21-server  
  
...  
  
For introductory help, type 'help'.  
> help citymindist
```

To make use of the `.serv` file, you would start the server with the `-r` option, such as:

```
$ freeciv21-server -p 5000 -r mygame.serv
```

Tip: The Longturn community creates two `.serv` files for each game. The first (top-level) file is used to set all of the *game option parameters* and a second is used to define the players. The second file is loaded from the first file via the `/read <filename> server` command.

Tip: As noted on other pages, best practice is to use a Bash `.sh` script to standardize all the command-line parameters and loading of the settings file to ensure a consistent experience for your players.

For more information on setting up players, refer to [Advanced Player Setup](#).

3.2.6 Authentication and Database Support (fcdB)

The Freeciv21 server allows for the authentication of users, although by default it is not configured, and anyone can connect with any username.

In order to support authentication, the Freeciv21 server needs access to a database backend in which to store the credentials. To support different database backends, the database access code is written in Lua using luasql. In principle, luasql supports SQLite3, MySQL, and Postgres backends. However, the Freeciv21 server is only built with SQLite3 and the provided `database.lua` is also only programmed to support SQLite3.

As well as storing and retrieving usernames and passwords, the supplied database access script logs the time and IP address of each attempted login, although this information is not used by the Freeciv21 server itself.

To use the Freeciv21 database and authentication, the server must be installed properly, as it searches for `database.lua` in the `$HOME/.config/freeciv21` directory. The server cannot simply be run from a build directory if authentication is required.

Quick Setup: SQLite

The simplest setup is to use the SQLite3 backend, and this is probably the best option for new deployments. In this setup, the authentication data is stored in a simple file accessed directly by the Freeciv21 server. There is no need for a separate database server process.

To set this up, first create a database configuration file called something like `fc_auth.conf`, with the database key specifying where the database file is to live. It must be readable and writable by the Freeciv21 server. A good default location is the user's `$HOME` directory. Here is an example:

```
[fcdb]
backend="sqlite"
database="/home/[user]/freeciv21.sqlite"
```

Note: The username `[user]` should be changed to a real user.

For more information on the format of this file, see below. There are more settings available, but this file is entirely sufficient for a SQLite3 setup.

Now we need some software and Lua scripts installed on the server to support the SQLite3 database as well as storing passwords with MD5 hashes.

```
$ sudo apt install luarocks lua5.3 liblua5.3-dev libsqlite3-dev
$ eval $(luarocks path --bin --lua-version 5.3)
$ luarocks --local --lua-version 5.3 install md5
$ luarocks --local --lua-version 5.3 install luasql-sqlite3
```

Lastly, we need to download the `database.lua` script file.

```
$ mkdir -p $HOME/.config/freeciv21
$ cd $HOME/.config/freeciv21
$ wget https://raw.githubusercontent.com/longturn/freeciv21/master/lua/database.lua
$ cd ~
```

Now start the server with:

```
$ freeciv21-server --Database ~/fc_auth.conf --auth --Newusers
```

The first time you do this, you need to create the database file and its tables with the following server command:

```
/fcdb lua sqlite_createdb()
```

You should now see a `/home/[user]/freeciv21.sqlite` file as per the same path given in the `fc_auth.conf` file.

At this time you are ready to create some users by connecting with the client. Due to the `--Newusers` flag, when you connect with the client with a previously unknown username, the server will prompt for a password and save the new account to the database.

You may want to prepopulate the users table this way and then restart the server without `--Newusers` for the actual game, or you can run the game with `--Newusers`.

Advanced SQLite Usage

SQLite supports working with a temporary database in memory which is never written to disk. To do this, specify `database=":memory:"` in the configuration file. The database will last only for the lifetime of the `freeciv21-server` process. Its contents will be lost if the server quits or crashes. It is not saved in the saved game file, either. You will probably need the `--Newusers` option.

Command-line Options

The following server command-line options are relevant to authentication:

- `-D` or `--Database <conffile>`: Specifies a configuration file describing how to connect to the database. Without this, all authentication will fail.
- `-a` or `--auth`: Enable authentication. Without this, anyone will be able to connect without authentication, and `--Database` has no effect.
- `-G` or `--Guests`: Allow guests. These are usernames with names starting with “guest”. If enabled, any number of guests may connect without accounts in the database. If a guest name is already in use by a connection, a new guest name is generated. Once connected, guests have the same privileges as any other account. If this option is not specified, accounts are required to connect, and guest account names are forbidden.
- `-N` or `--Newusers`: Allow Freeciv21 clients to create new user accounts through the Freeciv21 protocol. Without this, only accounts which already exist in the database can connect. This might be desirable if you want users to register via a web front end, for instance.

Lua script database.lua

This script is responsible for checking usernames, fetching passwords, and saving new users (if `--Newusers` is enabled). It encapsulates access to the database backend, and hence the details of the table layout.

The script lives in `lua/database.lua` in the source tree. Instructions above place the file into `$HOME/.config/freeciv21`.

The supplied version supports basic authentication against a SQLite3 database. It supports configuration as shown in the following example:

```
[fcdb]
backend="sqlite"
host="localhost"
user="Freeciv21"
port="3306"
password="s3krit"
database="Freeciv21"
table_user="auth"
table_log="loginlog"
```

If that is sufficient for you, it is not necessary to read on. Freeciv21 expects the following lua functions to be defined in `database.lua`:

- Try to load data for an existing user.
- Return `TRUE` if the user exists, `FALSE` otherwise in function `user_load(conn)`.
- Save a new user to the database in function `user_save(conn)`.
- Log the connection attempt (success is boolean) in function `user_log(conn, success)`.

- Test and initialise the database connection in function `database_init()`.
- Free the database connection in function `database_free()`.

Where `conn` is an object representing the connection to the client which requests access.

The return status of all of these functions should be one of:

```
fcdb.status.ERROR
fcdb.status.TRUE
fcdb.status.FALSE
```

indicating an error, a positive result, or a negative result. The following lua functions are provided by Freeciv21:

- Return the client-specified username in function `auth.get_username(conn)`.
- Return the client IP address (string) in function `auth.get_ipaddr(conn)`.
- Tell the server (the MD5 hash of) the correct password to check against.
- For this connection (usually to be called by function `user_load()`).
- Returns whether this succeeded in function `auth.set_password(conn, password)`.
- Return (the MD5 hash of) the password for this connection (as specified by the client in in function `user_save()`, or as previously set by the function `set_password()`.
- The function `auth.get_password(conn)` runs.
- Return a value from the --Database configuration file in function `fcdb.option(type)`.

`type` selects one of the entries in the configuration file by name (for instance the function `fcdb.option("backend")`).

Freeciv21 also provides some of the same Lua functions that ruleset scripts get: `log.*()`, `_()`, etc., but the script is executing in a separate context from ruleset scripts, and does not have access to signals, game data, etc.

3.3 Advanced Game Topics

This section contains pages of documentation that cover a collection of advanced game topics. If you are looking for more general or generic gameplay information, it would be best to start with the [Frequently Asked Questions \(FAQ\)](#), or [How to Play Freeciv21](#).

3.3.1 Advanced Player Setup

In Freeciv21, there are many ways to setup a game with varying types of players. Games can be single-player in that there is only one human against a collection of *AI*. Games can be setup to be all human, such as Longturn games. Lastly, you can do a combination of the two: multiple humans versus multiple *AI*. Players can play *FFA* or in pre-allied teams as well.

More details are available in [How to Play Freeciv21](#), and in the in-game help. The [Game Manual](#) has instructions on how you can manipulate players in the *pre-game interface*.

The sections below use the server command line features, which is considered an advanced operation.

Single Player

In most cases when you start a single-player game you can change the number of players and their difficulty, directly through the *pre-game interface*. However, you may want to do something more complex.

When playing on a remote server, you typically do all the player manipulation by hand. Start by selecting the ruleset you wish to play and then change the `aifill` setting to the number of *AI* players you wish to play against.

```
$ freeciv21-server --ruleset royale
...
> /set aifill 30
> /list
```

Note: The default number of players is often ruleset defined, so selecting a different ruleset can change the pre-configured set of `aifill` players. Have a look at the server output at startup to see what the value of `aifill` was set to.

Difficulty levels are set with the `/cheating`, `/hard`, `/normal`, `/easy`, `/novice`, and `/handicapped` commands.

You may also create *AI* players individually. For instance, to create one hard and one easy *AI* player, enter:

```
> /create ai1
> /hard ai1
> /create ai2
> /easy ai2
> /list
```

Human vs Human Players

The steps to create a human only game is similar to the previous section. Start the server with the ruleset you wish to play and then remove all of the *AI* players. Here is a quick recipe:

```
$ freeciv21-server --ruleset royale
...
> /set aifill 0
> /set nationset all
> /create [player username]
> /aitoggle [player username]
> /playernation [player username] [Nation] [0/1]
> /list
```

Note: The value of `player username` can be anything, but is often the logon ID of the player that will play, is case sensitive, and it sets the leader name. The value of `Nation` is one of the available nations, also case sensitive. Notice the command to enable all `nationset`. The bit value of `0/1` is 1 for Male and 0 for Female.

The human player will right-click their logon ID in the *pre-game interface* and select *Take this player* at game start. This is a one-time procedure.

Tip: You can also create a plain text file named `players.serv`. Inside the file you add all of the user creation commands into one file. Then include a `read players.serv` command line at the bottom of the *Server Settings File*.

Teams of AI or Human Players

Following from the section above, you can easily set the team for a player with the `/team` command, like this:

```
> /set aifill 2
> /team AI*2 1
> /team AI*3 1
> /team human1 2
> /team human2 2
> /list
```

You will now have a two team game of two *AI* players against two human players.

3.3.2 Launching the Game with fc21 Links

The Freeciv21 Server accepts connection requests with a `fc21://` protocol handler on all platforms. This makes it possible to include login links on the Longturn.net webserver, or for games self-hosted by players.

Here is a sample from a command line:

```
$ ./freeciv21-client "fc21://[username]:[password]@[server]:[port]" -a -t amplio2
```

In this example, a player is launching the client from a command line and including the URI as the primary input. By passing `[username]`, `[password]`, `[server]`, and `[port]` this command provides all the details needed to connect to a game. The `-a` enables auto-connect and `-t amplio2` tells the client to load the Amplio2 tileset.

Alternately, HTML can be used in a webpage. For example:

```
<a href="fc21://[username]:[password]@[server]:[port]">Game Name</a>
```

Notice the same 4 parameters. The `[password]` parameter can be omitted. If so, the server will prompt the user for credentials before connecting.

3.3.3 Controlling the Map Generator

The generated map and player placement for a game can be manipulated by changing varying *Server Options*. This page aims to give the reader some recipes or options on how to manipulate the varying options to create the map you want to play. It is assumed that the game master is using a *Server Settings File*.

The server options that impact what kind of map the generator creates are:

- alltemperate
- flatpoles
- generator
- huts
- landmass
- mapseed

- `mapsize`
- `revealmap`
- `separatepoles`
- `singlepole`
- `size`
- `specials`
- `startpos`
- `steepness`
- `teamplacement`
- `temperature`
- `tilesperplayer`
- `tinyisles`
- `topology`
- `wetness`
- `xsize`
- `ysize`

Some server options are more impacting than others to the makeup of the map. Let us start with talking about the options that are least impacting to the overall map.

These options change the makeup of tiles and not the map itself:

- `alltemperate` – When enabled, all tiles have a similar temperature. Players will all have similar tiles to work. Set to disabled if you want to control with the `temperature` and `wetness` options.
- `huts` – How many huts you want on the map. Longturn games typically have 0 huts.
- `specials` – How many specials (tile resources) you want on the map.
- `steepness` – Bigger numbers produce hilly and mountainous terrain, lower numbers do not.
- `temperature` – You can create a hot or cold world with this option.
- `tinyisles` – Set to enabled if you want small single-tile (1x1) islands on the map.
- `wetness` – This option allows you to create a dry or wet map. Lower numbers are dry, higher numbers are wet.
- `revealmap` – This setting is used to reveal the whole map at game start. Good to use for testing, but not for real games.

Poles and Base Topology

The concept of poles and overall topology is important in how you want to create your world map. There are some options that when toggled will create a pole at the top and bottom, no poles whatsoever, or poles in the map that you can work and navigate around.

The settings that define the poles and base topology are:

- `flatpoles` – A low setting (0) will give poles with water and a high (100) will fill in with mostly glacier tiles on the poles.
- `separatepoles` – When enabled will break poles up and when disabled will keep them together.

- `singlepole` – When enabled will allow for a single pole and when disabled will give two poles.
- `topology` – Sets the wrap (X, Y), the orientation (isometric or overhead) and the tile type (squares vs hexes).

Recipe: Blocking poles on the map

Set topology `"WRAPX|ISO"`, `separatepoles` enabled, and `singlepole` disabled. This gives you a map with a blocking (cannot navigate around it) pole at the top and bottom of the map. You navigate East and West. If you set `singlepole` enabled here, you will get a really thick blocking border.

You can create blocking poles on the left and right by changing `WRAPX` to `WRAPY`. You would then navigate North and South.

Lastly, you can create a map that is like a game board. If you remove both `WRAPX` and `WRAPY` on `topology`, you will get regular poles at the top and bottom and “black” on the left and right. You can only navigate inside the box.

Recipe: A set of navigable poles on the map

Set topology `"WRAPX|WRAPY|ISO"`, `separatepoles` enabled, and `singlepole` disabled. This gives you two poles on the map, which looks and acts like Earth where you can navigate around them.

If you set `singlepole` enabled instead of disabled you will get a single pole on the map instead of two.

Lastly, if you set `flatpoles 0` you will get navigable poles with passages of water inside of them.

Recipe: No poles on the map

Set topology `"WRAPX|WRAPY|ISO"`, `separatepoles` disabled, and `singlepole` disabled. This eliminates all poles on the map. You get a world with continents and islands, but no poles. There is no edge to the map as you can navigate in all directions indefinitely.

Recipe: Overhead square tiles

Set topology `"WRAPX|WRAPY"`

Recipe: Hex tiles

Set topology `"WRAPX|WRAPY|ISO|HEX"` will give isometric hex tiles. Removing the `ISO` will give overhead hex tiles.

Sizing Your Map

The overall size of the map (total number of X and Y tiles) is driven by a collection of settings. They are:

- `mapsize`
- `size`
- `tilesperplayer`
- `xsize`
- `ysize`

The `mapsize` option is the driver and has three possible configurations:

1. `FULLSIZE` – When used, you must also have the `size` option set. The value is simply a number (in thousands) of tiles.
2. `PLAYER` – When used, you must also have the `tilesperplayer` option set. The map generator will take this into account and try its best to give each player a similar number of tiles to settle.
3. `XYSIZE` – When used, you must also have the `xsize` and `ysize` options set. These values are similar to the `size` option. Give the map generator very specific number of tiles on the two axis.

No recipes here. As a game master, you can figure out how big or small you want your map. Longturn games use the `tilesperplayer` option for their games as a reference.

Finalizing the Map and Player Placement

This is probably one of the hardest aspects of map generation. Actually getting the map you want as a game master, but also ensuring that player placement (or even team placement) is done the way you want is not exactly directly mapped. There is a bit of randomness involved.

These options are the last piece to defining a game map:

- `generator`
- `landmass`
- `mapseed`
- `startpos`
- `teampacement`

As with the `revealmap` option, discussed earlier, the `mapseed` option is used during testing to keep randomness from creeping into your testing. By setting a value, you eliminate the RNG in the server from impacting the game map you want to create. If a game master is also playing the game, it is recommended to disable (remove or comment out) this option when the game starts so even the game master does not have knowledge of the map at game start.

We will get to some recipes in a bit, but before we do that, let us talk about the `generator` and `startpos` options. They work in tandem with each other.

First up, `generator` has the following configurations:

- **SCENARIO** – This configuration is for Scenario games only. This is a special use case.
- **RANDOM** – The default. As the name implies, there is a dependency on the built-in Random Number Generator (RNG) in the server. The generator will attempt to create equally spaced, relatively small islands. Player placement will be impacted by the `landmass` option. The larger the value the bigger the continents/islands. This option is also impacted by the `mapsize` option. Best to use the `FULLSIZE` or the `XYSIZE` configuration.
- **FRACTAL** – This is the setting most Longturn games use. This configuration will create earth-like maps. By default, all players are placed on the same continent. The `landmass` option can also impact placement.
- **ISLAND** – Each player is placed on their own island. Each island is similar in size, but not shape.
- **FAIR** – Every player gets the exact same island.
- **FRACTURE** – Similar to **FRACTAL**, however this configuration often places mountains on the coasts.

Now let us discuss `startpos`, which has the following configurations:

- **DEFAULT** – The default. This configuration uses the `generator` configuration to place players.
- **SINGLE** – One player per island/continent.
- **2or3** – As the configuration name implies, the `startpos` will place 2 or 3 players together on an island/continent.
- **ALL** – Everyone is placed on the same continent. Make sure you give enough tiles when using this configuration. The `landmass` and `tilesperplayer` will come in handy.
- **VARIABLE** – The server will use the RNG to give a bit of randomness to player placement. The size of the continents will be taken into account.

Recipe: Large Pangea-like world

To create a gigantic single continent and have every player start there, begin with the no poles recipe above. Then set `generator` to **RANDOM**, or **FRACTAL**, or **FRACTURE** and set `landmass` 85 (the max). Finally set `startpos` "ALL".

If you want more ocean or poles, you can reduce the `landmass` setting and add poles with the recipe above. Longturn games use a `landmass` setting of 40 with poles for many games.

Recipe: An archipelago with players on their own island

To create an archipelago and start each player on their own island, begin with the navigable poles on the map recipe above. This is recommended from a realism perspective. If you do not want poles, you can skip it. Set `generator` to `ISLAND` or `FAIR` (`ISLAND` is recommended) and then set `startpos` `"SINGLE"`. Finally set `landmass` to 15 or 20 (minimum is 15). This will allow for some small random non-populated islands on the map.

This recipe introduces a great use of the `tilesperplayer` option. Each player will get a similar sized island of the number of tiles you define. Do not forget to change the `mapsize` option to `PLAYER` if you go this route.

By playing around with the varying values, you can create many different kinds of maps. Let us move to discussing team games and team placement.

Team games are a special use case. Most games are what the Longturn community calls Free For All (FFA). The idea of an FFA game is there are no preset alliances at game start. Those form in game. Team games are the opposite of FFA games. Alliances (e.g. teams) are defined before the game starts. Getting player placement correct when teams are involved is quite important!

The `teampacement` option has the following configurations:

- `DISABLED` – If set, then the option configuration is ignored.
- `CLOSEST` – The default. The name implies what happens.
- `CONTINENT` – Everyone on the same continent. This requires tuning `landmass`, `generator`, and `startpos` to fit how you want the teams to get placed on the same continent.
- `HORIZONTAL` – Place team players in a East-West alignment.
- `VERTICAL` – Place team players in a North-South alignment.

Recipe: Two team game with each team on their own continent

Follow the [steps](#) to create a `players.serv` file.

Add a read `players.serv` entry to your *Server Settings File*.

Set `generator` `"FRACTURE"`, `landmass` to 30 or 40, `mapsize` `"PLAYER"`, `tilesperplayer` to something between 1 and 1000 (Longturn uses 500), `teampacement` `"CONTINENT"`, and `startpos` `"ALL"`.

You can add poles to the map with the recipe above.

Note: Do not be surprised if you need to play around with some of the other settings to get the map you are looking for. Every Longturn game goes through a process of generating test maps for players to evaluate and vote for. Play around with the settings and you will get the map you eventually want!

3.4 Program Manual Pages

The following pages are used to generate man pages for distribution with the Linux packages.

3.4.1 freeciv21-client

SYNOPSIS

```
freeciv21-client [ -f|--file <FILE> ] [ -l|--log <FILE> ] [ -n|--name <NAME> ] [ -s|--server <HOST> ] [ -p|--port <PORT> ] [ -t|--tiles <FILE> ] [ -h|--help ] [ -v|--version ]
```

DESCRIPTION

Freeciv21 is a free open source turn-based empire-building 4x strategy game, in which each player becomes the leader of a civilization. You compete against several opponents to build cities and use them to support a military and an economy. Players strive to complete an empire that survives all encounters with its neighbors to emerge victorious. Play begins at the dawn of history in 4,000 BCE.

Freeciv21 takes its roots in the well-known FOSS game Freeciv and extends it for more fun, with a revived focus on competitive multiplayer environments. Players can choose from over 500 nations and can play against the computer or other people in an active online community.

The code is maintained by the team over at Longturn.net and is based on the QT framework. The game supports both hex and square tiles and is easily modified to create custom rules.

This is the client interface *program* used to connect to a Freeciv21 server. For more information on the server, refer to freeciv21-server(6). The program can be launched without any command line parameters. A player can use the main menu to connect to a running game, open an existing savegame, start a new game, or start a new scenario game. However, if you wish to automate the connection to a running server or start a new game with a server settings file, command line options are available.

OPTIONS

The following options are accepted on the command line of the client. They may not be combined as with other tools. For example: `freeciv21-client -sp localhost 5556` will not work. Instead you will need to enter each option separately, such as, `freeciv21-client -s localhost -p 5556`.

- A, --Announce <PROTO>**
Announce game in LAN using protocol PROTO (IPv4/IPv6/none), Default is IPv4. Options include: IPv4, IPv6, or none.
- a, --autoconnect**
Skip connection dialog (usually with *url* or *-n*, *-s*, and *-p*).
- d, --debug <LEVEL>**
Set debug log level (fatal/critical/warning/info/debug). Default log level is `info`.
- F, --Fatal**
Raise a signal on failed assertion. An assertion is a code calculation error. With this set, the client process will SEGFAULT instead of issuing a warning message to the terminal console.
- f, --file <FILE>**
Load saved game FILE. Useful when wanting to restart a game.
- H, --Hackless**
Do not request hack access to local, but not spawned, server.
- l, --log <FILE>**
Use FILE as logfile.
- M, --Meta <HOST>**
Connect to the metaserver at HOST.

Note: With regard to the command line arguments concerning the metaserver, Freeciv21 does not have its own metaserver at this time and the legacy Freeciv metaserver does not support Freeciv21. This means, right now, that all commands related to the metaserver are held over from the fork from legacy Freeciv until the Longturn community creates a custom metaserver.

-n, --name <NAME>
Use NAME as username on server.

-p, --port <PORT>
Connect to server port PORT (usually with -a).

-P, --Plugin <PLUGIN>
Use PLUGIN for sound output []. The default is *SDL*.

-r, --read <FILE>
Read startup script FILE. Options are passed to the spawned server.

-s, --server <HOST>
Connect to the server at HOST (usually with -a).

-S, --Sound <FILE>
Read sound tags from FILE. Default is *stdsounds.soundspec*.

-m, --music <FILE>
Read music tags from FILE. Default is *stdmusic.musicspec*.

-t, --tiles <FILE>
Use data file *FILE.tilespec* for tiles.

-w, --warnings
Warn about deprecated modpack constructs.

-h, --help
Display help on command line options.

--help-all
Display help including Qt specific options.

-v, --version
Display version information.

url
Server information in URL format. See examples below.

EXAMPLES

Here are some examples:

freeciv21-client -n mycoolusername -s localhost -p 2004

Connect to an already running server with a username of *mycoolusername*, server host name of *localhost*, and port of *2004*.

freeciv21-client “”fc21://[username]:[password]@[server]:[port]”” -a -t amplio2

Connect to an already running server using *URL* format. The parameters *[username]*, *[password]*, *[server]*, and *[port]* are replaced with valid values. This command also passes the autoconnect option (-a) as well as loads the Amplio2 Tileset.

Note: The communication between the client and the server is plain text, so make sure the password you use is unique. This is especially true for Longturn games on the open Internet.

ENVIRONMENT

The Freeciv21 client accepts these environment variables:

FREECIV_DATA_PATH

A colon separated list of directories pointing to the Freeciv21 data directories. By default Freeciv21 looks in the following directories, in order, for any data files: the current directory; the “data” subdirectory of the current directory; the subdirectory “.local/share/freeciv21” in the user’s home directory; and the directory where the files are placed by running “cmake –target install”.

HOME

Specifies the user’s home directory.

http_proxy

Set this variable accordingly when using a proxy. For example, “”http://my_proxy_address/””.

LANG or LANGUAGE

Sets the language and locale on some platforms.

LC_ALL or LC_CTYPE

Similar to LANG (see documentation for your system).

USER

Specifies the username of the current user.

3.4.2 freeciv21-server

SYNOPSIS

```
freeciv21-server [ -f|--file <FILE> ] [ -l|--log <FILE> ] [ -p|--port <PORT> ] [ -r|--read <FILE> ] [ -s|--saves  
<DIR> ] [ --ruleset <RULESET> ] [ -a|--auth ] [ -D|--Database <FILE> ] [ -G|--Guests ] [ -N|--Newusers ] [ -h|--help ]  
[ -v|--version ]
```

DESCRIPTION

Freeciv21 is a free open source turn-based empire-building 4x strategy game, in which each player becomes the leader of a civilization. You compete against several opponents to build cities and use them to support a military and an economy. Players strive to complete an empire that survives all encounters with its neighbors to emerge victorious. Play begins at the dawn of history in 4,000 BCE.

Freeciv21 takes its roots in the well-known FOSS game Freeciv and extends it for more fun, with a revived focus on competitive multiplayer environments. Players can choose from over 500 nations and can play against the computer or other people in an active online community.

The code is maintained by the team over at Longturn.net and is based on the QT framework. The game supports both hex and square tiles and is easily modified to create custom rules.

An HTML version of this manual page along with much more information is available on our documentation website at <https://longturn.readthedocs.io/>.

This is the server *program* used to establish a Freeciv21 server. For more information on the game, refer to freeciv21-client(6).

This manual page only lists the command line arguments for the server *program*. For details of the directives a server administrator can use to control aspects of the server, see `freeciv21-game-manual(6)`.

OPTIONS

The following options are accepted on the command line of the server. They may not be combined as with other tools. For example: `freeciv21-server -fp savegame.sav 5556` will not work. Instead you will need to enter each option separately, such as, `freeciv21-server -s savegame.sav -p 5556`.

Tip: It is generally considered a best practice to write a `.sh` script to run your server. This way you do not have to remember all the command line options to use every time you run your server.

-A, --Announce <PROTO>

Announce game in LAN using protocol PROTO (IPv4/IPv6/none). Default is IPv4. Options include: IPv4, IPv6, or none.

-B, --Bind-meta <ADDR>

Connect to metaserver from this address. Default will be your public IP address assigned from your internet service provider.

Note: With regard to the command line arguments concerning the metaserver, Freeciv21 does not have its own metaserver at this time and the legacy Freeciv metaserver does not support Freeciv21. This means, right now, that all commands related to the metaserver are held over from the fork from legacy Freeciv until the Longturn community creates a custom metaserver.

-b, --bind <ADDR>

Listen for clients on ADDR. Default is the IP address of the server computer.

-d, --debug <LEVEL>

Set debug log level (fatal/critical/warning/info/debug). Default log level is `info`.

-e, --exit-on-end

When a game ends, exit instead of restarting.

-F, --Fatal

Raise a signal on failed assertion. An assertion is a code calculation error. With this set, the server process will SEGFAULT instead of issuing a warning message to the terminal console.

-f, --file <FILE>

Load saved game FILE. Useful when wanting to restart a game.

-i, --identity <ADDR>

Be known as ADDR at metaserver or LAN client.

-k, --keep

Keep updating game information on metaserver even after failure. Default is not to keep updating.

-l, --log <FILE>

Use FILE as logfile. Generally a very good idea when running a server.

-M, --Metaserver <ADDR>

Set ADDR as metaserver address. Allows you to point to a specific metaserver instead of the default: <https://meta.freeciv.org/>

-m, --meta

Notify metaserver and send server's info.

- p, --port <PORT>**
Listen for clients on port PORT. Default is 5556 as assigned by [IANA](#)
- q, --quitidle <TIME>**
Quit if no players for TIME seconds.
- R, --Ranklog <FILE>**
Use FILE as ranking logfile. Generally a very good idea when running a server.
- r, --read <FILE>**
Read startup file FILE.
- S, --Serverid <ID>**
Sets the server id to ID.
- s, --saves <DIR>**
Save games to directory DIR. Generally a very good idea to save games at least every turn and depending on how long the turns are set to, to save within a turn. In the case of a server crash, restarting from a save comes in very handy.
- t, --timetrack**
Prints stats about elapsed time on misc tasks. Typically used to test code performance.
- w, --warnings**
Warn about deprecated modpack constructs.
- ruleset <RULESET>**
Load ruleset RULESET. Default is the Civ2Civ3 ruleset.
- scenarios <DIR>**
Save scenarios to directory DIR.
- a, --auth**
Enable database authentication (requires --Database).
- D, --Database <FILE>**
Enable database connection with configuration from FILE.
- G, --Guests**
Allow guests to login if auth is enabled. See
- N, --Newusers**
Allow new users to login if auth is enabled.
- h, --help**
Display help on command line options and quits.
- help-all**
Display help including Qt specific options and quits.
- v, --version**
Display version information and quits.

EXAMPLES

Here are some examples:

freeciv21-server -file oldgame.sav -p 2224

Starts a server on port 2224, loading the savegame named *oldgame.sav*.

freeciv21-server -R ranklog -l logfile -r start-script -f oldgame.sav.xz -p 2224

Starts a server on port 2224, loading the savegame named *oldgame.sav.xz*. Player ranking events are logged to *ranklog*, other logging events are placed in *logfile*. When the server first starts, it immediately executes the server settings script named *start-script*. See *freeciv21-server-manual(6)* for more information on the server settings script.

freeciv21-server -a -D users.conf -G -N -q 60 -l logfile -s ./saves -p 2224

Starts a server on port 2224 with authentication enabled (*-a*, *-D*, *-G*, and *-N*). If no activity for 60 seconds the server will quit (*-q*). Logging events are placed in *logfile* and periodic saves are placed in the *./saves* directory.

FILES

The *freeciv21-server* requires the default ruleset files to be readable at start.

- *civ2civ3/buildings.ruleset*
- *civ2civ3/cities.ruleset*
- *civ2civ3/effects.ruleset*
- *civ2civ3/game.ruleset*
- *civ2civ3/governments.ruleset*
- *civ2civ3/nations.rulest*
- *civ2civ3/styles.ruleset*
- *civ2civ3/techs.ruleset*
- *civ2civ3/terrain.ruleset*
- *civ2civ3/units.ruleset*
- *civ2civ3/script.lua*
- *default/default.lua*
- *default/nationlist.ruleset*

These files are for the default ruleset for the game (*civ2civ3*), which are loaded if you do not use the *-ruleset* argument. Alternate rules can be loaded with the *rulesetdir* directive in a start up script. Type *help rulesetdir* at the server command prompt for more information.

If you are going to enable user authentication, you will need the *database.lua* file. The file is available on the GitHub repository (see below).

ENVIRONMENT

The Freeciv21 server accepts these environment variables:

FREECIV_CAPS

A string containing a list of “capabilities” provided by the server. The compiled default should be correct for most purposes, but if you are familiar with the capability string facility in the source you may use it to enforce some constraints between clients and server.

FREECIV_COMPRESSION_LEVEL

Sets the compression level for network traffic.

FREECIV_MULTICAST_GROUP

Sets the multicast group (for the LAN tab).

FREECIV_DATA_PATH

A colon separated list of directories pointing to the Freeciv21 data directories. By default Freeciv21 looks in the following directories, in order, for any data files: the current directory; the “data” subdirectory of the current directory; the subdirectory “.local/share/freeciv21” in the user’s home directory; and the directory where the files are placed by running “cmake –target install”.

FREECIV_SAVE_PATH

A colon separated list of directories pointing to the Freeciv21 save directories. By default Freeciv21 looks in the following directories, in order, for save files: the current directory; and the subdirectory “.local/share/freeciv21/saves” in the user’s home directory.

(This does not affect where the server creates save game files. See the *–saves* option for that.)

FREECIV_SCENARIO_PATH

A colon separated list of directories pointing to the Freeciv21 scenario directories. By default Freeciv21 looks in the following directories, in order, for scenario files: the current directory; the “data/scenarios” subdirectory of the current directory; the subdirectory “.local/share/freeciv21/scenarios” in the user’s home directory; and the directory where the files are placed by running “cmake –target install”.

(This does not affect where the server creates scenario files. See the *–scenarios* option for that.)

HOME

Specifies the user’s home directory. For example, “`http://my_proxy_address/`”.

http_proxy

Set this variable accordingly when using a proxy.

LANG or LANGUAGE

Sets the language and locale on some platforms.

LC_ALL or LC_CTYPE

Similar to LANG (see documentation for your system).

USER

Specifies the username of the current user.

BUGS

Please report bugs to the Freeciv21 bug tracker at <https://github.com/longturn/freeciv21/issues/new/choose>

MORE INFORMATION

See the Longturn home page at <https://longturn.net/>. You can also find the code repository at <https://github.com/longturn/freeciv21/>.

3.4.3 freeciv21-game-manual

SYNOPSIS

This manual page refers to freeciv21-server(6) setting directives and has no OS command line options.

DESCRIPTION

Freeciv21 is a free open source turn-based empire-building 4x strategy game, in which each player becomes the leader of a civilization. You compete against several opponents to build cities and use them to support a military and an economy. Players strive to complete an empire that survives all encounters with its neighbors to emerge victorious. Play begins at the dawn of history in 4,000 BCE.

Freeciv21 takes its roots in the well-known FOSS game Freeciv and extends it for more fun, with a revived focus on competitive multiplayer environments. Players can choose from over 500 nations and can play against the computer or other people in an active online community.

The code is maintained by the team over at Longturn.net and is based on the QT framework. The game supports both hex and square tiles and is easily modified to create custom rules.

The freeciv21-server(6) program has its own command line interface. This page describes all of the server option directives an administrator can issue to the freeciv21-server. For more information on the server, refer to freeciv21-server(6). For more information on the game, refer to freeciv21-client(6).

OPTIONS

After a server has been started from the command-line, an administrator can issue a set of commands to the server's own command-line. This server command-line is separate from the OS terminal command-line.

/start

This command starts the game. When starting a new game, it should be used after all human players have connected, and *AI* players have been created (if required), and any desired changes to initial server options have been made. After **/start**, each human player will be able to choose their nation, and then the game will begin. This command is also required after loading a savegame for the game to recommence. Once the game is running this command is no longer available, since it would have no effect.

/help

With no arguments gives some introductory help. With argument "commands" or "options" gives respectively a list of all commands or all options. Otherwise the argument is taken as a command name or option name, and help is given for that command or option. For options, the help information includes the current and default values for that option. The argument may be abbreviated where unambiguous.

/list colors

List the player colors.

/list connections

Gives a list of connections to the server.

/list delegations

List of all player delegations.

/list ignored users

List of a player's ignore list.

/list map image definitions

List of defined map images.

/list players

The list of the players in the game.

/list rulesets

List of the available rulesets (for `/read` command).

/list scenarios

List of the available scenarios.

/list nationsets

List of the available nation sets in this ruleset.

/list teams

List of the teams of players.

/list votes

List of the running votes.

/quit

Quit the game and shutdown the server.

/cut <connection-name>

Cut specified client's connection to the server, removing that client from the game. If the game has not yet started that client's player is removed from the game, otherwise there is no effect on the player. Note that this command now takes connection names, not player names. See `/list connections`.

/explain <option-name>

The `/explain` command gives a subset of the functionality of `/help`, and is included for backward compatibility. With no arguments it gives a list of options (such as `/help options`), and with an argument it gives help for a particular option (such as `/help <option-name>`).

/show all|vital|situational|rare|changed|locked|rulesetdir

With no arguments, shows vital server options (or available options, when used by clients). With an option name argument, show only the named option, or options with that prefix. With `all`, it shows all options. With `vital`, `situational`, or `rare`, a set of options with this level. With `changed`, it shows only the options which have been modified from the ruleset defaults. While with `locked` all settings locked by the ruleset will be listed. With `ruleset`, it will show the current ruleset directory name.

/wall <message>

For each connected client, pops up a window showing the message entered.

/connectmsg <message>

Set message to send to clients when they connect. Empty message means that no message is sent.

/vote yes|no|abstain [vote number]

A player with basic *level access* issuing a control level command starts a new vote for the command. The `/vote` command followed by "yes", "no", or "abstain", and optionally a vote number, gives your vote. If you do not add a vote number, your vote applies to the latest vote. You can only suggest one vote at a time. The vote will pass immediately if more than half of the voters who have not abstained vote for it, or fail immediately if at least half of the voters who have not abstained vote against it.

Note: Voting is not a feature that is used very often, but does come in handy.

/cancelvote <vote number>

With no arguments this command removes your own vote. If you have an admin access level, you can cancel any vote by vote number, or all votes with the all argument.

/debug diplomacy|ferries|tech|city|units|unit|timing|info

Print [AI](#) debug information about given entity and turn continuous debugging output for this entity on or off.

- debug diplomacy <player>
- debug ferries
- debug tech <player>
- debug city <x> <y>
- debug units <x> <y>
- debug unit <id>
- debug timing
- debug info

/set <option-name> <value>

Set an option on the server. The syntax and legal values depend on the option. See the help for each option. Some options are “bitwise”, in that they consist of a choice from a set of values. Separate these with |, for instance, /set topology wrapx|iso. For these options, use syntax like /set topology "" to set no values.

/team <player> <team>

A team is a group of players that start out allied, with shared vision, embassies, and fight together to achieve team victory with averaged individual scores. Each player is always a member of a team (possibly the only member). This command changes which team a player is a member of. Use "" if names contain whitespace.

/rulesetdir <directory>

Choose new ruleset directory or modpack.

/metamessage <meta-line>

Set user defined metaserver info line. If parameter is omitted, previously set metamessage will be removed. For most of the time user defined metamessage will be used instead of automatically generated messages, if it is available.

Note: Freeciv21 does not have its own metaserver at this time and the legacy Freeciv metaserver does not support Freeciv21. This means, right now, that all commands related to the metaserver are held over from the fork from legacy Freeciv until the Longturn community creates a custom metaserver.

/metapatches <meta-line>

Set metaserver patches line. See Note about Freeciv21 metaserver above.

/metaconnection up|down|persistent|?

/metaconnection ? reports on the status of the connection to the metaserver. /metaconnection down or /metac d brings the metaserver connection down. /metaconnection up or /metac u brings the metaserver connection up. /metaconnection persistent or /metac p is like ‘up’, but keeps trying after failures. See Note about Freeciv21 metaserver above.

/metaserver <address>

Set address (URL) for metaserver to report to. Same as --Metaserver on the command-line. See NOTE about Freeciv21 metaserver above.

/aitoggle <player-name>

Toggle *AI* status of player. By default, new players are AI.

/take <player-name>

Only the console and connections with `cmdlevel hack` can force other connections to take over a player. If you are not one of these, only the `<player-name>` argument is allowed. If `-` is given for the player name and the connection does not already control a player, one is created and assigned to the connection. The `/allowtake option` controls which players may be taken and in what circumstances.

For example, if you have `cmdlevel hack` and are connected to a server, you can issue `/take <player-name>` `-` to take over any player. If you do not have `cmdlevel hack`, then the `/allowtake option` must be properly set as well as a proper `/delegate command` by the player wishing to delegate is completed first. Then a player can use `/take` to take the player while the delegation is in place.

/observe <player-name>

Only the console and connections with `cmdlevel hack` can force other connections to observe a player. If you are not one of these, only the `<player-name>` argument is allowed. If the console gives no player-name or the connection uses no arguments, then the connection is attached to a global observer. The `/allowtake option` controls which players may be observed and in what circumstances.

For example, if you have `cmdlevel hack` and are connected to a server, you can issue `/observe` with no `<player-name>` parameter. The server will change your connection to a global observer, able to view all nations. A global observer can make no changes and can only see information. If a user with `cmdlevel hack` issues `/observer <player-name>`, then they can only observe that particular nation only. To restore to original connection, you issue `/take <player-name>` for your own username.

/detach <connection-name>

Only the console and connections with `cmdlevel hack` can force other connections to detach from a player.

This rarely used command essentially forces a connected client to disconnect from a server. To see the connections, issue a `/list connections` command as noted *above*.

/create <player-name> [*ai type*]

With the `/create` command a new player with the given name is created. If `player-name` is empty, a random name will be assigned when the game begins. Until then the player will be known by a name derived from its type. The `ai type` parameter can be used to select which *AI* module will be used for the created player. This requires that the respective module has been loaded or built in to the server. If the game has already started, the new player will have no units or cities. Also, if no free player slots are available, the slot of a dead player can be reused (removing all record of that player from the running game).

/away

Toggles away mode for your nation. In away mode, the *AI* will govern your nation but make only minimal changes.

Note: The term *minimal changes* is not well understood at this time. The server help does not provide more details. An enterprising enthusiast could read the *AI* code to determine what the term means and provide more details. Any real player is not going to want the AI to run their nation and will *delegate* instead.

/handicapped <player-name>

With no arguments, sets all *AI* players to skill level Handicapped, and sets the default level for any new AI players to Handicapped. With an argument, sets the skill level for the specified player only. This skill level has the same features as Novice, but may suffer additional ruleset-defined penalties.

- Does not build offensive diplomatic units.
- Gets reduced bonuses from huts.
- Prefers defensive buildings and avoids close diplomatic relations.

- Can see through *FOW*.
- Does not build air units.
- Has complete map knowledge, including unexplored territory.
- Naive at diplomacy.
- Limits growth to match human players.
- Believes its cities are always under threat.
- Always offers cease-fire on first contact.
- Does not bribe worker or city founder units.
- Has erratic decision-making.
- Research takes 250% as long as usual.
- Has reduced appetite for expansion.

/novice <player-name>

With no arguments, sets all *AI* players to skill level Novice, and sets the default level for any new AI players to Novice. With an argument, sets the skill level for the specified player only.

- Does not build offensive diplomatic units.
- Gets reduced bonuses from huts.
- Prefers defensive buildings and avoids close diplomatic relations.
- Can see through *FOW*.
- Does not build air units.
- Has complete map knowledge, including unexplored territory.
- Naive at diplomacy.
- Limits growth to match human players.
- Believes its cities are always under threat.
- Always offers cease-fire on first contact.
- Does not bribe worker or city founder units.
- Has erratic decision-making.
- Research takes 250% as long as usual.
- Has reduced appetite for expansion.

/easy <player-name>

With no arguments, sets all *AI* players to skill level Easy, and sets the default level for any new AI players to Easy. With an argument, sets the skill level for the specified player only.

- Does not build offensive diplomatic units.
- Gets reduced bonuses from huts.
- Prefers defensive buildings and avoids close diplomatic relations.
- Can see through *FOW*.
- Does not build air units.
- Has complete map knowledge, including unexplored territory.

- Naive at diplomacy.
- Limits growth to match human players.
- Always offers cease-fire on first contact.
- Does not bribe worker or city founder units.
- Can change city production type without penalty.
- Has erratic decision-making.
- Has reduced appetite for expansion.

/normal <player-name>

With no arguments, sets all *AI* players to skill level **Normal**, and sets the default level for any new AI players to **Normal**. With an argument, sets the skill level for the specified player only.

- Does not build offensive diplomatic units.
- Can see through *FOW*.
- Has complete map knowledge, including unexplored territory.
- Can skip anarchy during revolution.
- Always offers cease-fire on first contact.
- Does not bribe worker or city founder units.
- Can change city production type without penalty.

/hard <player-name>

With no arguments, sets all *AI* players to skill level **Hard**, and sets the default level for any new AI players to **Hard**. With an argument, sets the skill level for the specified player only.

- Has no restrictions on national budget.
- Can target units and cities in unseen or unexplored territory.
- Knows the location of huts in unexplored territory.
- Can see through *FOW*.
- Has complete map knowledge, including unexplored territory.
- Can skip anarchy during revolution.
- Can change city production type without penalty.

/cheating <player-name>

With no arguments, sets all *AI* players to skill level **Cheating**, and sets the default level for any new AI players to **Cheating**. With an argument, sets the skill level for the specified player only.

- Can target units and cities in unseen or unexplored territory.
- Knows the location of huts in unexplored territory.
- Can see through *FOW*.
- Has complete map knowledge, including unexplored territory.
- Can skip anarchy during revolution.
- Can change city production type without penalty.

/experimental <player-name>

With no arguments, sets all *AI* players to skill level **Experimental**, and sets the default level for any new AI players to **Experimental**. With an argument, sets the skill level for the specified player only. **THIS IS ONLY FOR TESTING OF NEW AI FEATURES!** For ordinary servers, this level is no different to **Hard**.

- Has no restrictions on national budget.
- Can target units and cities in unseen or unexplored territory.
- Knows the location of huts in unexplored territory.
- Can see through *FOW*.
- Has complete map knowledge, including unexplored territory.
- Can skip anarchy during revolution.
- Can change city production type without penalty.

/cmdlevel none|info|basic|ctrl|admin|hack

The command access level controls which server commands are available to users via the client chatline. The available levels are:

- **none**: no commands
- **info**: informational or observer commands only
- **basic**: commands available to players in the game
- **ctrl**: commands that affect the game and users
- **admin**: commands that affect server operation
- **hack**: *all* commands - dangerous!

With no arguments, the current command access levels are reported. With a single argument, the level is set for all existing connections, and the default is set for future connections. If **new** is specified, the level is set for newly connecting clients. If **first come** is specified, the **first come** level is set. It will be granted to the first client to connect, or if there are connections already, the first client to issue the **/first** command. If a connection name is specified, the level is set for that connection only. Command access levels do not persist if a client disconnects, because some untrusted person could reconnect with the same name. Note that this command now takes connection names, not player names.

/first

If there is none, become the game organizer with increased permissions.

/timeoutshow

Shows information about the timeout for the current turn, for instance how much time is left.

/timeoutset <time>

This command changes the remaining time for the current turn. Passing a value of **0** ends the turn immediately. The time is specified as hours, minutes, and seconds using the format **hh:mm:ss** (minutes and hours are optional).

/timeoutadd <time>

This increases the timeout for the current turn, giving players more time to finish their actions. The time is specified as hours, minutes, and seconds using the format **hh:mm:ss** (minutes and hours are optional). Negative values are allowed

/timeoutincrease <turn> <turninc> <value> <valuemult>

Every **<turn>** turns, add **<value>** to the timeout timer, then add **<turninc>** to **<turn>** and multiply **<value>** by **<valuemult>**. Use this command in concert with the option **/timeout**. Defaults are **0 0 0 1**.

/ignore [type=]<pattern>

The given pattern will be added to your ignore list. You will not receive any messages from users matching

this pattern. The type may be either `user`, `host`, or `ip`. The default type (if omitted) is to match against the username. The pattern supports unix glob style wildcards, i.e., `*` matches zero or more character, `?` exactly one character, `[abc]` exactly one of `a`, `b`, or `c`, etc. To access your current ignore list, issue `/list ignore`.

`/unignore <range>`

The ignore list entries in the given range will be removed. You will be able to receive messages from the respective users. The range argument may be a single number or a pair of numbers separated by a dash `-`. If the first number is omitted, it is assumed to be 1. If the last is omitted, it is assumed to be the last valid ignore list index. To access your current ignore list, issue `/list ignore`.

`/playercolor <player-name> <color>`

This command sets the color of a specific player, overriding any color assigned according to the `plrcolormode` setting. The color is defined using hexadecimal notation (hex) for the combination of Red, Green, and Blue color components (RGB), similarly to HTML. For each component, the lowest (darkest) value is 0 (in hex: 00), and the highest value is 255 (in hex: FF). The color definition is simply the three hex values concatenated together (RRGGBB). For example, the following command sets Caesar to pure red: `playercolor Caesar ff0000`. Before the game starts, this command can only be used if the `plrcolormode` setting is set to `PLR_SET`. A player's color can be unset again by specifying `reset`. Once the game has started and colors have been assigned, this command changes the player color in any mode; `reset` cannot be used. To list the player colors, use `/list colors`.

`/playernation <player-name> [nation] [is-male] [leader] [style]`

This command sets the nation, leader name, style, and gender of a specific player. The string "random" can be used to select a random nation. The gender parameter should be 1 for male, 0 for female. Omitting any of the player settings will reset the player to defaults. This command may not be used once the game has started.

`/endgame`

End the game immediately in a draw.

`/surrender`

This tells everyone else that you concede the game, and if all but one player (or one team) have conceded the game in this way then the game ends.

`/remove <player-name>`

This *completely* removes a player from the game, including all cities and units etc. Use with care!

`/save <file-name>`

Save the current game to file `<file-name>`. If no `file-name` argument is given saves to `<auto-save name prefix><year>m.sav[.gz]`. To reload a savegame created by `/save`, start the server with the command-line argument: `--file <filename>` or `-f <filename>` and use the `/start` command once players have reconnected.

`/scensave <file-name>`

Save the current game to file `<file-name>` as a scenario. If no `file-name` argument is given saves to `<auto-save name prefix><year>m.sav[.gz]`. To reload a savegame created by `/scensave`, start the server with the command-line argument: `--file <filename>` or `-f <filename>` and use the `/start` command once players have reconnected.

`/load <file-name>`

Load a game from `<file-name>`. Any current data including players, rulesets and server options are lost.

`/read <file-name>`

Process server commands from file.

`/write <file-name>`

Write current settings as server commands to file.

`/reset game|ruleset|script|default`

Reset all settings if it is possible. The following levels are supported:

- **game**: using the values defined at the game start.
- **ruleset**: using the values defined in the ruleset.
- **script**: using default values and rereading the start script.
- **default**: using default values.

/default <option name>

Reset the option to its default value. If the default ever changes in a future version, the option's value will follow that change.

/lua cmd <script line>

Evaluate a line of Freeciv21 script or a Freeciv script file in the current game. Variations are:

- **lua cmd <script line>**
- **lua unsafe-cmd <script line>**
- **lua file <script file>**
- **lua unsafe-file <script file>**

The unsafe prefix runs the script in an instance separate from the ruleset. This instance does not restrict access to Lua functions that can be used to hack the computer running the Freeciv21 server. Access to it is therefore limited to the console and connections with cmdlevel **hack**.

/kick <user>

The connection given by the **user** argument will be cut from the server and not allowed to reconnect. The time the user would not be able to reconnect is controlled by the **kicktime** setting.

/delegate to <username>

Delegation allows a user to nominate another user who can temporarily take over control of their player while they are away. Variations are:

- **/delegate to <username>**: Allow **<username>** to **delegate** take your player.
- **/delegate cancel**: Nominated user can no longer take your player.
- **/delegate take <player-name>**: Take control of a player who has been delegated to you. Behaves like **/take**, except that the **/allowtake** restrictions are not enforced.
- **/delegate restore**: Relinquish control of a delegated player (opposite of **/delegate take**) and restore your previous view, if any. This also happens automatically if the player's owner reconnects.
- **/delegate show**: Show who control of your player is currently delegated to, if anyone.

The **[player-name]** argument can only be used by connections with cmdlevel **admin** or above to force the corresponding change of the delegation status.

/aicmd <player> <command>

Execute a command in the context of the **AI** for the given player.

/fcdb lua <script>

The argument **reload** causes the database script file to be re-read after a change, while the argument **lua** evaluates a line of Lua script in the context of the Lua instance for the database.

/mapping define <mapdef>

Create image files of the world/player map. Variations are:

- **mapping define <mapdef>**
- **mapping show <id>|all**
- **mapping create <id>|all**

- `mapimg delete <id>|all`
- `mapimg colortest`

This command controls the creation of map images. Supported arguments:

- `define <mapdef>`: define a map image; returns numeric `<id>`.
- `show <id>|all`: list map image definitions or show a specific one.
- `create <id>|all`: manually save image(s) for current map state.
- `delete <id>|all`: delete map image definition(s).
- `colortest`: create test image(s) showing all colors.

Multiple definitions can be active at once. A definition `<mapdef>` consists of colon-separated options:

Table 3.2: mapdef options

Option	(Default)	Description
<code>format=<[tool]]format></code>	(ppm ppm)	file format
<code>show=<show></code>	(all)	which players to show
<code>plrname=<name></code>		player name
<code>plrid=<id></code>		numeric player id
<code>plrbv=<bit vector></code>		see example; first char = id 0
<code>turns=<turns></code>	(1)	save image each <code><turns></code> turns (0=no autosave, save with create)
<code>zoom=<zoom></code>	(2)	magnification factor (1-5)
<code>map=<map></code>	(bcku)	which map layers to draw

`<[tool]]format>` = use image format `<format>`, optionally specifying toolkit `<tool>`. The following toolkits and formats are compiled in:

- 0: ppm

`<show>` determines which players are represented and how many images are saved by this definition:

- `none`: no players, only terrain.
- `each`: one image per player.
- `human`: one image per human player.
- `all`: all players on a single image.
- `plrname`: just the player named with `plrname`.
- `plrid`: just the player specified with `plrid`.
- `plrbv`: one image per player in `plrbv`.

`<map>` can contain one or more of the following layers:

- `a`: show area within borders of specified players.
- `b`: show borders of specified players.
- `c`: show cities of specified players.
- `f`: show fog of war (single-player images only).
- `k`: show only player knowledge (single-player images only).

- **t**: full display of terrain types.
- **u**: show units of specified players.

Examples of `<mapdef>`:

- `zoom=1:map=tcub:show=all:format=ppm|ppm`
- `zoom=2:map=tcub:show=each:format=png`
- `zoom=1:map=tcub:show=plrname:plrname=Otto:format=gif`
- `zoom=3:map=cu:show=plrby:plrby=010011:format=jpg`
- `zoom=1:map=t:show=none:format=magick|jpg`

/rfcstyle

Switch server output between ‘RFC-style’ and normal style.

/serverid

Simply returns the id of the server.

USER

Specifies the username of the current user.

BUGS

Please report bugs to the Freeciv21 bug tracker at <https://github.com/longturn/freeciv21/issues/new/choose>

MORE INFORMATION

See the Longturn home page at <https://longturn.net/>. You can also find the code repository at <https://github.com/longturn/freeciv21/>.

3.4.4 freeciv21-manual

SYNOPSIS

```
freeciv21-manual [ -l|--log <FILE> ] [ -r|--ruleset <RULESET> ] [ -w|--wiki ] [ -h|--help ] [ -v|--version ]
```

DESCRIPTION

Freeciv21 is a free open source turn-based empire-building 4x strategy game, in which each player becomes the leader of a civilization. You compete against several opponents to build cities and use them to support a military and an economy. Players strive to complete an empire that survives all encounters with its neighbors to emerge victorious. Play begins at the dawn of history in 4,000 BCE.

Freeciv21 takes its roots in the well-known FOSS game Freeciv and extends it for more fun, with a revived focus on competitive multiplayer environments. Players can choose from over 500 nations and can play against the computer or other people in an active online community.

The code is maintained by the team over at Longturn.net and is based on the QT framework. The game supports both hex and square tiles and is easily modified to create custom rules.

This is a program to read a set of *RULESET* files and write a manual to a set of HTML files of the same name.

OPTIONS

The following options are accepted on the command line of the manual generator. They may not be combined as with other tools. For example: `freeciv21-manual -rw civ2civ3` will not work. Instead you will need to enter each option separately, such as, `freeciv21-manual -r civ2civ3 -w`.

-d, --debug <LEVEL>

Set debug log level (fatal/critical/warning/info/debug). Default log level is `info`.

-F, --Fatal

Raise a signal on failed assertion. An assertion is a code calculation error. With this set, the client process will SEGFAULT instead of issuing a warning message to the terminal console.

-l, --log <FILE>

Use FILE as logfile.

-r, --ruleset <RULESET>

Make manual for RULESET.

-w, --wiki

Write manual in wiki format.

-h, --help

Display help on command line options.

--help-all

Display help including Qt specific options.

-v, --version

Display version information.

USER

Specifies the username of the current user.

BUGS

Please report bugs to the Freeciv21 bug tracker at <https://github.com/longturn/freeciv21/issues/new/choose>

MORE INFORMATION

See the Longturn home page at <https://longturn.net/>. You can also find the code repository at <https://github.com/longturn/freeciv21/>.

3.4.5 freeciv21-modpack-qt

SYNOPSIS

`freeciv21-modpack-qt [-i|--install <URL>] [-L|--List <URL>] [-h|--help] [-v|--version]`

DESCRIPTION

Freeciv21 is a free open source turn-based empire-building 4x strategy game, in which each player becomes the leader of a civilization. You compete against several opponents to build cities and use them to support a military and an economy. Players strive to complete an empire that survives all encounters with its neighbors to emerge victorious. Play begins at the dawn of history in 4,000 BCE.

Freeciv21 takes its roots in the well-known FOSS game Freeciv and extends it for more fun, with a revived focus on competitive multiplayer environments. Players can choose from over 500 nations and can play against the computer or other people in an active online community.

The code is maintained by the team over at Longturn.net and is based on the QT framework. The game supports both hex and square tiles and is easily modified to create custom rules.

This is a program to read and install a modpack URL from the command line. When given no command line parameters the Qt base GUI will load and allow for modpack installation. For a pure command line modpack installer, see freeciv21-modpack(6).

OPTIONS

The following options are accepted on the command line of the modpack installer. They may not be combined as with other tools. For example: `freeciv21-modpack-qt -Lp https://raw.githubusercontent.com/longturn/modpacks/main/index.json ~/.local/fc21` will not work. Instead you will need to enter each option separately, such as, `freeciv21-modpack-qt -L https://raw.githubusercontent.com/longturn/modpacks/main/index.json -p ~/.local/fc21`.

-d, --debug <LEVEL>

Set debug log level (fatal/critical/warning/info/debug). Default log level is `info`.

-i, --install <URL>

Automatically install modpack from a given URL.

-L, --List <URL>

Load modpack list from given URL.

-p, --prefix <DIR>

Install modpacks to given directory hierarchy. Default location is `~/.local/share/freeciv21`.

-h, --help

Display help on command line options.

--help-all

Display help including Qt specific options.

-v, --version

Display version information.

USER

Specifies the username of the current user.

BUGS

Please report bugs to the Freeciv21 bug tracker at <https://github.com/longturn/freeciv21/issues/new/choose>

MORE INFORMATION

See the Longturn home page at <https://longturn.net/>. You can also find the code repository at <https://github.com/longturn/freeciv21/>.

3.4.6 freeciv21-modpack

SYNOPSIS

```
freeciv21-modpack [ -i|--install <URL> ] [ -L|--List <URL> ] [ -h|--help ] [ -v|--version ]
```

DESCRIPTION

Freeciv21 is a free open source turn-based empire-building 4x strategy game, in which each player becomes the leader of a civilization. You compete against several opponents to build cities and use them to support a military and an economy. Players strive to complete an empire that survives all encounters with its neighbors to emerge victorious. Play begins at the dawn of history in 4,000 BCE.

Freeciv21 takes its roots in the well-known FOSS game Freeciv and extends it for more fun, with a revived focus on competitive multiplayer environments. Players can choose from over 500 nations and can play against the computer or other people in an active online community.

The code is maintained by the team over at Longturn.net and is based on the QT framework. The game supports both hex and square tiles and is easily modified to create custom rules.

This is a program to read and install a modpack URL from the command line. If you are looking for a GUI based modpack installer, refer to freeciv21-modpack-qt(6).

OPTIONS

The following options are accepted on the command line of the modpack installer. They may not be combined as with other tools. For example: `freeciv21-modpack -Lp https://raw.githubusercontent.com/longturn/modpacks/main/index.json ~/.local/fc21` will not work. Instead you will need to enter each option separately, such as, `freeciv21-modpack -L https://raw.githubusercontent.com/longturn/modpacks/main/index.json -p ~/.local/fc21`.

-d, --debug <LEVEL>

Set debug log level (fatal/critical/warning/info/debug). Default log level is `info`.

-i, --install <URL>

Automatically install modpack from a given URL.

-L, --List <URL>

Load modpack list from given URL.

-p, --prefix <DIR>

Install modpacks to given directory hierarchy. Default location is `~/.local/share/freeciv21`.

-h, --help

Display help on command line options.

--help-all

Display help including Qt specific options.

-v, --version

Display version information.

USER

Specifies the username of the current user.

BUGS

Please report bugs to the Freeciv21 bug tracker at <https://github.com/longturn/freeciv21/issues/new/choose>

MORE INFORMATION

See the Longturn home page at <https://longturn.net/>. You can also find the code repository at <https://github.com/longturn/freeciv21/>.

3.4.7 freeciv21-ruleup**SYNOPSIS**

```
freeciv21-ruleup [ -r|--ruleset <RULESET> ] [ -o|--output <DIRECTORY> ] [ -h|--help ] [ -v|--version ]
```

DESCRIPTION

Freeciv21 is a free open source turn-based empire-building 4x strategy game, in which each player becomes the leader of a civilization. You compete against several opponents to build cities and use them to support a military and an economy. Players strive to complete an empire that survives all encounters with its neighbors to emerge victorious. Play begins at the dawn of history in 4,000 BCE.

Freeciv21 takes its roots in the well-known FOSS game Freeciv and extends it for more fun, with a revived focus on competitive multiplayer environments. Players can choose from over 500 nations and can play against the computer or other people in an active online community.

The code is maintained by the team over at Longturn.net and is based on the QT framework. The game supports both hex and square tiles and is easily modified to create custom rules.

This command line utility allows a user to upgrade a ruleset designed for an older version of Freeciv21 to work on a newer version. freeciv21-ruleup does not create well-formatted human-readable ruleset files, so some hand editing will be needed to aid readability.

OPTIONS

The following options are accepted on the command line of the ruleup tool. They may not be combined as with other tools. For example: `freeciv21-ruleup -ro ~/myrulset ~/myupgradedruleset` will not work. Instead you will need to enter each option separately, such as, `freeciv21-ruleup -r ~/myruleset -o ~/myupgradedruleset`.

-F, --Fatal

Raise a signal on failed assertion. An assertion is a code calculation error. With this set, the client process will SEGFAULT instead of issuing a warning message to the terminal console.

-r, --ruleset

The path to the old ruleset that needs to be upgraded.

- o, --output**
The path to write the upgraded ruleset.
 - h, --help**
Display help on command line options.
 - help-all**
Display help including Qt specific options.
 - v, --version**
Display version information.
- USER**
Specifies the username of the current user.

BUGS

Please report bugs to the Freeciv21 bug tracker at <https://github.com/longturn/freeciv21/issues/new/choose>

MORE INFORMATION

See the Longturn home page at <https://longturn.net/>. You can also find the code repository at <https://github.com/longturn/freeciv21/>.

3.5 Using the Modpack Installer Utility

The modpack installer is a simple tool to download custom Freeciv21 content called **modpacks** from the Internet and automatically install them into the correct location to be used by Freeciv21.

A **modpack** can consist of one or more of:

- **rulesets** for game rules
- **scenarios** for game maps (with or without players/cities/etc)
- **tilesets** for game graphics
- **soundsets** or **musicsets** for sound effects and in-game music

They can be installed using the modpack installer tool `freeciv21-modpack-qt` that comes with the standard installation. There is also a command-line-only tool called `freeciv21-modpack`, which is mainly useful for headless game servers.

The easiest way to open the modpack installer is by clicking the *Mods* button on the *Start Screen* or the *Add Modpacks* option from the *Game* menu.

In standard Freeciv21 builds, when you start the graphical modpack installer, it will show a list of modpacks curated by the Freeciv21 developers and Longturn community. You can select one and click *Install modpack*. The tool will download the files for the selected modpack, any other modpacks it depends on, and install them ready for the main Freeciv21 programs to use.

You can also point the installer at modpacks or lists on other servers:

- If someone has given you an individual modpack URL (ending in `.json`), you can paste it into the *Modpack URL* box, and when you click *Install modpack*, the modpack will immediately downloaded and install.

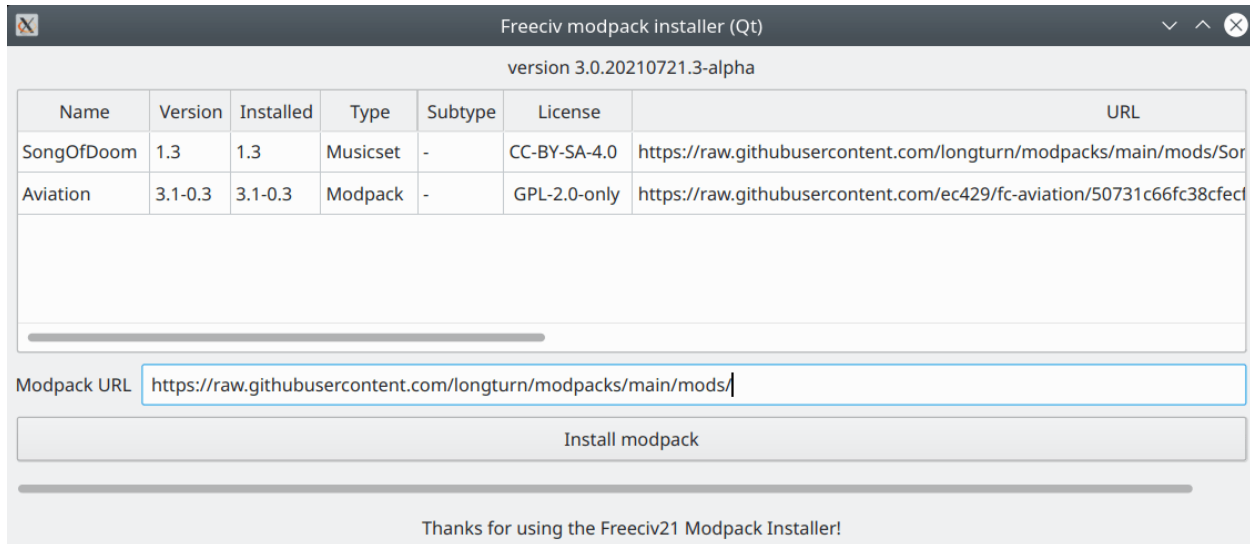


Fig. 3.57: Freeciv21 Modpack Installer GUI application

Note: If you have been given a URL ending up `.modpack` or `.mpd1`, that is probably for legacy Freeciv and not Freeciv21. These modpack files are not compatible with Freeciv21.

- If someone has given you a URL for a list of modpacks (also ending in `.json`) and you want to browse them with the standard Freeciv21 modpack installer build, you need to start the tool with that URL on its command line, for instance:

```
$ freeciv21-modpack --List https://example.com/3.1/my-modpacks.json
```

Note: The capital letter in `--List`

The tool has some other command-line options, but most users will not need to use them. Use `--help` for a list of them.

Once you have installed a modpack, how you use it depends on the modpack type:

- Scenarios (pre-defined maps) should be listed under *Start scenario game* from the [game start page](#), or from the game server prompt via `/list scenarios`.

Tip: For network play, scenarios need only be installed on the game server.

- Rulesets should appear on the *Ruleset* drop-down from the game's *Start new game* page as shown in [Fig. 3.2](#) in the [Game Manual](#). On the game server, you can load a ruleset with `/read <name>` or failing that perhaps `/rulesetdir <name>`.
- Tilesets should appear for selection in the local interface options, in the appropriate topology-specific *Tileset* drop-down under *Graphics*.

Note: Tilesets should be installed on the computer with the game interface.

- Soundsets and musicsets should appear in the dropdowns on the *Sound* page of the interface options.

With standard Freeciv21 builds, modpacks get installed into a per-user area and not into the main Freeciv21 installation. So you should not need any special permissions to download them. If you uninstall the Freeciv21 game any modpacks you downloaded are likely to remain on your system. Conversely, if you delete downloaded modpacks by hand, the standard rulesets, tilesets, etc. supplied with Freeciv21 will not be touched.

The precise location where files are downloaded to depends on your build and platform. For Unix systems, it is likely to be under the hidden `~/.local/share/freeciv21` directory in your home directory. For Windows based systems it will be in your user profile directory in a hidden `AppData` folder, typically, `C:\Users\[MyUserName]\AppData\Roaming\freeciv21`. It is likely to be near where the Freeciv21 interface stores its saved games.

Most modpacks are specific to a particular major version of Freeciv21; for instance, while a 3.0 ruleset or tileset can be used with all Freeciv21 3.0.x releases, it cannot be used as-is with any 3.1.x release. So, most modpacks are installed in a specific directory for the major version, such as `~/.local/share/freeciv21/3.1/` on Unix.

Note: The modpack installer displays which version it will install for at the top of its window.

An exception to this is scenario maps. Scenarios created for one version of Freeciv21 can usually be loaded in later versions, so they are installed in a version-independent location (typically `~/.local/share/freeciv21/scenarios/` on Unix).

Once a modpack is installed, there is no uninstall action, and if you remove the files by hand, the installer will still consider the modpack to be installed. The installer maintains its own database (`.control/modpacks.db`) listing which modpack versions are installed, but does not keep track of which files were installed by which modpack. If the database gets out of sync with reality (or is deleted), it's harmless for already installed modpacks and the main Freeciv21 programs (which do not consult the database), but can confuse the modpack installer's dependency tracking later.

Modpacks consist mostly of data files read by the Freeciv21 engine. They do not contain compiled binary code and are thus platform-independent. Rulesets can contain code in the form of Lua scripts, but this is executed in a sandbox to prevent obvious security exploits. Modpacks are installed to a specific area and cannot overwrite arbitrary files on your system. Nevertheless, you should only install modpacks from sources you trust.

3.6 Common Ruleset Reference Pages

3.6.1 Building Flags

Every building (e.g. city improvement) in a ruleset can be given zero or more *flags* by ruleset *modders*. Each flag gives additional features to the building that has been assigned the flag. A reader can find the flags by looking at the `buildings.ruleset` file in any ruleset directory. The following flags are used in all of the rulesets shipped by the Freeciv21 developers.

VisibleByOthers

Anyone who can see your city knows whether it has this improvement. Great and small wonders are always visible. City Walls are often visible by others as well.

SaveSmallWonder

If you lose the city with this building in it, and the `savepalace` server setting is enabled, another Palace will be built for free in a random city. This flag should only be used with building genus `SmallWonder`.

Gold

This is not a real building, per se. The city's shield production is turned into gold indefinitely. Most rulesets call this Capitalization or Coinage. The building's genus should be `Special`.

DisasterProof

Natural disasters never destroy a building with this flag. This flag is important for building genus Improvement as others are automatically disaster proof.

3.6.2 Technology Advance Flags

Every technology advance in a ruleset can be given zero or more *flags* by ruleset *modders*. Each flag gives additional features to the technology advance that has been assigned the flag. A reader can find the flags by looking at the `techs.ruleset` file in any ruleset directory. The following flags are used in all of the rulesets shipped by the Freeciv21 developers.

Bonus_Tech

The player gets extra technology when reaching the technology with this flag first.

Bridge

Settler unit types can build roads with the RequiresBridge flag over roads with the PreventsOtherRoads flag. This is typically on Rivers.

Build_Airborne

This flag is mostly to help the AI (e.g. an AI hint). From now on the player can build air units.

Claim_Ocean

With this flag, the player can claim ocean tiles even if they are not adjacent to a border source.

Claim_Ocean_Limited

With this flag, the player's oceanic border sources claim ocean tiles even if they are not adjacent to a border source.

3.6.3 Unit Class Flags

Unit types (e.g. a unit such as a Phalanx) are associated or grouped together in classes. Every unit type is associated with a single unit class. Each class can have zero or more *flags* assigned to it by ruleset *modders*. The flags give additional features to all of the unit types associated with the class. A reader can find the unit type flags by looking at the `units.ruleset` file in any ruleset directory. The following flags are used in all of the rulesets shipped by the Freeciv21 developers.

Airliftable

Allows a unit to be airlifted from a suitable city. This quite often requires an Airport.

AttackNonNative

Allows a native tiled unit to attack a non-native tiled unit (e.g. Cannon attacks a Galleon from inside a city). The Unit Type Flag `Only_Native_Attack` can override this.

AttFromNonNative

Allows a unit to attack another unit from non-native tiles (e.g. Marines in a Transport attacking a city). The Unit Type Flag `Only_Native_Attack` can override this.

BuildAnywhere

Allows a unit to be built even in the middle of non-native terrain.

CanFortify

Allows a Unit to fortify inside and outside of cities, gaining a defense bonus.

CanOccupyCity

Allows a unit to enter a city and take ownership of it.

CanPillage

Allows a unit to pillage a tile's infrastructure improvements. This can be your own tile or that of another nation.

CollectRansom

Allows a unit to collect a ransom in gold after killing a Barbarian Leader.

DamageSlows

Unit classes with this flag are slowed down (have less than 100% of expected movement points) when damaged.

DoesntOccupyTile

These kinds of units do not fully occupy the tile they are on. If the unit is of an enemy nation, your city's citizens can still work that tile. Unit classes without this flag fully occupy the tile and prevent citizen workers from utilizing the tile's capabilities.

KillCitizen

A unit class with this flag will kill city citizens during attack. If too many citizens are killed as part of taking a city, the player may end up with ruins instead of a city.

Missile

As the flag name implies, these are missiles.

TerrainDefense

Unit classes with this flag give unit types a defense bonus from certain terrain.

TerrainSpeed

Units use terrain specific speed.

Unreachable

Unit can be attacked only by units explicitly listing this class in its *targets*, unless on a city or native base. For class members which are transports, cargo cannot load/unload except in a city or native base, unless that unit explicitly lists this class in its *embarks/disembarks* actions.

ZOC

Unit is subject to zone of control rules.

3.6.4 Unit Type Flags

Every unit type (e.g. a unit such as a Phalanx) in a ruleset can be given zero or more *flags* by ruleset *modders*. Each flag gives additional features to the unit type that has been assigned the flag. A reader can find the flags by looking at the `units.ruleset` file in any ruleset directory. The following flags are used in all of the rulesets shipped by the Freeciv21 developers.

AirAttacker

Very bad at attacking AEGIS Cruiser.

Airbase

This flag does not have a pre-set use. It is available for ruleset modders to add as a condition to an *action enabler requirements vector*.

AddToCity

Citizen containing units (e.g. Settlers) can add population to a city. The unit must travel to the target city.

BadCityDefender

If attacked while in a city, firepower is set to 1 and firepower of attacker is doubled.

BadWallAttacker

The firepower of this unit is set to 1 if attacking a city defended by a City Walls, or other city building defense.

BarbarianOnly

Only Barbarians can build this unit.

Bombarder

Can bombard cities.

CanEscape

This unit has, given that certain conditions are fulfilled, a 50% chance to escape rather than being killed when `killstack` is enabled and the defender of its tile is defeated. The conditions are that it has more move points than required to move to an adjacent tile plus the attacker's move points and that the attacker does not have the `CanKillEscaping` unit type flag.

CanKillEscaping

An attack from this unit ignores the `CanEscape` unit type flag.

Cant_Fortify

Unit cannot fortify inside or outside of cities.

Capturable

Can be captured by some enemy units.

Capturer

Can capture some enemy units. See `Capturable`.

Cities

Can found new cities.

CityBuster

This unit has double firepower against cities.

Coast

Sea Class only. Can refuel on coast. Set fuel to force unit to regularly end turn on coast.

CoastStrict

Sea Class only. Cannot leave the coast. Trireme units often have this flag.

Consensus

Undisbandable when your bureaucracy has a veto.

Diplomat

Can defend against diplomat actions.

EvacuateFirst

The game will try to rescue units with this flag before it tries to rescue units without it when their transport is destroyed. Think of the Birkenhead drill ("women and children first"). Replace "women and children" with "units with the `EvacuateFirst` unit type flag".

Fanatic

Can only be built by governments that allow them. See `civ2/governments.ruleset`, Fundamentalism government for an example.

FieldUnit

Cause unhappiness even when not being aggressive.

GameLoss

Losing one of these units means you lose the game, but it is produced without homecity and upkeep

HasNoZOC

Unit has no Zone of Control (ZOC), thus any unit can move around it freely.

Helicopter

Defends very badly against Fighters.

HelpWonder

Can help to build a wonder. The unit must travel to target city. These are typically Caravan and Freight units.

Horse

Attack value reduced when attacking Pikemen.

IgTer

Use constant move cost defined in the `igter_cost` variable in `terrain.ruleset`, rather than `terrain/road` etc. cost, unless terrain cost is less.

IgZOC

Ignores unit zone of control (ZOC), even if the unit class has the ZOC flag.

Infra

Can build infrastructure.

Marines

Can launch attack from non-native tiles.

NeverProtects

Does not protect reachable units on its tile from enemy attackers, even if the `unreachableprotects` server setting is enabled and the unit class is `unreachable`.

NewCityGamesOnly

Unit cannot be built on scenarios where founding new cities is not allowed. Give this flag to units that would make no sense to have in a game with such a scenario.

NoBuild

This unit cannot be built.

NoHome

This unit has no homecity and will be free of all upkeep, and therefore will not revolt along with its city of origin should it be incited.

NonMil

A non-military unit: no attacks, no martial law, and can enter peaceful borders. See `DoesntOccupyTile`.

NoVeteran

This unit cannot gain veteran levels through experience, as if both `base_raise_chance` and `work_raise_chance` were zero.

Nuclear

Is nuclear capable.

NuclearOP

Over Powered Nuclear. See `sandbox/units.ruleset`.

OneAttack

Only attacks once and is destroyed/consumed as part of the attacking action.

Only_Native_Attack

Cannot attack targets on non-native tiles even if the unit class can.

Paratroopers

Can be paratrooped from a friendly city or suitable base.

Provoking

A unit considering to auto attack this unit will choose to do so even if has better odds when defending against it then when attacking it. Applies when the `autoattack` server setting is enabled.

RealDiplomat

Can do real diplomat actions, unlike tech transfer units.

RealSpy

Can do real spy actions, unlike tech transfer units.

Settlers

Can irrigate and build roads.

Shield2Gold

Switch from shield upkeep to gold upkeep.

Spy

Strong in diplomatic battles. *Must* also be Diplomat.

Submarine

Attack value reduced when attacking Destroyer.

SuperSpy

This unit always wins diplomatic contests, that is, unless it encounters another SuperSpy, in which case the defender wins. Can also be used on non-diplomat units, in which case it can protect cities from diplomats. Also 100% spy survival chance.

TradeRoute

Can establish trade routes. Must travel to target city.

Transform

Can transform terrain.

Unbriable

Unit cannot be bribed.

Unique

A player can only have one of these units in the game at the same time. Barbarians cannot use this at present.

3.6.5 Unit Type Role Flags

Every unit type (e.g. a unit such as a Phalanx) in a ruleset can be given zero or more roles by ruleset *modders*. Each role gives additional features to the unit type that has been assigned the flag. A reader can find the roles by looking at the `units.ruleset` file in any ruleset directory. The following roles are used in all of the rulesets shipped by the Freeciv21 developers.

AttackFastStartUnit

Can be designated as an attacking unit at game start.

AttackStrongStartUnit

Can be designated as a strong attacking unit at game start.

Barbarian

Can be created as land Barbarian.

BarbarianTech

Can be created as land Barbarian, if someone has researched its technology requirements.

BarbarianBoat

Can be created as boat for sea Barbarian.

BarbarianBuild

Can be built by barbarians.

BarbarianBuildTech

Can be built by barbarians if someone has researched its technology requirements.

BarbarianLeader

This unit is the Barbarian Leader. Only one can be defined once.

BarbarianSea

Can be created as a Barbarian that disembarks from a barbarian boat.

BarbarianSeaTech

Can be created as a Barbarian that disembarks from a barbarian boat if someone has researched its technology requirements.

BorderPolice

Can peacefully expel certain foreign units. See `Expellable`.

CitiesStartUnit

Can be designated as city founding unit at game start.

DefendGood

AI hint: Good for defending with.

DefendGoodStartUnit

Can be designated as an good defending unit at game start.

DefendOk

AI hint: Ok for defending with.

DefendOkStartUnit

Can be designated as an ok defending unit at game start.

DiplomatStartUnit

Can be designated as Diplomat unit at game start.

Expellable

Can be peacefully expelled from foreign tiles. See `BorderPolice`.

Explorer

This unit performs Explorer functions.

ExplorerStartUnit

Can be designated as an Explorer unit at game start.

Ferryboat

AI hint: Useful for ferrying.

FerryStartUnit

Can be designated as a ferry boat unit at game start.

FirstBuild

First to be built when a city is founded.

HeavyWeight

Airliftable from Airport after you learn Fusion Power.

Hut

Can be found in a hut.

HutTech

Can be found in a hut.

Partisan

Can be created as a Partisan. Only one unit can have this flag.

LightWeight

Airliftable once you learn *Flight*.

MediumWeight

Airliftable from Airport after you learn Advanced Flight.

Settlers

Acts as a Settler type unit.

WorkerStartUnit

Can be designated as a Worker unit at game start.

MODDING

The Modding category is an area for documentation editors to provide tips and other details on modifying aspects of Freeciv21 such as Rulesets, Musicsets, Soundsets and Tilesets. All of these areas allow for a large amount of variability in game play that is not hardcoded in the software. This is one of the strengths of Freeciv21.

There are five major areas of Modding that are often called “Modpacks” and are written by “Modders”. The sections below describe these major Modding areas.

All of the Modpacks are written in plain text files, except for Scenarios. The plain text files resemble `ini` files in format and are known as `spec` files to Freeciv21. Scenarios are specially designed saved games that have been edited for a specific purpose. Examples include: Earth Map, Europe 1900, Americas Map, etc.

4.1 Rulesets

Rulesets are a collection of `spec` files that fully define a game’s rules. Rulesets are broken down into 12 files. The `spec` files for rulesets are defined as follows:

- `game.serv`
- `game/buildings.ruleset`
- `game/cities.ruleset`
- `game/effects.ruleset`
- `game/game.ruleset`
- `game/governments.ruleset`
- `game/nations.ruleset`
- `game/script.lua`
- `game/styles.ruleset`
- `game/techs.ruleset`
- `game/terrain.ruleset`
- `game/units.ruleset`

Have a look at `civ2civ3.serv` and associated files in `/civ2civ3` for an example.

Refer to Ruleset specific documents:

4.1.1 Rulesets Overview

Quickstart

Rulesets allow modifiable sets of data for units, advances, terrain, improvements, wonders, nations, cities, governments, and miscellaneous game rules, without requiring recompilation, in a way which is consistent across a network and through savegames.

- To play Freeciv21 normally: do not do anything special. The new features all have defaults which give the standard Freeciv21 behavior.
- To play a game with rules more like Civ1, start the server with:

```
$ freeciv21-server -r data/civ1.serv
```

and any other command-line arguments you normally use, depending on how you have Freeciv21 installed you may have to give the installed data directory path instead of `data`.

Start the game normally. The game must be network-compatible to the server (usually meaning the same or similar version), but otherwise nothing special is needed. However, some third-party rulesets may potentially require special graphics to work properly, in which case the game should have those graphics available and be started with an appropriate `--tiles` argument.

Note that the Freeciv21 *AI* might not play as well with rules other than standard Freeciv21 *Classic* or *Civ2Civ3* rulesets. The *AI* is supposed to understand and utilize all sane rules variations, so please report any *AI* failures so that they can be fixed.

The rest of this file contains:

- More detailed information on creating and using custom/mixed rulesets.
- Information on implementation, and notes for further development.

Using and Modifying Rulesets

Rulesets are specified using the server command `rulesetdir`. The command line example given above just reads a file which uses this command (as well as a few of the standard server options). The server command specifies in which directory the ruleset files are to be found.

The ruleset files in the data directory are user-editable, so you can modify them to create modified or custom rulesets (without having to recompile Freeciv21). It is suggested that you *do not* edit the existing files in the “civ2civ3”, “classic”, “experimental”, “multiplayer”, “civ1”, or “civ2” directories, but rather copy them to another directory and edit the copies. This is so that its clear when you are using modified rules and not the standard ones.

The format used in the ruleset files should be fairly self-explanatory. A few points:

- The files are not all independent, since e.g., units depend on advances specified in the techs file.
- Units have a field, “roles”, which is like “flags”, but determines which units are used in various circumstances of the game (rather than intrinsic properties of the unit). See comments in `common/unit.h`
- Rulesets must be in UTF-8. Translatable texts should be American English ASCII.

Restrictions and Limitations

- `units.ruleset` :

Restrictions:

- At least one unit with role “FirstBuild” must be available from the start (i.e., `tech_req` = “None”).
- There must be units for these roles:
 - “Explorer”
 - “FerryBoat” (Must be able to move at sea)
 - “Hut” (Must be able to move on land)
 - “Barbarian” (Must be able to move on land)
 - “BarbarianLeader” (Must be able to move on land)
 - “BarbarianBuild” (Must be able to move on land)
 - “BarbarianBoat” (Must be able to move at sea)
 - “BarbarianSea”

- `nations.ruleset` :

Restrictions:

- Government used during revolution cannot be used as `default_government` or `init_government` for any nation.

Implementation Details

This section and following section will be mainly of interest to developers who are familiar with the Freeciv21 source code.

Rulesets are mainly implemented in the server. The server reads the files and then sends information to the game. Rulesets are used to fill in the basic data tables on units etc., but in some cases some extra information is required.

For units and advances, all information regarding each unit or advance is now captured in the data tables, and these are now “fully customizable”, with the old enumeration types completely removed.

Game Settings Defined In The Ruleset

Game settings can be defined in the section `[settings]` of the file `game.ruleset`. The name key is equal to the setting name as listed by ‘show all’. If the setting should be locked by the ruleset, the last column should be set to `TRUE`.

```
set =
{ "name", "value", "lock"
  "bool_set", TRUE, FALSE
  "int_set", 123, FALSE
  "str_set", "test", FALSE
}
```

Scenario Capabilities

Some scenarios can be unlocked from a ruleset, meaning that they are not meant to be used with strictly one ruleset only. To control that such a scenario file and a ruleset are compatible, capabilities are used. Scenario file lists capabilities it requires from the ruleset, and ruleset lists capabilities it provides.

Some standard capabilities are:

- `std-terrains`: Ruleset provides at least terrain types `Inaccessible`, `Lake`, `Ocean`, `Deep Ocean`, `Glacier`, `Desert`, `Forest`, `Grassland`, `Hills`, `Jungle`, `Mountains`, `Plains`, `Swamp`, `Tundra`. Ruleset provides `River` extra

4.1.2 Effects

The `effects.ruleset` file contains all effects in play in a Freeciv21 ruleset. They have the following form (this is perhaps the most complicated example we could find):

```
[effect_hydro_plant]
type = "Output_Bonus"
value = 25
reqs =
{ "type", "name", "range", "present", "quiet"
  "Building", "Factory", "City", TRUE, FALSE
  "Building", "Hydro Plant", "City", TRUE, FALSE
  "OutputType", "Shield", "Local", TRUE, TRUE
  "Building", "Hoover Dam", "Player", FALSE, FALSE
  "Building", "Nuclear Plant", "City", FALSE, FALSE
}
```

The text in the brackets is the entry name, which just has to be unique, but is otherwise not used. The `type` field tells Freeciv21 which effect you are defining. The `value` is the effect's value, which depends on which effect it is. The `reqs` table contain a list of requirements for this effect being in effect. You need to satisfy all requirements listed here for this effect to take effect in the game. Requirements with `present = TRUE` must be present, those with `present = FALSE` must not be present.

Value is integral b parameter for many effects (must be in the range -32767 to 32767).

Requirement range may be one of: `None`, `Local`, `CAdjacent` (Cardinally Adjacent), `Adjacent`, `City`, `Continent`, `Player`, `Allied`, `World`. Some requirement types may only work at certain ranges. This is described below. In particular, at present, `Continent` effects can affect only cities and units in cities.

A requirement may have a `survives` field, and if this `TRUE`, the effect survives destruction. This is supported for only a few conditions and ranges: wonders (at world or player range), nations, and advances (both at world range only).

A requirement may have a `present` field, and if this is `FALSE`, the requirement is negated (the condition must not be true for the req to be met).

A requirement may have a `quiet` field, and if this is `TRUE`, the help system does not try to autogenerate text about that requirement. This can be used if the help system's text is unclear or misleading, or if you want to describe the requirement in your own words. The `quiet` field has no effect on the game rules.

Requirement types and supported ranges

Table 4.1: Effect Flags

Requirement	Supported ranges
Tech	World, Alliance, Team, Player
TechFlag	World, Alliance, Team, Player
MinTechs	World, Player
Achievement	World, Alliance, Team, Player
Gov	Player
Building	World, Alliance, Team, Player, Continent, Traderoute, City, Local
BuildingGenus	Local
Extra	Local, Adjacent, CAdjacent, Traderoute, City
BaseFlag	Local, Adjacent, CAdjacent, Traderoute, City
RoadFlag	Local, Adjacent, CAdjacent, Traderoute, City
ExtraFlag	Local, Adjacent, CAdjacent, Traderoute, City
Terrain	Local, Adjacent, CAdjacent, Traderoute, City
Good	City
UnitType	Local
UnitFlag	Local
UnitClass	Local
UnitClassFlag	Local
Nation	World, Alliance, Team, Player
NationGroup	World, Alliance, Team, Player
Nationality	Traderoute, City
DiplRel	World, Alliance, Team, Player, Local
Action	Local
OutputType	Local
Specialist	Local
MinYear	World
MinCalFrag	World
Topology	World
ServerSetting	World
Age (of unit)	Local
Age (of city)	City
Age (of player)	Player
MinSize	Traderoute, City
MinCulture	World, Alliance, Team, Player, Traderoute, City
MinForeignPct	Traderoute, City
AI	Player
MaxUnitsOnTile	Local, Adjacent, CAdjacent
TerrainClass	Local, Adjacent, CAdjacent, Traderoute, City
TerrainFlag	Local, Adjacent, CAdjacent, Traderoute, City
TerrainAlter	Local
CityTile	Local, Adjacent, CAdjacent
CityStatus	Traderoute, City
Style	Player
UnitState	Local
Activity	Local
MinMoveFrag	Local
MinVeteran	Local
MinHitPoints	Local

continues on next page

Table 4.1 – continued from previous page

Requirement	Supported ranges
VisionLayer	Local
NationalIntelligence	Local

- `MinSize` is the minimum size of a city required.
- `AI` is the `AI` player difficulty level.
- `TerrainClass` is either “Land” or “Oceanic”.
- `CityTile` is either “Center” (city center) or “Claimed” (owned).
- `CityStatus` is “OwnedByOriginal”.
- `DiplRel` is a diplomatic relationship as shown in *nations view*.
- `MaxUnitsOnTile` is about the number of units present on a tile.
- `UnitState` is “Transported”, “Transporting”, “OnNativeTile”, “OnLivableTile”, “InNativeExtra”, “OnDomesticTile”, “MovedThisTurn”, or “HasHomeCity”.
- `Activity` is “Idle”, “Pollution”, “Mine”, “Irrigate”, “Fortified”, “Fortress”, “Sentry”, “Pillage”, “Goto”, “Explore”, “Transform”, “Fortifying”, “Fallout”, “Base”, “Road”, “Convert”, “Cultivate”, or “Plant”.
- `MinMoveFrag` is the minimum move fragments (*MP*) the unit must have left.
- `MinCalFrag` is the minimum sub-year division the calendar must have reached, if enabled (see `[calendar].fragments` in `game.ruleset`).
- `Nationality` is fulfilled by any citizens of the given nationality present in the city.
- `ServerSetting` is if a Boolean server setting is enabled. The setting must be visible to all players and affect the game rules.

Details about requirement types

The “DiplRel” requirement type

Look for the diplomatic relationship “Never met”, “War”, “Cease-fire”, “Armistice”, “Peace”, “Alliance”, “Team”, “Gives shared vision”, “Receives shared vision”, “Hosts embassy”, “Has embassy”, “Hosts real embassy” (not from an effect), “Has real embassy”, “Has Casus Belli” (reason for war), “Provided Casus Belli” or “Is foreign”.

A `DiplRel` is considered fulfilled for the range:

- “world” if some player in the world has the specified diplomatic relationship to some other living player.
- “player” if the player has the specified diplomatic relationship to some other living player.
- “local” if the first player has the specified relationship to the second player. Example: When testing a build requirement for an extra the first player is the owner of the unit and the second player the owner of the terrain the extra is built on.

Only the exact relationship required fulfills it. Example: An “Alliance” or an “Armistice” agreement will not fulfill a “Peace” requirement.

It is possible to create a requirement that in some situations will not have a player to check. In those cases the requirement will always be considered unfulfilled. This applies to both present and not present requirements. The ranges “Alliance”, “Team”, “Player” and “Local” needs a player. The Local range also needs the player the first player’s relationship is to.

Example: The requirements below are about the relationship to the owner of a tile. The table shows in what situations a requirement is fulfilled.

Table 4.2: DiplRel Example

Requirement	Fulfilled when the tile is		
	Domestic	Unclaimed	Foreign
"DiplRel", "Is foreign", "Local", TRUE	no	no	yes
"DiplRel", "Is foreign", "Local", FALSE	yes	no	no

The “MaxUnitsOnTile” requirement type

Check the number of units present on a tile. Is TRUE if no more than the specified number of units are present on a single tile.

Tip: By using negation (“not present”) it is possible to check if a tile has more than the given numbers. It is possible to combine a negated and a non negated requirement to specify a range.

The “UnitState” requirement type

Transported

is fulfilled if the unit is transported by another unit.

Transporting

is fulfilled if the unit is transporting another unit.

OnNativeTile

is fulfilled if the unit is on a tile with native terrain or with a native Extra. Does not care about details like cities and safe tiles.

OnLivableTile

is fulfilled if the unit is on a tile where it can exist outside of a transport.

InNativeExtra

is fulfilled if the unit is on a tile with an extra native to it.

OnDomesticTile

is fulfilled if the unit is on a tile owned by its player.

MovedThisTurn

is fulfilled if the unit has moved this turn.

HasHomeCity

is fulfilled if the unit has a home city.

The “NationalIntelligence” requirement type

This is only used with the *Nation_Intelligence effect*.

Effect types

Tech_Parasite

Gain any advance known already by AMOUNT number of other teams, if `team_pooled_research` is enabled, or AMOUNT number of other players otherwise.

Note: If you have two such effects, they add up (the number of players required to gain an advance is increased).

Airlift

Allow airlift to/from a city. The value tells how many units per turn can be airlifted, unless server setting `airlifttingstyle` sets the number unlimited for either source or destination city. If airlifts are set to unlimited, they are enabled by any positive value of this effect.

Any_Government

Allow changing to any form of government regardless of tech prerequisites.

Capital_City

The city with a positive value is a capital city. Player’s city with highest `Capital_City` value (or random among those with equal positive value) is the primary capital. Cities with lesser positive value are secondary capitals.

Gov_Center

The city with this effect is governmental center. Corruption and waste depends on distance to nearest such city.

Enable_Nuke

Allows the production of nuclear weapons.

Enable_Space

Allows the production of space components.

Specialist_Output

Specify what outputs a specialist is producing. Should be used with an `OutputType` requirement.

Output_Bonus

City production is increased by AMOUNT percent.

Output_Bonus_2

City production is increased by AMOUNT percent after *Output_Bonus*, so is multiplicative with it.

Output_Add_Tile

Add AMOUNT to each worked tile.

Output_Inc_Tile

Add AMOUNT to each worked tile that already has at least 1 output.

Output_Per_Tile

Increase tile output by AMOUNT percent.

Output_Tile_Punish_Pct

Reduce the output of a tile by AMOUNT percent. The number of units to remove is rounded down. Applied after everything except a city center’s minimal output.

Output_Waste_Pct

Reduce waste by AMOUNT percent.

Force_Content

Make AMOUNT unhappy citizens content. Applied after martial law and unit penalties.

Give_Imm_Tech

Give AMOUNT techs immediately.

Conquest_Tech_Pct

Percent chance that a player conquering a city learns a tech from the former owner.

Growth_Food

Saves some food in the granary when a city grows (or shrinks). This effect controls how much food there will be in the city's granary after growing, as a percentage of the foodbox at the new size, provided there was sufficient food before growing. This also reduces the aqueductloss penalty in the same fraction.

Note: This is traditionally used for the Granary.

Growth_Surplus_Pct

New in version 3.1.

How much of the excess food is kept when a city growth, as a percentage. For example, with a value of 100, a city with a granary full at 18/20 and generating 5 food would start the next turn with 3 bushels in its granary. With a value of 50, it would have only 1 bushel.

Have_Contact

If value > 0, gives contact to all the other players.

Have_Embassies

If value > 0, gives an embassy with all the other players owner has ever had contact with.

Irrigation_Pct

The tile gets value % of its terrain's Irrigation_Food_Incr bonus value.

Note: This is how irrigation-like extras have an effect.

Mining_Pct

The tile gets value % of its terrain's Mining_Shield_Incr bonus.

Note: This is how mine-like extras have an effect.

Make_Content

Make AMOUNT unhappy citizens content. Applied before martial law and unit penalties.

Make_Content_Mil

Make AMOUNT unhappy citizens caused by units outside of a city content.

Make_Content_Mil_Per

Make AMOUNT per unit of unhappy citizens caused by units outside of a city content.

Make_Happy

Make AMOUNT citizens happy.

Enemy_Citizen_Unhappy_Pct

There will be one extra unhappy citizen for each value/100 citizens of enemy nationality in the city.

No_Anarchy

No period of anarchy between government changes.

Note: This also neuters the *Has_Senate* effect.

Nuke_Proof

City is nuke proof.

Pollu_Pop_Pct

Increases pollution caused by each unit of population by AMOUNT percent (adds to baseline of 100%, i.e. 1 pollution per citizen).

Pollu_Pop_Pct_2

Increases pollution caused by each unit of population by AMOUNT percent (adds to baseline of 100%, i.e. 1 pollution per citizen). This factor is applied after *Pollu_Pop_Pct*, so is multiplicative with it.

Pollu_Prod_Pct

Increases pollution caused by shields by AMOUNT percent.

Health_Pct

Reduces possibility of illness (plague) in a city by AMOUNT percent.

Reveal_Cities

Immediately make all cities known.

Reveal_Map

Immediately make entire map known.

Border_Vision

Give vision on all tiles within the player's borders. Happens during turn change. Does nothing unless the borders setting is set to "Enabled". You can lock it if border vision rules are important to your ruleset.

Incite_Cost_Pct

Increases revolt cost by AMOUNT percent.

Unit_Bribe_Cost_Pct

Increases unit bribe cost by AMOUNT percent. Requirements are from the point of view of the target unit, not the briber.

Max_Stolen_Gold_Pm

The upper limit on the permille of the players gold that may be stolen by a unit doing the *Steal Gold* and the *Steal Gold Escape* actions. Evaluated against the city stolen from.

Thiefs_Share_Pm

The permille of the gold stolen by a unit doing the *Steal Gold* and the *Steal Gold Escape* actions that is lost before it reaches the player ordering it. Evaluated against the actor unit.

Maps_Stolen_Pct

The percent probability that the map of a tile is stolen in the actions *Steal Maps* and *Steal Maps Escape*. DiplRel reqs are unit owner to city owner. Requirements evaluated against tile or city not supported. Default value: 100%

Illegal_Action_Move_Cost

The number of move fragments lost when the player tries to do an action that turns out to be illegal. Only applied when the player was not aware that the action was illegal and its illegality therefore reveals new information.

Illegal_Action_HP_Cost

The number of hit points (*HP*) lost when the player tries to do an action that turns out to be illegal. Only applied when the player was not aware that the action was illegal and its illegality therefore reveals new information. Can kill the unit. If the action always causes the actor unit to end up at the target tile two consolation prizes are given. An area with the radius of the actor unit's *vision_radius_sq* tiles is revealed. The player may also get contact with the owners of units and cites adjacent to the target tile.

Action_Success_Actor_Move_Cost

The number of move fragments lost when a unit successfully performs an action. Evaluated and done after the action is successfully completed. Added on top of any movement fragments the action itself subtracts.

Action_Success_Target_Move_Cost

The number of move fragments subtracted from a unit when someone successfully performs an action on it. Evaluated and done after the action is successfully completed. Added on top of any movement fragments the action itself subtracts. Only supported for actions that targets an individual unit.

Casus_Belli_Caught

Checked when a player is caught trying to do an action. Will cause an incident with the intended victim player if the value is 1 or higher. The incident gives the intended victim a casus belli against the actor player. A value of 1000 or higher is international outrage. International outrage gives every other player a casus belli against the actor.

Casus_Belli_Success

Checked when a player does an action to another player. Will cause an incident with the intended victim player if the value is 1 or higher. The incident gives the intended victim a casus belli against the actor player. A value of 1000 or higher is international outrage. International outrage gives every other player a casus belli against the actor.

Casus_Belli_Complete

Checked when a player completes an action that takes several turns against another player. Will cause an incident with the intended victim player if the value is 1 or higher. The incident gives the intended victim a casus belli against the actor player. A value of 1000 or higher is international outrage. International outrage gives every other player a casus belli against the actor. Only *Pillage* is currently supported.

Action_Odds_Pct

Modifies the odds of an action being successful. Some actions have a risk: the actor may get caught before he can perform it. This effect modifies the actor's odds. A positive value helps him. A negative value makes it more probable that he will get caught. Currently supports the actions *Incite City*, *Incite City Escape*, *Steal Gold*, *Steal Gold Escape*, *Steal Maps*, *Steal Maps Escape*, *Suitcase Nuke*, *Suitcase Nuke Escape*, *Sabotage City*, *Sabotage City Escape*, *Targeted Sabotage City*, *Targeted Sabotage City Escape*, *Sabotage City Production*, *Sabotage City Production Escape*, *Surgical Strike Building*, *Surgical Strike Production*, *Steal Tech*, *Steal Tech Escape Expected*, *Targeted Steal Tech*, *Targeted Steal Tech Escape Expected*, and *Spread Plague*.

Size_Adj

Increase maximum size of a city by AMOUNT.

Size_Unlimit

Make the size of a city unlimited.

Unit_Slots

Number of unit slots city can have units in. New units cannot be built, nor can homecity be changed so that maintained units would use more slots than this. Single unit does not necessarily use single slot - that is defined separately for each unit type.

SS_Structural, SS_Component, SS_Module

A part of a spaceship; this is a "Local" ranged effect. It (for now) applies to improvements which cannot be built unless *Enable_Space* is felt. Buildings which have this effect should probably not be given any other effects.

Spy_Resistant

In diplomatic combat defending diplomatic units will get an AMOUNT percent bonus. All *Spy_Resistant*'s are summed before being applied.

Building_Saboteur_Resistant

If a spy specifies a target for sabotage, then she has an AMOUNT percent chance to fail.

Stealings_Ignore

When determining how difficult it is to steal a tech from enemy, AMOUNT previous times tech has been stolen

from the city is ignored. Negative AMOUNT means that number of times tech has already been stolen from target city does not affect current attempt at all. With this effect it is possible to allow diplomats to steal tech multiple times from the same city, or make it easier for spies.

Move_Bonus

Add AMOUNT movement to units. Use “UnitClass” requirement with range of “Local” to give it a specific class of units only.

Unit_No_Lose_Pop

No population lost when a city’s defender is lost.

Unit_Recover

Units recover AMOUNT extra hitpoints (*HP*) per turn.

Upgrade_Unit

Upgrade AMOUNT obsolete units per turn.

Upkeep_Free

Improvements with AMOUNT or less upkeep cost become free to upkeep (others are unaffected).

Tech_Upkeep_Free

If this value is greater than 0, the tech upkeep is reduced by this value. For tech upkeep style “Basic” this is total reduction, for tech upkeep style “Cities” this reduction is applied to every city.

No_Unhappy

No citizens in the city are ever unhappy.

Veteran_Build

Increases the veteran class of newly created units of this type. The total AMOUNT determines the veteran class (clipped at the maximum for the unit).

Veteran_Combat

Increases the chance of units of this type becoming veteran after combat by AMOUNT percent.

Combat_Rounds

Maximum number of rounds combat lasts. Unit is the attacker. Zero and negative values mean that combat continues until either side dies.

HP_Regen

Units that do not move recover AMOUNT percentage (rounded up) of their full hitpoints (*HP*) per turn.

Note: This effect is added automatically to implement *HP* recovery in cities. This behavior can be turned off by requiring the +HP_Regen_Min option in `effects.ruleset`.

HP_Regen_Min

Lower limit on *HP_Regen*. That is, the recovery percentage is the larger of *HP_Regen* and HP_Regen_Min.

Note: This effect is added automatically to implement HP recovery in cities. This behavior can be turned off by requiring the +HP_Regen_Min option in `effects.ruleset`.

City_Vision_Radius_Sq

Increase city vision radius in squared distance by AMOUNT tiles.

Note: This effect is added automatically for VisionLayers other than Main, with a value of 2, and a Vision-Layer=Main requirement is added to any existing instances of this effect. This behaviour can be turned off by

requiring the `+VisionLayer` option in `effects.ruleset`, allowing you to use VisionLayer requirements to specify which layer (Main, Stealth or Subsurface) the effect applies to.

Unit_Vision_Radius_Sq

Increase unit vision radius in squared distance by AMOUNT tiles.

Note: A VisionLayer=Main requirement is added automatically to any existing instances of this effect. This behaviour can be turned off by requiring the `+VisionLayer` option in `effects.ruleset`, allowing you to use VisionLayer requirements to specify which layer (Main, Stealth or Subsurface) the effect applies to.

Defend_Bonus

Increases defensive bonuses of units. Any unit requirements on this effect will be applied to the `_attacking_` unit. Attackers with “BadWallAttacker” flag will have their firepower set to 1.

Attack_Bonus

Increases offensive bonuses of units. Unit requirements on this effect are the attacking unit itself.

Fortify_Defense_Bonus

Percentage defense bonus multiplicative with *Defend_Bonus*, usually given to *fortified* units. Unit requirements on this effect are the defending unit itself.

Gain_AI_Love

Gain AMOUNT points of “AI love” with *AI*’s.

Turn_Years

Year advances by AMOUNT each turn unless `Slow_Down_Timeline` causes it to be less.

Turn_Fragments

Year fragments advance by AMOUNT each turn.

Slow_Down_Timeline

Slow down the timeline based on the AMOUNT. If AMOUNT ≥ 3 the timeline will be max 1 year/turn; with AMOUNT $= 2$ it is max 2 years/turn; with AMOUNT $= 1$ it is max 5 years/turn; with AMOUNT ≤ 0 the timeline is unaffected. The effect will be ignored if `game.spacerace` isn’t set.

Civil_War_Chance

Base chance in per cent of a nation being split by civil war when its capital is captured is increased by this AMOUNT. This percentage is increased by 5 for each city in civil disorder and reduced by 5 for each one celebrating.

City_Unhappy_Size

The maximum number of citizens in each city that are naturally content. In larger cities, new citizens above this limit start out unhappy. Before *Empire_Size_Base* and *Empire_Size_Step* are applied.

Empire_Size_Base

Once your civilization has more cities than the value of this effect, each city gets one more unhappy citizen. If the sum of this effect and *Empire_Size_Step* is zero, there is no such penalty.

Empire_Size_Step

After your civilization reaches *Empire_Size_Base* size, it gets one more unhappy citizen for each AMOUNT of cities it gets above that. Set to zero to disable. You can use `Empire_Size_Step` even if *Empire_Size_Base* is zero.

Max_Rates

The maximum setting for each tax rate is AMOUNT.

Martial_Law_Each

The AMOUNT of citizens pacified by each military unit giving martial law.

Martial_Law_Max

The maximum AMOUNT of units that will give martial law in city.

Rapture_Grow

Can rapture grow cities.

Revolution_Unhappiness

If value is greater than zero, it tells how many turns citizens will tolerate city disorder before government falls.

If value is zero, government never falls.

Has_Senate

Has a senate that prevents declarations of war in most cases.

Inspire_Partisans

Partisan units (defined in `units.ruleset`) may spring up when this player's cities are taken. The number of partisans created is a function of the city size. When multiple unit types have the Partisan flag, the game tries to choose one the player can build. If the player cannot build any, the first Partisan unit type is used. If the created partisans units have a non-zero `pop_cost`, the city will lose citizens accordingly.

Note: This effect can create units that the player could normally not build (because the tech hasn't been researched or the `NoBuild` flag is set). It is your responsibility to set the appropriate requirements to avoid creating advanced units if a city is conquered in the early game.

Happiness_To_Gold

Make all *Make_Content* and *Force_Content* effects instead generate gold.

Max_Trade_Routes

Number of trade routes that city can establish. This is forced on trade route creation only. Existing trade routes are never removed due to reduction of effect value. This is to avoid micro-management, need to create same trade routes again after their max number has been temporarily down.

Fanatics

Units with "Fanatics" flag incur no upkeep.

No_Diplomacy

Cannot use any diplomacy.

Not_Tech_Source

Tech cannot be received from this player by any means.

Trade_Revenue_Bonus

One time trade revenue bonus is multiplied by $2^{(\text{amount} \div 1000)}$. The AMOUNT value is taken from the caravan's home city.

Trade_Revenue_Exponent

One time trade revenue bonus is raised to the $1 + \frac{\text{amount}}{1000}$ power. This is applied before *Trade_Revenue_Bonus*.

Traderoute_Pct

Percentage bonus for trade from traderoutes. This bonus applies after the value of the traderoute is already calculated. It affects one end of the traderoute only.

Unhappy_Factor

Multiply unhappy unit upkeep by AMOUNT.

Upkeep_Factor

Multiply unit upkeep by AMOUNT.

Unit_Upkeep_Free_Per_City

In each city unit upkeep is deducted by this AMOUNT. As usual, you can use with "OutputType" requirement to specify which kind of upkeep this should be.

Output_Waste

Base AMOUNT in percentage that each city has in waste. Waste can be used with any output type, use an “OutputType” requirement to specify which.

Output_Waste_By_Distance

For each tile in real distance that a city is from nearest Government Center, it gets $\frac{\text{amount}}{100}$ of extra waste.

Output_Waste_By_Rel_Distance

City gets extra waste based on distance to nearest Government Center, relative to world size. The AMOUNT of this extra waste is $\frac{\text{distance} \times \text{amount}}{100 \times \text{max_distance}}$

Output_Penalty_Tile

When a tile yields more output than AMOUNT, it gets a penalty of -1.

Output_Inc_Tile_Celebrate

Tiles get AMOUNT extra output when city working them is celebrating.

Upgrade_Price_Pct

Increases unit upgrade cost by AMOUNT percent. This effect works at player level. You cannot adjust upgrade costs for certain unit type or for units upgraded in certain city.

Unit_Shield_Value_Pct

Increase the unit’s value in shields by AMOUNT percent. When this effect is used to determine how many shields the player gets for the actions *Recycle Unit* and *Help Wonder* it gets access to unit state. When it is used to influence the gold cost of *Upgrade Unit* it only has access to unit type.

Retire_Pct

The chance that unit gets retired (removed) when turn changes. Retirement only happens if there are no enemy units or cities within a few tiles. This exists mainly to implement barbarian behavior.

Visible_Wall

Instruct the game to show specific buildings version of the city graphics. Zero or below are considered normal city graphics.

Tech_Cost_Factor

Factor for research costs.

Building_Build_Cost_Pct

Percentage added to building building cost.

Building_Buy_Cost_Pct

Percentage added to building buy cost.

Unit_Build_Cost_Pct

Percentage added to unit building cost.

Unit_Buy_Cost_Pct

Percentage added to unit buy cost.

Nuke_Improvement_Pct

Percentage chance that an improvement would be destroyed while nuking the city. Only regular improvements (not wonders) are affected. Improvements protected from Sabotage (Eg: City Walls) are not affected.

Nuke_Infrastructure_Pct

Percentage chance that an extra located within a nuclear blast area gets destroyed. Only “Infra” extras such as roads and irrigation are affected, and *rmreqs* are also checked. Note that an *Extra* requirement will match any extra on the tile, not only the one considered for destruction.

Shield2Gold_Factor

Factor in percent for the conversion of unit shield upkeep to gold upkeep. A value of 200 would transfer 1 shield upkeep to 2 gold upkeep. The range of this effect must be player or world. Note that only units with the *Shield2Gold* flag will be affected by this.

Tile_Workable

If value > 0, city can work target tile.

Migration_Pct

Increase the calculated migration score for the a city by AMOUNT in percent.

City_Radius_Sq

Increase the squared city radius by AMOUNT. Currently, this can only usefully have “MinSize”, “Building”, or “Tech” requirements.

City_Build_Slots

Increase the number of units with no population cost a city can build in a turn, if there are enough shields.

City_Image

The index for the city image of the given city style.

Victory

Positive value means that player wins the game.

Performance

Value is how much performance type culture city produces.

History

Value is how much history type (cumulative) culture city produces.

National_Performance

Value is how much performance type culture, not tied to any specific city, the nation produces.

National_History

Value is how much history type (cumulative) culture, not tied to any any specific city, the nation produces.

Infra_Points

City increases owner’s infra points by value each turn. If overall points are negative after all cities have been processed, they are set to 0.

Bombard_Limit_Pct

Bombardment may only reduce units to AMOUNT percent (rounded up) of their total hitpoints (*HP*). Unit requirements on this effect are the defending unit itself.

Note: This effect is added automatically with a value of 1 and no reqs. This behavior can be turned off by requiring the +Bombard_Limit_Pct option in `effects.ruleset`.

Wonder_Visible

If the value of this effect is larger than 0 for a small wonder, the wonder will be visible to all players and reported in the intelligence panel in the *nations view*. Great wonders are always visible to everyone through the wonders report (F7). When a small wonder is lost (for instance, because the city it is in is lost or some of its requirements become invalid), it also becomes visible to everyone (this is a limitation of the server).

Note: This effect is added automatically with a value of 1 for great wonders (since they are shown in the wonders report anyway). This behavior can be turned off by requiring the +Wonder_Visible option in `effects.ruleset`.

Nation_Intelligence

Controls the information available in the *Nations View*. *See the detailed description*.

Nation_Intelligence

New in version 3.0: Add the +Nation_Intelligence option to effects.ruleset.

Changed in version 3.1: Removed support for Infra Points.

The Nation_Intelligence effect controls what kind of information about a foreign nation is visible in the *Nations View*. A value of 0 or lower hides the information, while it becomes visible when the effect evaluates to 1 or bigger. For instance, the following effect repurposes the United Nations to reveal everyone's secrets to everyone else:

```
[effect_united_nations_intelligence]
type = "Nation_Intelligence"
value = 1
reqs =
{ "type",      "name",      "range",  "present"
  "Building", "United Nations", "World",  TRUE
}
```

The true power of Nation_Intelligence, however, comes from its combined use with NationalIntelligence requirements. NationalIntelligence lets you select precisely what is visible to a player. Let us say that we want a player owning the Marco Polo Embassy to know the wonders built by any other player it has ever met. This could be achieved as follows:

```
[effect_marco_polo_intelligence]
type = "Nation_Intelligence"
value = 1
reqs =
{ "type",      "name",      "range",  "present"
  "NationalIntelligence", "Wonders",      "Local",  TRUE
  "Building",      "Marco Polo's Embassy", "Player", TRUE
  "DiplRel",      "Never met",      "Local",  FALSE
}
```

It is important to note that the effect works from the point of view of the player who *sees* the information. It cannot depend on what the other player does (except when using DiplRel requirements).

Note: Note the Local range used in the DiplRel requirement above. This is very important: if we had used Player, the effect would be enabled only once the player has met every other player in the game (no player satisfies Never met).

The possible values for NationalIntelligence requirements are as follows:

Table 4.3: National Intelligence Requirements

Culture	The amount of culture accumulated by a player. Not yet shown in the user interface.
Diplomacy	Diplomatic agreements such as Peace, Alliance, and shared vision.
Gold	The amount of gold in the treasury.
Government	The other player's government.
History	The history accumulated by a player. Not yet shown in the user interface.
Multipliers	The value of multipliers selected by the other player. Not yet shown in the user interface.
Mood	For an AI player, whether it is currently farming or building up an army. Not yet shown in the user interface.
Score	The game score.
Tax Rates	The rates of gold, science and luxury in the national budget.
Techs	The techs discovered by a player as well as the current research.
Wonders	The list of wonders owned by the player (see also the <code>Wonder_Visible</code> effect).

Compatibility

For legacy rulesets, the traditional rules are added automatically if the `Nation_Intelligence` option is not requested in `effects.ruleset`. They are rather complicated; to facilitate porting rulesets, we provide a ready-made file implementing them. It can be imported directly from within `effects.ruleset`:

```
*include "default/nation_intelligence_effects.ruleset"
```

Most shipped rulesets use this technique.

4.1.3 Actions

An action is something a player can do to achieve something in the game. It has properties like cost, chance of success and effects. Some of those properties are configurable using effects and other ruleset settings. To learn how to change them read [Effects](#) and the rule set files of classic. An action enabler allows a player to do an action.

Generalized Action Enablers

Some actions have generalized action enablers. An action like that can have zero, one or more action enablers defined for it in the ruleset. The player can do the action only when at least one generalized action enabler says that the action is enabled (and all its hard requirements are fulfilled). A ruleset author can therefore disable an action by not defining any action enablers for it in his ruleset.

A generalized action enabler lives in `game.ruleset`. It consists of the action it enables and two requirement vectors. The first requirement vector, `actor_reqs`, applies to the entity doing the action. The second, `target_reqs`, applies to its target. If both requirement vectors are fulfilled the action is enabled as far as the action enabler is concerned. Note that an action's hard requirements still may make it impossible.

In some situations an action controlled by generalized action enablers may be impossible because of limitations in Freeciv21 itself. Those limitations are called hard requirements. The hard requirements of each action are documented below in the section [Actions and Their Hard Requirements](#).

If the player does not have the knowledge required to find out if an action is enabled or not the action is shown to the player in case it is possible. The game will indicate the uncertainty to the player.

Should the player order a unit to do an illegal action the server will inform the player that his unit was unable to perform the action. The actor unit may also lose a ruleset configurable amount of move fragments.

Example:

```
[actionenabler_veil_the_threat_of_terror]
action = "Incite City"
actor_reqs =
{ "type",      "name",          "range", "present"
  "DiplRel", "Has Casus Belli",  "Local", TRUE
  "DiplRel", "Provided Casus Belli", "Local", FALSE
  "DiplRel", "Is foreign",      "Local", TRUE
}
```

Another Example:

```
[actionenabler_go_bind_your_sons_to_exile]
action = "Incite City"
actor_reqs =
{ "type",      "name",          "range", "present"
  "Tech",      "Flight",        "Player", TRUE
  "DiplRel", "Is foreign", "Local", TRUE
}
target_reqs =
{ "type",      "name",          "range", "present"
  "Tech",      "Writing",       "Player", False
}
```

Above are two action enablers. They both enable the action “Incite City”. If all the conditions of at least one of them are fulfilled it will be enabled. No information is given to the player about what action enabler enabled an action.

The first action enabler, `actionenabler_veil_the_threat_of_terror`, is simple. It allows a player to incite a city if he has a reason to declare war on its owner AND the city’s owner does not have a reason to declare war on him.

The second action enabler, `actionenabler_go_bind_your_sons_to_exile`, is more complex. It allows a player that has Flight to bribe the cities of civilizations that do not have Writing. The complexity is caused by the requirement that the target does not know Writing. If the civilization of the target city knows Writing or not may be unknown to the acting player. To avoid this complexity a requirement that the acting player has an embassy to the target cities civilization (and therefore knowledge about its techs) can be added.

Requirement Vector Rules

An action enabler has two requirement vectors that must be TRUE at the same time. This creates some corner cases you will not find with single requirement vectors. The rules below tries to deal with them.

A `DiplRel` requirement with the range `Local` should always be put in the actor requirements.

- A local `DiplRel` requirement can always be expressed as an actor requirement.
- Only having to care about local `DiplRel` requirements in the actor requirements allows the Freeciv21 code responsible for reasoning about action enablers to be simpler and faster.
- If player A having a diplomatic relationship to player B implies that player B has the same relationship to player A the relationship is symmetric. Examples: `Is foreign` and `War`
- Symmetric local `DiplRel` requirements can be moved directly from the target requirement vector to the actor requirement vector.
- Asymmetric local `DiplRel` requirements must test for the same thing in the opposite direction. Example: `Hosts embassy -> Has embassy`

Actions and Lua

Right before an action is executed, but after it is known to be legal, a signal is emitted to Lua. It has access to the same information as the server. It obviously does not have access to the result of the action since it is not done yet.

The signal's name starts with `action_started_`, then the actor kind, then another `_` and in the end the target kind. The signal that is emitted when a unit performs an action on a city is therefore `action_started_unit_city`.

The signal has three parameters. The first parameter is the action that is about to get started. The second is the actor. The third parameter is the target. The parameters of `action_started_unit_city` is therefore `action`, `actor_unit`, and finally `target city`.

To get the rule name of an action, that is the name used in action enablers, you can use the method `rule_name()`. To get a translated name that is nice to show to players use `name_translation()`.

Example 1

The following Lua code will log all actions done by any unit to a city, to another unit, to a unit stack, to a tile or to itself:

```
function action_started_callback(action, actor, target)
    local target_owner
    if target == nil then
        target_owner = "it self"
    elseif target.owner == nil then
        target_owner = "unowned"
    else
        target_owner = target.owner.nation:plural_translation()
    end

    log.normal(_("%d: %s (rule name: %s) performed by %s %s (id: %d) on %s"),
        game.current_turn(),
        action:name_translation(),
        action:rule_name(),
        actor.owner.nation:plural_translation(),
        actor.utype:rule_name(),
        actor.id,
        target_owner)
end

signal.connect("action_started_unit_city", "action_started_callback")
signal.connect("action_started_unit_unit", "action_started_callback")
signal.connect("action_started_unit_units", "action_started_callback")
signal.connect("action_started_unit_tile", "action_started_callback")
signal.connect("action_started_unit_self", "action_started_callback")
```

Example 2

The following Lua code will make a player that poisons the population of cities risk civil war:

```
function action_started_callback(action, actor, target)
    if action:rule_name() == "Poison City" then
        edit.civil_war(actor.owner, 5);
    end
end

signal.connect("action_started_unit_city", "action_started_callback")
```

Actions and Their Hard Requirements

Freeciv21 can only allow a player to perform an action when the action's hard requirements are fulfilled. Some, but not all, hard requirements can be expressed in an action enabler. Putting them there makes it clearer what the rule actually is. Parts of Freeciv21 reasons about action enablers. Examples are self contradicting rule detection and the help system. Including the hard requirements rules in each enabler of its action is therefore obligatory for some hard requirements. Those hard requirements are marked with an exclamation mark (!).

Actions Done By A Unit Against A City

Establish Embassy

Establish a real embassy to the target player.

Rules:

- UI name can be set using `ui_name_establish_embassy`.
- actor must be aware that the target exists.
- actor cannot have a real embassy to the target player. (!)
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Establish Embassy Stay

Establish a real embassy to the target player.

Rules:

- UI name can be set using `ui_name_establish_embassy`.
- spends the actor unit.
- actor must be aware that the target exists.
- actor cannot have a real embassy to the target player. (!)
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Investigate City

Look at the *city dialog* of a foreign city.

Rules:

- UI name can be set using `ui_name_investigate_city`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Investigate City Spend Unit

Look at the *city dialog* of a foreign city.

Rules:

- UI name can be set using `ui_name_investigate_city`.
- spends the actor unit.
- actor must be aware that the target exists.

- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Sabotage City

Destroy a building or the production in the target city.

Rules:

- UI name can be set using `ui_name_sabotage_city`.
- spends the actor unit.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Sabotage City Escape

Destroy a building or the production in the target city.

Rules:

- UI name can be set using `ui_name_sabotage_city_escape`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Targeted Sabotage City

Destroy a building in the target city.

Rules:

- UI name can be set using `ui_name_targeted_sabotage_city`.
- spends the actor unit.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Targeted Sabotage City Escape

Destroy a building in the target city and escape.

Rules:

- UI name can be set using `ui_name_targeted_sabotage_city_escape`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Sabotage City Production

Sabotage the city's production.

Rules:

- UI name can be set using `ui_name_sabotage_city_production`.
- spends the actor unit.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Sabotage City Production Escape

Sabotage the city's production and escape.

Rules:

- UI name can be set using `ui_name_sabotage_city_production_escape`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Poison City

Kill a citizen in the target city.

Rules:

- UI name can be set using `ui_name_poison_city`.
- spends the actor unit.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Poison City Escape

Kill a citizen in the target city and escape.

Rules:

- UI name can be set using `ui_name_poison_city_escape`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Spread Plague

Bio-terrorism. Infect the target city with an illness.

Rules:

- UI name can be set using `ui_name_spread_plague`.
- set if the actor unit is spent with `spread_plague_actor_consuming_always`.
- may infect trade route connected cities if `illness.illness_on` is TRUE.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Steal Tech

Steal a random tech from the target's owner.

Rules:

- UI name can be set using `ui_name_steal_tech`.
- spends the actor unit.
- will always fail when the tech theft is expected. Tech theft is expected when the number of previous tech thefts from the target city is above the limit set by the *Stealings_Ignore* effect.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Steal Tech Escape Expected

Steal a random tech from the target's owner and escape.

Rules:

- UI name can be set using `ui_name_steal_tech_escape`.

- more likely to fail when the tech theft is expected. Tech theft is expected when the number of previous tech thefts from the target city is above the limit set by the *Stealings_Ignore* effect.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Targeted Steal Tech

Steal a specific tech from the targets owner.

Rules:

- UI name can be set using `ui_name_targeted_steal_tech`.
- spends the actor unit.
- will always fail when the tech theft is expected. Tech theft is expected when the number of previous tech thefts from the target city is above the limit set by the *Stealings_Ignore* effect.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Targeted Steal Tech Escape Expected

Steal a specific tech from the targets owner and escape.

Rules:

- UI name can be set using `ui_name_targeted_steal_tech_escape`.
- more likely to fail when the tech theft is expected. Tech theft is expected when the number of previous tech thefts from the target city is above the limit set by the *Stealings_Ignore* effect.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Incite City

Pay the target city to join the actor owner's side.

Rules:

- UI name can be set using `ui_name_incite_city`.
- spends the actor unit.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Incite City Escape

Pay the target city to join the actor owner's side and escape.

Rules:

- UI name can be set using `ui_name_incite_city_escape`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

- target must be foreign. (!)

Steal Gold

Steal some gold from the owner of the target city.

Rules:

- UI name can be set using `ui_name_steal_gold`.
- adjustable with the *Max_Stolen_Gold_Pm* effect and with the *Thiefs_Share_Pm* effect.
- spends the actor unit.
- actor must be aware that the target exists.
- the targets owner must have more than 0 gold.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Steal Gold Escape

Steal some gold from the owner of the target city and escape.

Rules:

- UI name can be set using `ui_name_steal_gold_escape`.
- adjustable with the *Max_Stolen_Gold_Pm* effect and with the *Thiefs_Share_Pm* effect.
- actor must be aware that the target exists.
- the targets owner must have more than 0 gold.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Steal Maps

Steal parts of the owner of the target city's map.

Rules:

- UI name can be set using `ui_name_steal_maps`.
- adjustable with the *Maps_Stolen_Pct* effect and the ruleset setting `steal_maps_reveals_all_cities`.
- spends the actor unit.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Steal Maps Escape

Steal parts of the owner of the target city's map and escape.

Rules:

- UI name can be set using `ui_name_steal_maps_escape`.
- adjustable with the *Maps_Stolen_Pct* effect and the ruleset setting `steal_maps_reveals_all_cities`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)

Suitcase Nuke

Cause a nuclear explosion in the target city.

Rules:

- UI name can be set using `ui_name_suitcase_nuke`.
- spends the actor unit.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Suitcase Nuke Escape

Cause a nuclear explosion in the target city and escape.

Rules:

- UI name can be set using `ui_name_suitcase_nuke_escape`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Destroy City

Destroys the target city.

Rules:

- UI name can be set using `ui_name_destroy_city`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.

Establish Trade Route

Establish a trade route to the target city.

Rules:

- UI name can be set using `ui_name_establish_trade_route`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target or on the tile next to it.
- actor must have a home city. (!)
- target must be foreign or `trademindist` tiles away from that home city.
- trade route type pct (see “Trade settings”) cannot be 0%.
- it is possible to establish a trade route between the cities as far as the two cities them self are concerned.
Example: If one of the cities cannot have any trade routes at all it is impossible to establish a new one.

Enter Marketplace

Get a one time bonus without creating a trade route.

Rules:

- UI name can be set using `ui_name_enter_marketplace`.
- actor must be aware that the target exists.
- if `force_trade_route` is TRUE “Establish Trade Route” must be impossible.
- actor must be on the same tile as the target or on the tile next to it.
- actor must have a home city. (!)

- target must be foreign or trademindist tiles away from that home city.
- trade route type (see Trade settings) cannot be 0%.

Help Wonder

Add the shields used to build the actor to the target city.

Rules:

- UI name can be set using `ui_name_help_wonder`.
- adjustable with the *Unit_Shield_Value_Pct* effect.
- actor must be aware that the target exists.
- actor must be on the same tile as the target unless `help_wonder_max_range` allows it to be further away. Default `help_wonder_max_range` is 1.
- target city must need the extra shields to complete its production.

Recycle Unit

Add half the shields used to build the unit to target.

Rules:

- UI name can be set using `ui_name_recycle_unit`.
- adjustable with the *Unit_Shield_Value_Pct* effect.
- actor must be aware that the target exists.
- “Help Wonder” must be impossible.
- actor must be on the same tile as the target unless `recycle_unit_max_range` allows it to be further away. Default `recycle_unit_max_range` is 1.
- target city must need the extra shields to complete its production.

Join City

Add the actor to the target city’s population.

Rules:

- UI name can be set using `ui_name_join_city`.
- actor must be aware that the target exists.
- actor must have population to add (set in `pop_cost`).
- actor must be on the same tile as the target or on the tile next to it.
- target city population must not become higher that the `add_to_size_limit` setting permits.
- target must be able to grow to the size that adding the unit would result in.

Home City

Set target city as the actor unit’s new home city.

Rules:

- UI name can be set using `ui_name_home_city`.
- actor must be aware that the target exists.
- actor must be on the same tile as the target.
- actor must not have the NoHome unit type flag. (!)
- cannot set existing home city as new home city.

- target city has enough unused unit maintenance slots to support the actor unit. No problem if the actor unit spends 0 city slots.

Upgrade Unit

Upgrade the actor unit using the target's facilities.

Rules:

- UI name can be set using `ui_name_upgrade_unit`.
- adjustable with the *Unit_Shield_Value_Pct* effect.
- actor must be aware that the target exists.
- actor must be on the same tile as the target.
- actor player must have enough gold to pay for the upgrade.
- actor unit must have a type to upgrade to (`obsoleted_by`).
- actor unit's upgraded form must be able to exist at its current location.
- actor unit's upgraded form must have room for its current cargo.
- target player must be able to build the unit upgraded to.
- target city must be domestic. (!)

Airlift Unit

Airlift actor unit to target city.

Rules:

- UI name can be set using `ui_name_airlift_unit`.
- max legal distance to the target can be set using `airlift_max_range`.
- actor must be aware that the target exists.
- the actor unit is not transporting another unit. (!)
- the actor unit is not inside the target city.
- the actor unit can exist in the target city (outside a transport).
- the actor unit is in a city. (!)
- the city the actor unit is in:
 - is domestic or, if `airliftingstyle` permits it, allied.
 - has Airlift (see the *Airlift* effect and the `airliftingstyle` setting).
- the target city is domestic or, if `airliftingstyle` permits it, allied.
- the target city has Airlift.

Nuke City

Detonate in the target city. Cause a nuclear explosion.

Rules:

- UI name can be set using `ui_name_nuke_city`.
- if `force_capture_units` is TRUE “Capture Units” must be impossible.
- if `force_bombard` is TRUE “Bombard”, “Bombard 2”, and “Bombard 3” must be impossible.
- the actor unit must be on a tile next to the target unless `nuke_city_max_range` allows it to be further away.

Conquer City

Conquer the target city.

Rules:

- UI name can be set using `ui_name_conquer_city`.
- actor must be aware that the target exists.
- if `force_capture_units` is TRUE “Capture Units” must be impossible.
- if `force_bombard` is TRUE “Bombard”, “Bombard 2”, and “Bombard 3” must be impossible.
- if `force_explode_nuclear` is TRUE “Explode Nuclear”, “Nuke Units”, and “Nuke City” must be impossible.
- “Attack” must be impossible.
- the actor unit must be on a tile next to the target.
- the actor player’s nation cannot be an animal barbarian. (!)
- the actor unit’s current transport, if the actor unit is transported, must be in a city or in a base native to the current transport if the current transport’s unit class has the `Unreachable` unit class flag and the actor’s unit type does not list the current transport’s unit class in disembarks.
- the actor unit does not have the `CoastStrict` unit type flag or the target city is on or adjacent to a tile that does not have the `UnsafeCoast` terrain flag.
- the actor unit cannot be diplomatically forbidden from entering the tile of the target city.
- the actor unit has the `CanOccupyCity` unit class flag. (!)
- the actor cannot have the `NonMil` unit type flag. (!)
- the actor unit has at least one move fragment left. (!)
- the actor’s relationship to the target is War. (!)
- actor unit must be able to exist outside of a transport at the target’s tile.
- the target must be foreign. (!)
- the target city contains 0 units. (!)

Conquer City 2

Conquer the target city.

Rules:

- UI name can be set using `ui_name_conquer_city_2`.
- A copy of “Conquer City”.
- See “Conquer City” for everything else.

Surgical Strike Building

Destroy a specific building.

Rules:

- UI name can be set using `ui_name_surgical_strike_building`.
- actor must be aware that the target exists.
- the actor unit must be on a tile next to the target.

Surgical Strike Production

Destroy the city production.

Rules:

- UI name can be set using `ui_name_surgical_strike_production`.
- actor must be aware that the target exists.
- the actor unit must be on a tile next to the target.

Actions Done By A Unit Against Another Unit

Sabotage Unit

Halve the target unit's hit points.

Rules:

- UI name can be set using `ui_name_sabotage_unit`.
- spends the actor unit.
- actor must be on the same tile as the target or on the tile next to it.
- target must be visible for the actor.

Sabotage Unit Escape

Halve the target unit's hit points and escape.

Rules:

- UI name can be set using `ui_name_sabotage_unit_escape`.
- actor must be on the same tile as the target or on the tile next to it.
- target must be visible for the actor.

Bribe Unit

Make the target unit join the actor owner's side.

Rules:

- UI name can be set using `ui_name_bribe_unit`.
- actor must be on the same tile as the target or on the tile next to it.
- target must be foreign. (!)
- target must be visible for the actor.

Expel Unit

Expel the target unit to its owner's capital.

Rules:

- UI name can be set using `ui_name_expel_unit`.
- actor must be on the same tile as the target or on the tile next to it.
- target must be visible for the actor.
- target's owner must have a capital.

Heal Unit

Restore the target unit's health.

Rules:

- UI name can be set using `ui_name_heal_unit`.
- actor must be on the same tile as the target or on the tile next to it.
- target must be visible for the actor.

Transport Alight

Exit target transport to same tile.

Rules:

- UI name can be set using `ui_name_transport_alight`.
- actor must be on the same tile as the target.
- actor must be transported. (!)
- actor must be on a livable tile. (!)
- target must be transporting. (!)
- target must be in a city or in a native base if the target's unit class has the `Unreachable` unit class flag and the actor's unit type does not list the target's unit class in disembarks.
- target must be visible for the actor.

Transport Unload

Unload the target unit to same tile.

Rules:

- UI name can be set using `ui_name_transport_unload`.
- actor must be on the same tile as the target.
- actor must have a `transport_cap` greater than 0.
- actor must be transporting. (!)
- actor must be in a city or in a native base if the actor's unit class has the `Unreachable` unit class flag and the target's unit type does not list the actor's unit class in disembarks.
- target must be transported. (!)
- target must be on a livable tile. (!)
- target must be visible for the actor.

Transport Board

Enter target transport on the same tile.

Rules:

- UI name can be set using `ui_name_transport_board`.
- the actor unit cannot currently be transported by the target unit.
- the actor unit's current transport, if the actor unit is transported, must be in a city or in a base native to the current transport if the current transport's unit class has the `Unreachable` unit class flag and the actor's unit type does not list the current transport's unit class in the disembarks field.
- the actor's unit class must appear in the target unit type's cargo field.
- the actor unit unit type must be different from the target unit type.
- the actor unit or its (recursive) cargo, if it has cargo, must be unable to transport itself, the target unit or the target unit's transporters. See `unit_transport_check()`.
- boarding will not cause a situation with more than 5 recursive transports.

- the target unit must be domestic, allied or on the same team as the actor unit is. (!)
- target must be in a city or in a base native to it if the target's unit class has the `Unreachable` unit class flag and the actor's unit type does not list the target's unit class in the `embarks` field.
- the target must be transporting fewer units than its unit type's `transport_cap` field.
- target must be visible to the actor.

Transport Embark

Enter target transport on a different tile.

Rules:

- UI name can be set using `ui_name_transport_embark`.
- the actor unit must be on a tile next to the target.
- the actor unit has at least one move fragment left. (!)
- the actor unit cannot currently be transported by the target unit.
- the actor unit's current transport, if the actor unit is transported, must be in a city or in a base native to the current transport if the current transport's unit class has the `Unreachable` unit class flag and the actor's unit type does not list the current transport's unit class in the `disembarks` field.
- the actor unit's type must be the target tile's terrain animal if the player's nation is an animal barbarian.
- the actor unit cannot be diplomatically forbidden from entering the target tile.
- the actor unit does not have the `CoastStrict` unit type flag or the target city is on or adjacent to a tile that does not have the `UnsafeCoast` terrain flag.
- actor unit must be able to move to the target tile.
- the actor's unit class must appear in the target unit type's `cargo` field.
- the actor unit unit type must be different from the target unit type.
- the actor unit or its (recursive) cargo, if it has cargo, must be unable to transport itself, the target unit or the target unit's transporters. See `unit_transport_check()`.
- boarding will not cause a situation with more than 5 recursive transports.
- the target unit must be domestic, allied, or on the same team as the actor unit is. (!)
- target must be in a city or in a base native to it if the target's unit class has the `Unreachable` unit class flag and the actor's unit type does not list the target's unit class in the `embarks` field.
- the target must be transporting fewer units than its unit type's `transport_cap` field.
- the target tile cannot contain any city or units not allied to the actor unit and all its cargo.
- target must be visible to the actor.

Actions Done By A Unit Against All Units At A Tile

Capture Units

Steal the target units.

Rules:

- UI name can be set using `ui_name_capture_units`.
- actor must be on a tile next to the target.
- target must be foreign. (!)
- target cannot be transporting other units. (!)

Bombard

Bombard the units (and city) at the tile without killing them.

Rules:

- UI name can be set using `ui_name_bombard`.
- if `force_capture_units` is TRUE, “Capture Units” must be impossible.
- actor must have a `bombard_rate` > 0.
- actor must have an `attack` > 0.
- actor must be on a tile next to the target or, if `bombard_max_range` allows it, further away.
- target cannot be in a city the actor player is not at war with.
- target owner must be at war with actor. (!)

Bombard 2

Bombard the units (and city) at the tile without killing them.

Rules:

- UI name can be set using `ui_name_bombard_2`.
- actor must be on a tile next to the target or, if `bombard_2_max_range` allows it, further away.
- A copy of “Bombard”.
- See “Bombard” for everything else.

Bombard 3

Bombard the units (and city) at the tile without killing them.

Rules:

- UI name can be set using `ui_name_bombard_3`.
- actor must be on a tile next to the target or, if `bombard_3_max_range` allows it, further away.
- A copy of “Bombard”.
- See “Bombard” for everything else.

Attack

Attack an enemy unit, possibly survive.

Rules:

- UI name can be set using `ui_name_attack`.
- if `force_capture_units` is TRUE, “Capture Units” must be impossible.

- if `force_bombard` is TRUE, “Bombard”, “Bombard 2”, and “Bombard 3” must be impossible.
- if `force_explode_nuclear` is TRUE, “Explode Nuclear”, “Nuke Units”, and “Nuke City” must be impossible.
- the actor must be on the tile next to the target.
- the actor’s attack must be above 0.
- the actor cannot have the `NonMil` unit type flag. (!)
- the actor must be native to the target tile unless it has the `AttackNonNative` unit class flag and not the `Only_Native_Attack` unit type flag.
- the target tile has no non enemy units. (!)
- the target tile has no non enemy city.
- one or all (unreachable protects) non transported units at the target tile must be reachable. A unit is reachable if any of the following is true:
 - it does not have the `Unreachable` unit class flag.
 - it is listed in the actor unit’s targets.
 - it is in a city.
 - it is on a tile with a native Extra.

Suicide Attack

Attack an enemy unit, die immediately.

Rules:

- UI name can be set using `ui_name_suicide_attack`.
- if `force_capture_units` is TRUE, “Capture Units” must be impossible.
- if `force_bombard` is TRUE, “Bombard”, “Bombard 2”, and “Bombard 3” must be impossible.
- if `force_explode_nuclear` is TRUE, “Explode Nuclear”, “Nuke Units”, and “Nuke City” must be impossible.
- the actor must be on the tile next to the target.
- the actor’s attack must be above 0.
- the actor cannot have the `NonMil` unit type flag. (!)
- the actor must be native to the target tile unless it has the `AttackNonNative` unit class flag and not the `Only_Native_Attack` unit type flag.
- the target tile has no non enemy units. (!)
- the target tile has no non enemy city.
- one or all (unreachable protects) non transported units at the target tile must be reachable. A unit is reachable if any of the following is true:
 - it does not have the `Unreachable` unit class flag.
 - it is listed in the actor unit’s targets.
 - it is in a city.
 - it is on a tile with a native Extra.

Nuke Units

Detonate at the target unit stack. Cause a nuclear explosion.

Rules:

- UI name can be set using `ui_name_nuke_units`.
- if `force_capture_units` is TRUE, “Capture Units” must be impossible.
- if `force_bombard` is TRUE, “Bombard”, “Bombard 2”, and “Bombard 3” must be impossible.
- the actor unit must be on a tile next to the target unless `nuke_units_max_range` allows it to be further away.
- one or all (unreachable protects) non transported units at the target tile must be reachable. A unit is reachable if any of the following is true:
 - it does not have the `Unreachable` unit class flag.
 - it is listed in the actor unit’s targets.
 - it is in a city.
 - it is on a tile with a native Extra.

Spy Attack

Trigger a diplomatic battle to eliminate tile defenders.

Rules:

- UI name can be set using `ui_name_spy_attack`.
- the actor must be on the tile next to the target.
- the target tile must have at least 1 diplomatic defender.

Actions Done By A Unit Against A Tile**Found City**

Found a city at the target tile.

Rules:

- UI name can be set using `ui_name_found_city`.
- city name must be legal.
- the scenario setting `prevent_new_cities` must be false.
- actor must be on the same tile as the target.
- target must not have the `NoCities` terrain flag. (!)
- target must not be closer than `citymindist` to nearest city.

Explode Nuclear

Detonate at the target tile. Cause a nuclear explosion.

Rules:

- UI name can be set using `ui_name_explode_nuclear`.
- if `force_capture_units` is TRUE, “Capture Units” must be impossible.
- if `force_bombard` is TRUE, “Bombard”, “Bombard 2”, and “Bombard 3” must be impossible.

- actor must be on the same tile as the target unless `explode_nuclear_max_range` allows it to be further away.

Paradrop Unit

Move the actor unit to the target tile.

Rules:

- UI name can be set using `ui_name_paradrop_unit`.
- can result in the conquest of the city at the target tile if:
 - the actor player is not an animal barbarian.
 - the actor unit has the `CanOccupyCity` unit class flag.
 - the actor do not have the `NonMil` unit type flag.
 - the actor's relationship to the target is War.
 - the target city contains 0 units.
- the distance between actor and target is from 1 to `paratroopers_range`.
- the actor unit has not paradropped this turn.
- the actor unit is not transporting another unit. (!)
- the actor unit cannot be diplomatically forbidden from entering the target tile. (!)
- the target tile is known (does not have to be seen) by the actor.
- if the target tile is seen:
 - the actor unit must be able to exist outside a transport on the target tile. If the target tile does not have a visible transport the actor unit is able to load into on landing.
 - the target tile cannot contain a city belonging to a player the actor has Peace, Cease-Fire, or Armistice with.
 - the target tile cannot contain any seen unit belonging to a player the actor player has Peace, Cease-Fire, or Armistice with.

Transform Terrain

Transform tile terrain type.

Rules:

- UI name can be set using `ui_name_transform_terrain`.
- the actor unit has `Settlers` flag. (!)
- terrain type must be one that can be transformed.

Cultivate

Transform tile terrain type by irrigating.

Rules:

- UI name can be set using `ui_name_cultivate_tf`.
- the actor unit has `Settlers` flag. (!)
- terrain type must be one that can be transformed by irrigating.

Plant

Transform tile terrain type by planting.

Rules:

- UI name can be set using `ui_name_plant`.
- the actor unit has `Settlers` flag. (!)
- terrain type must be one that can be transformed by mining.

Pillage

Pillage extra from tile.

Rules:

- UI name can be set using `ui_name_pillage`.
- terrain type must be one where pillaging is possible.
- the target extra must be present at the target tile.
- the terrain of the target tile must have a non zero `pillage_time`.
- no other unit can be pillaging the target extra.
- the target extra must have the `Pillage` removal cause.
- the target extra's `rmreqs` must be fulfilled.
- the target extra cannot be a dependency of another extra present at the target tile.
- the target extra cannot have the `AlwaysOnCityCenter` extra flag if the target tile has a city.
- the target extra cannot have the `AutoOnCityCenter` extra flag if the target tile has a city and the city's owner can rebuild it.
- the target extra must be the rule chosen extra if the `civstyle` section's `pillage_select` is `FALSE`.

Clean Pollution

Clean extra from the target tile.

Rules:

- UI name can be set using `ui_name_clean_pollution`.
- actor must be on the same tile as the target.
- the actor unit has the `Settlers` unit type flag. (!)
- the target extra must be present at the target tile.
- the terrain of the target tile must have a non zero `clean_pollution_time`.
- the target extra must have the `CleanPollution` removal cause.
- the target extra's `rmreqs` must be fulfilled.
- the target extra cannot have the `AlwaysOnCityCenter` extra flag if the target tile has a city
- the target extra cannot have the `AutoOnCityCenter` extra flag if the target tile has a city and the city's owner can rebuild it.

Clean Fallout

Clean extra from the target tile.

Rules:

- UI name can be set using `ui_name_clean_fallout`.
- actor must be on the same tile as the target.
- the actor unit has the `Settlers` unit type flag. (!)
- the target extra must be present at the target tile.

- the terrain of the target tile must have a non zero `clean_fallout_time`.
- the target extra must have the `CleanFallout` removal cause.
- the target extra's `rmreqs` must be fulfilled.
- the target extra cannot have the `AlwaysOnCityCenter` extra flag if the target tile has a city.
- the target extra cannot have the `AutoOnCityCenter` extra flag if the target tile has a city and the city's owner can rebuild it.

Build Road

Build road at the target tile.

Rules:

- UI name can be set using `ui_name_road`.
- actor must be on the same tile as the target.
- the actor unit has the `Settlers` unit type flag. (!)
- the target tile cannot have an extra that the target extra must bridge over (see extra type's `bridged_over` value) unless the actor player knows a tech with the `Bridge` tech flag.
- the target extra (the extra to be built) is a road.
- the target tile does not already have the target extra.
- the target extra is buildable (see extra type's `buildable` value).
- the target tile's terrain's `road_time` is not 0.
- if the target extra is both a road and a base the target tile's terrain's `base_time` is not 0.
- if the target extra is both a road and a base the target extra cannot claim land (see base type's `border_sq` value) if the target tile has a city.
- to begin a road the build requirements of the target road (see road type's `first_reqs` value) must be fulfilled. Building a road when no (cardinal) adjacent tile has the target extra is considered beginning it.
- the build requirements of the target extra (see extra type's `reqs` value) must be fulfilled.

Build Base

Build base at the target tile.

Rules:

- UI name can be set using `ui_name_build_base`.
- the actor unit has the `Settlers` unit type flag. (!)
- the target tile cannot have an extra that the target extra must bridge over (see extra type's `bridged_over` value) unless the actor player knows a tech with the `Bridge` tech flag.
- the target extra (the extra to be built) is a base.
- the target tile does not already have the target extra.
- the target extra is buildable (see extra type's `buildable` value).
- the target tile's terrain's `base_time` is not 0.
- the target extra cannot claim land (see base type's `border_sq` value) if the target tile has a city.
- if the target extra is both a road and a base the target tile's terrain's `road_time` is not 0.
- the build requirements of the target extra (see extra type's `reqs` value) must be fulfilled.

Build Mine

Build mine at the target tile.

Rules:

- UI name can be set using `ui_name_build_mine`.
- actor must be on the same tile as the target.
- the actor unit has the `Settlers` unit type flag. (!)
- the target tile cannot have an extra that the target extra must bridge over (see extra type's `bridged_over` value) unless the actor player knows a tech with the `Bridge` tech flag.
- the target extra (the extra to be built) is a mine.
- the target tile does not already have the target extra.
- the target extra is buildable (see extra type's `buildable` value).
- the target tile's terrain's `mining_time` is not 0.
- the target tile's terrain's `mining_result` is "yes".
- if the target extra is both a mine and a base the target tile's terrain's `base_time` is not 0.
- if the target extra is both a mine and a base the target extra cannot claim land (see base type's `border_sq` value) if the target tile has a city.
- if the target extra is both a mine and a road the target tile's terrain's `road_time` is not 0.
- the build requirements of the target extra (see extra type's `reqs` value) must be fulfilled.

Build Irrigation

Build irrigation at the target tile.

Rules:

- UI name can be set using `ui_name_irrigate`.
- actor must be on the same tile as the target.
- the actor unit has the `Settlers` unit type flag. (!)
- the target tile cannot have an extra that the target extra must bridge over (see extra type's `bridged_over` value) unless the actor player knows a tech with the `Bridge` tech flag.
- the target extra (the extra to be built) is an irrigation.
- the target tile does not already have the target extra.
- the target extra is buildable (see extra type's `buildable` value).
- the target tile's terrain's `irrigation_time` is not 0.
- the target tile's terrain's `irrigation_result` is "yes".
- if the target extra is both an irrigation and a base the target tile's terrain's `base_time` is not 0.
- if the target extra is both an irrigation and a base the target extra cannot claim land (see base type's `border_sq` value) if the target tile has a city.
- if the target extra is both an irrigation and a road the target tile's terrain's `road_time` is not 0.
- the build requirements of the target extra (see extra type's `reqs` value) must be fulfilled.

Transport Disembark

Exit transport to target tile.

Rules:

- UI name can be set using `ui_name_transport_disembark`.
- the actor unit must be on a tile next to the target.
- the actor unit has at least one move fragment left. (!)
- actor must be transported. (!)
- the actor unit's transport must be in a city or in a native base if the transport's unit class has the `Unreachable` unit class flag and the actor's unit type does not list the target's unit class in disembarks.
- the actor unit does not have the `CoastStrict` unit type flag or the target city is on or adjacent to a tile that does not have the `UnsafeCoast` terrain flag.
- the actor unit cannot be diplomatically forbidden from entering the target tile.
- the actor unit's type must be the target tile's terrain animal if the player's nation is an animal barbarian.
- actor unit must be able to exist outside of a transport at the target tile.
- actor unit must be able to move to the target tile.
- the target tile is not blocked for the actor unit by some other unit's zone of control (ZOC)
- the target tile cannot contain any city or units not allied to the actor unit and all its cargo.

Transport Disembark 2

Exit transport to target tile.

Rules:

- UI name can be set using `ui_name_transport_disembark_2`.
- A copy of "Transport Disembark".
- See "Transport Disembark" for everything else.

Actions Done By A Unit To It Self

Disband Unit

Disband the unit.

Rules:

- spends the actor unit. Gives nothing in return. No shields spent to build the unit is added to the shield stock of any city even if the unit is located inside it.
- UI name can be set using `ui_name_disband_unit`.
- "Help Wonder" must be impossible.
- "Recycle Unit" must be impossible.

Fortify

Fortify at tile.

Rules:

- UI name can be set using `ui_name_fortify`.
- the actor unit cannot already be fortified. (!)

Convert Unit

Convert the unit to another unit type.

Rules:

- UI name can be set using `ui_name_convert_unit`.
- actor unit must have a type to convert to (`convert_to`).
- actor unit's converted form must be able to exist at its current location.
- actor unit's converted form must have room for its current cargo.

Ruleset Defined Actions

User actions are “blank”. The ruleset does everything they do. The following ruleset variables allows user action number `n` to be further customized:

- `ui_name_user_action_n`: The UI name shown to the user in the action selection dialog.
- `user_action_n_target_kind`: The kind of target the action is done to. See `target_reqs`. Legal values: “individual cities”, “individual units”, “unit stacks”, “tiles”, or “itself”
- `user_action_n_min_range` and `user_action_n_max_range`: What distance from the actor to the target is permitted for the action.
- `user_action_n_actor_consuming_always`: TRUE if Freeciv21 should make sure that the actor is spent after the action is successfully done.

The current ruleset defined actions are “User Action 1”, “User Action 2”, and “User Action 3”.

4.1.4 Achievements

Achievements are something a player can gain in game by reaching set goals. Achievements active in the ruleset are defined in `game.ruleset`.

Depending on whether achievement is defined unique or not, they are granted only to the first player, or all players, to reach the goal for the achievement. Goal is defined by achievement type and another value specific to that achievement type.

Achievement Types**Map_Known**

Achievement is granted when player has mapped at least `<value>`% of the world.

Spaceship

This achievement is granted when player launches spaceship. `<Value>` is ignored.

Multicultural

Achievement is granted when player has citizens of at least `<value>` different nationalities (across all their cities).

Cultured_City

Achievement is granted when player has a city with at least `<value>` culture points.

Cultured_Nation

Achievement is granted when player has at least `<value>` culture points.

Lucky

Achievement is granted on turn when random number generator gives value less than `<value>` out of 10,000.

Huts

Achievement is granted once player has entered <value> huts.

Metropolis

Achievement is granted once there's city of at least size <value> in the player's empire.

Literate

Achievement is granted when player's literacy percent is at least <value>.

Land_Ahoy

Achievement is granted when player has seen <value> different islands/continents. Home continent counts, so to give achievements for finding first other continent, use value 2.

4.1.5 Nation Sets and Flags

This page describes the contents of the nation .spec files. This is intended as developer reference, and for people wanting to create/compile alternative nation files for Freeciv21. A nation consists of a nation file in the rulesets and a flag in the tilesets.

There are already many nations, but of course some of them are missing. We are quite open to new “nations” even if they are just part of a larger country, but obscure towns that nobody knows will be rejected. Things like US states or German Länder are OK. South African provinces are probably not.

The following information is required to add a new nation:

- The name of the citizens (singular and plural), for instance *Indian* and *Indians*.
- A set of “groups” under which the nation will be shown in the client. Usually, you can copy the groups from an existing nation.
- A short historical description of the nation, shown in the Help dialog.
- Several (real) leader names, preferably both men and women. Prefer recognized historically important leaders, eg Henry IV (https://en.wikipedia.org/wiki/Henry_IV_of_France) and Napoléon Bonaparte (<https://en.wikipedia.org/wiki/Napoleon>), but not Jean Casimir-Perier (https://en.wikipedia.org/wiki/Jean_Casimir-Perier).
- Special names for the ruler in certain governments, if applicable. For instance, when in Despotism the Egyptian leader is called *Pharaoh*.
- A flag, preferably something official (if your nation does not have an official flag, think twice before including it). Wikipedia has many of those under free licenses.
- A few nations that will be preferred by the server when there is a civil war. For instance, the *Confederate* are a civil war nation for the Americans.
- City names. List each city only once, even if it changed name in the course of history. Try to use names from the same epoch.

Most of the information goes into your nation's .ruleset file. There are many examples in the code repository.

Four variants of the flag are needed:

- A “small” flag, 29x20 with a 1px black border
- A “large” flag, 44x30 with a 1px black border
- A “shield”, 14x14 with a 1px black border and a shield shape
- A “large shield”, 19x19 with a 1px black border and a shield shape

Information on where the files go is described below.

Local Nation Files

Most supplied rulesets nations can be added locally without need to modify Freeciv21 distribution files. This section discuss the way the local override files work. Later sections assume that a nation is being *added* to the main Freeciv21 distribution, even if only to locally modified copy.

Freeciv21 searches data files from several directories in priority order. Local nations overrides mechanism uses this to include files from user data directory, `~/.local/share/freeciv21/<version>/override/`. Freeciv21 distribution has empty versions of those files in a lower priority directory. Once user adds the file, it gets used instead of the empty one.

- `~/.local/share/freeciv21/<version>/override/nation.ruleset`

Ruleset sections for nations that user wants to add. This can of course use `*include` directives so that individual nations are in separate files. See below sections for the format of the nation rulesets.

- `~/.local/share/freeciv21/<version>/override/flags.spec`
- `~/.local/share/freeciv21/<version>/override/shields.spec`
- `~/.local/share/freeciv21/<version>/override/flags-large.spec`
- `~/.local/share/freeciv21/<version>/override/shields-large.spec`

Spec files for flag graphics to add. See below sections for the format of spec files and graphics files.

How to add a Nation

To add a nation of your own, you should look at the following files:

- `data/nation/<nationname>.ruleset`

This is the new nation, which you will have to create. It may help to copy one of the other nation files over and edit it. See below for a style guide for nation files.

- The `<nationname>` bit is to be replaced with the nations name. **Do not** use whitespaces and special characters. Underlines are ok though.
- The name should be the same as the name of the nation inside the ruleset file.
- The file must be encoded in UTF-8.
- `data/default/nationlist.ruleset`

This lists all nation files. Add your nation (`data/nation/<nationname>.ruleset`) to this list.

- `data/flags/*`

This is the flags directory. You will have to add a flag-file (see below) for your nation to work.

- `data/scenarios/*`

You can add starting position for your nation on a scenario map.

Before a nation can be included in the main distribution, the following files will also have to be edited. Unless you know what you're doing you should not need to worry about this.

- `data/nation/CMakeLists.txt`

Another list of nation files - add your nation (`<nationname>.ruleset`) to this list.

- `translations/nations/POTFILES.in`

Here is yet another list of nations files; again add your nation (`data/nation/<nationname>.ruleset`) to it. Nations part of the “core” group go to `translations/freeciv/POTFILES.in` instead.

Note: There is a short list of nations listed in `translations/core/POTFILES.in` that translators can target first as part of an internationalization efforts. The larger list of nations in `translations/nations/POTFILES.in` is where new less-used nations should go.

How to add a Flag

Overview

PNG is the preferred form for graphics, and flags should be made exclusively in SVG.

A new nation needs a new flag. All flags are stored in SVG (Scalable Vector Graphics) format. Sodipodi and Inkscape are two good SVG editors. If you are creating a real-world nation you can probably find a free or public domain flag that can be used. One good place to look is the Open Clip Art Library (OCAL). Remember that any flags we add must be licenced under the GPL and should be *attributed* to their original author, so make a note of where you found the flag, what its licence is, and who made it.

We also welcome improvements to existing flags. Most of our existing flags come from the Sodipodi clipart collection, and some of them are less than perfect. One common problem is that the colors are wrong. If you fix a flag for a real nation be sure to cite your source so we can be sure it is accurate. Good sources for nation flag data are Wikipedia or Flags Of The World.

If you want to improve an imaginary flag, this is also welcome. We recommend you first contact the original author of the flag to discuss your ideas for changes.

Flag Guidelines

Here are a few guidelines for flags:

- Flags should be rectangles, since an outline is added to them automatically.
- Flags often come in multiple aspect ratios. A 3:2 ratio looks best for Freeciv21 and currently every flag has this ratio. For a flag that is “supposed” to be 2:1 or 4:3, you can often find a 3:2 version as well.

Flag Specifics

To add a flag you will have to edit the following files:

- `data/tilesets/flags/<flagname>.svg`

Here is the SVG flag image. This is not used directly by Freeciv21 but is rendered into PNG files (at various resolutions for different tilesets). The SVG file is not used in Freeciv21, but all the other steps for adding flags are the same. The `<flagname>` should either be the name of the country that represents the flag, or the common name for the actual flag. When in doubt, use the same name as the name of the nation.

- `data/tilesets/flags/<flagname>.png`
- `data/tilesets/flags/<flagname>-shield.png`

These are the flag images that are used by Freeciv21. They are rendered from the SVG file. Once this file has been created it can be used with older versions of Freeciv21 as well. To run the conversion program you will need to install Inkscape, ImageMagick, and (optionally) pngquant.

- `data/misc/flags.spec`

This file has a reference to the flag PNG graphic. The “tag” here must match the flag tag you put in the nation ruleset file (usually `f.<flagname>`) and the “file” should point to the PNG image at `flags/<flagname>.png`.

- `data/misc/flags-large.spec`

Just like `flags.spec`, but large version of the graphics.

- `data/misc/shields.spec`

Just like `flags.spec`, this file must include a reference to the flag PNG graphic. The only difference is that the file should point to the “shield” graphic, `flags/<flagname>-shield.png`.

- `data/misc/shields-large.spec`

Just like `shields.spec`, but large version of the graphics.

Contents and Style

What Nations Can Be Added

A nation in Freeciv21 should preferably be a current independent country or a historical kingdom or realm. A nation that is currently governed by or the part of a greater political entity, or in other ways lacks complete independence could in most cases be made a Freeciv21 nation as well, but must never be listed as *modern* (see [Nation grouping](#) below.)

Copyrighted content may not be added unless full permission is granted by the holder of the copyright. This rule effectively disallows the inclusion of nations based on most literary works.

Nation Grouping

Freeciv21 supports a classification of nations in an unlimited number of groups and every nation should be assigned to at least one. We currently have ancient, medieval, early modern, modern, african, american, asian, european, oceanian and imaginary groups. Modern nations are existing and politically independent countries. A nation listed as ancient, medieval, or early modern should have had an independent dynasty or state in ancient (until 500 AD), medieval (500 - 1500), or early modern (1500 - 1800) times respectively. Finally, an imaginary nation is - as the name suggests - a product of someone’s imagination.

Nation Naming

The default name of the nation should be the name of the people, country, or empire in English adjective form. For example, the nation of ancient Babylon is called “Babylonian” in Freeciv21. The plural form should be standard English as well. For example, plural for the Polish nation is “Poles” in Freeciv21. UTF-8 is permitted in nation names.

Conflicting Nations

To specify one or more nations that the AI shouldn’t pick for the same game, use this syntax:

```
conflicts_with="<nationname>", "<nationname>", ...
```

You only have to specify this in the nation you are adding, since it works in both directions. Reasons for conflicting nations could be either that they represent the same people in different eras (example: Roman - Italian) or that the two nations have too similar flags that they are easily mixed up in the game (example: Russian - Serbian).

Civil War Nations

Specify one or more civil war nations. When a player's capital is captured, that player might suffer a civil war where his or her nation is divided and a new player created. The nation for this new player is selected from one of the civil war nations specified in the ruleset. A civil war nation should be linguistically, geographically, and/or historically related to the current nation. A linguistic relation is especially important, since city names after a nation run out of their own city names, are selected from the civil war nations' city lists.

Legend

A legend is required in a nation ruleset. The legend can be a summarized history of the nation, or just a piece of trivia. UTF-8 is permitted in legends.

Leaders

A leader should be a historically notable political leader of the nation. Two living persons per nation are permitted - one of each sex. An ideal leader list should contain between five and ten names. Use the person's full name to avoid ambiguity. Monarchs should be marked with the appropriate succession number, using Roman numerals in standard English style (not German e.g. "Otto II."; Hungarian e.g. "IV. Béla"; Danish e.g. "Valdemar 4." etc.)

Freeeciv21 supports any Unicode character, but please keep to Latin letters. When transcribing from a non-Latin writing system, be consistent about the system of transcription you are using. Also, try to avoid unnecessarily technical and/or heavily accented systems of transcription. Subject to the above, leaders should be written in native orthography, e.g. "Karl XII" instead of "Charles XII" for the Swedish king.

For consistency and readability, put only one leader per line. Feel free to provide a hint of the leader's identity or a brief background in a comment beside any leader: This information might be used in-game at a later stage.

Leader titles for each government type (including Despotism and Anarchy) may be specified in a separate tag. UTF-8 is permitted in leader titles. If the male and female titles are identical in English, give the latter the `?female:` qualifier. Use a unique title for each government. Ruler titles should be in English, though exceptions are made for non-English titles as long as they are understood outside of their own language regions and commonly used in non-academic contexts. Titles from the default ruleset may not be used.

Flag

You should provide a unique flag for your nation. Using a flag that is already used by another nation in the game is not acceptable. An alternative flag does not have to be specified.

Style

A nation must specify a default style. With the supplied rulesets each national style has direct relation to equivalent city style. The available city styles depends on the tileset used. Practically every tileset has four city styles: "European", "Classical" (Graeco-Roman style), "Asian" (Pagoda style) and "Tropical" (African or Polynesian style). In Amplio2 tileset, "Babylonian" and "Celtic" are also available. If the tileset used by a client does not support a particular city style, a fallback style is used. Selecting a style for your nation is not that strict. Just try to keep it somewhat "realistic."

Cities

As for the list of city names, you should make a clear decision about the type of the nation you add. An *ancient* or *medieval* nation may list any city that it at some point controlled. However if your nation is listed as *modern*, its city list must be restricted to cities within the country's current borders.

The reason for this is, we do not want Freeciv21 to be used as a political vehicle for discussions about borders or independence of particular nations. Another reason is to avoid overlapping with other nations in the game.

A city should appear in its native form, rather than Anglicized or Graeco-Roman forms. For example, the Danish capital is "København" rather than "Copenhagen", and the ancient Persian capital is "Parsa" rather than "Persepolis."

City names support any Unicode character, but please keep to Latin letters. When transcribing from a non-Latin writing system, be consistent about the system of transcription you are using. Also, try to avoid unnecessarily technical or heavily accented systems of transcription.

The ordering of cities should take both chronology of founding and overall historical importance into consideration. Note that a city earlier in the list has a higher chance of being chosen than later cities.

Natural City Names

Freeciv21 supports "natural" geographic placements of cities. Cities can be labeled as matching or not matching a particular type of terrain, which will make them more (or less) likely to show up as the "default" name. The exact format of the list entry is

```
"<cityname> (<label>, <label>, ...)"
```

where the cityname is just the name for the city (note that it may not contain quotes or parenthesis), and each "label" matches (case-insensitive) a terrain type for the city (or "river"), with a preceding ! to negate it. The terrain list is optional, of course, so the entry can just contain the cityname if desired. A city name labeled as matching a terrain type will match a particular map location if that map location is on or adjacent to a tile of the named terrain type; in the case of the "river" label (which is a special case) only the map location itself is considered. A complex example:

```
"Wilmington (ocean, river, swamp, forest, !hills, !mountains, !desert)"
```

will cause the city of Wilmington to match ocean, river, swamp, and forest tiles while rejecting hills, mountains, and deserts. Although this degree of detail is probably unnecessary to achieve the desired effect, the system is designed to degrade smoothly so it should work just fine.

Note: A note on scale: it might be tempting to label London as !ocean, i.e. not adjacent to an ocean. However, on a reasonably-sized Freeciv21 world map, London will be adjacent to the ocean. Labeling it !ocean will tend to give bad results. This is a limitation of the system, and should be taken into account when labelling cities.

At this point, it is useful to put one city per line, only. Finally, do not forget to leave a blank line feed in the end of your nation ruleset.

4.2 Tilesets

Freeciv21 allows full customization of the appearance of the map including terrain, cities, units, buildings, and a few elements of the user interface. Freeciv21 already ships with a variety of presets that can be selected from the menu by navigating to *Game* → *Load Another Tileset*.



Fig. 4.1: The same map with two tilesets: *hexemplio* (left) and *isophex* (right).

Nearly every aspect of the map rendering can be customized. In practice, this is achieved using a myriad of small images, called *sprites*, that are assembled together to form the final map. For instance, the units above are made of up to four sprites drawn on top of each other: the flag, the health bar, the yellow activity indicator, and finally the image that represents the unit itself. Customization is made possible thanks to a system of configuration files that specify where to find the sprites and how to assemble them.

Tilesets are a collection of *spec* files that fully define the look and feel of the game map, units, buildings, etc. This is effectively the graphics layer of Freeciv21. A tileset Modder can create a whole new custom graphics look and feel. The file layout for a tileset can vary depending on how the author wants to break out the varying layers. It will always start with a top-level *.tilespec* file and with a directory of the same name will have *.png* graphics files and associated *.spec* files to explain to Freeciv21 what to do when.

Have a look at *amplio2.tilespec* and associated files in */amplio2* for an example. The following guides document specific aspects of tileset creation:

4.2.1 Tileset tutorial

The tileset tutorial provides an introduction to the customization of graphics in Freeciv21. It assumes general knowledge of core game concepts such as units, cities, and terrain. It is also good to have at least a superficial understanding of how *rulesets* are written before starting this tutorial, because tilesets specify how to render on screen the objects defined in the ruleset.

Besides a working Freeciv21, the only thing you will need to complete the tutorial is a good text editor — graphic assets will be provided when needed. For Windows users, we recommend the excellent *Notepad++*; on Linux, a good editor like *Gedit*, *KWrite*, or *Mousepad* usually comes preinstalled. Do not attempt to use *Word* or *LibreOffice*.

The contents of the tutorial are summarized below. Beginners are encouraged to start from the beginning and read the first chapters in the order they are presented, because they follow a gradual approach and jumping right in the middle might be overwhelming. More advanced users may jump immediately to the chapter they are interested in, although we also encourage them to read everything.

First steps

This chapter will get you started by creating a new tileset for the tutorial, that you can edit safely.

Adding units

In this chapter, we add graphics for two fictitious units to our tileset.

First steps

When customizing graphics, the easiest is to start from an existing tileset and modify it. You should *never* modify directly the tilesets shipped by Freeciv21: first because if you break them, you have no “good” reference to compare to, and second because any update of Freeciv21 will overwrite your changes. Therefore, we will start this tutorial by duplicating the `amplio2` tileset into a new tileset called, very imaginatively, `tutorial`. In order to find the files, it is helpful to start a game and open *Help* → *About Freeciv21* from the menu. You will find there the list of directories in which Freeciv21 looks for its data files: one of them will contain a file called `amplio2.tilespec`.

Note: On Windows, the file explorer may not show the complete name of files by default. Because of this, `amplio2.tilespec` might appear as a file called `amplio2` with the description saying “TILESPEC file”. This can be changed in Windows Explorer by checking *View* → *Show/hide* → *File name extensions*. In the rest of this documentation, the complete name will always be used.

Once you have found the `amplio2.tilespec` file, copy it to another of the Freeciv21 folders (we suggest using a folder inside your home directory) and rename it to `tutorial.tilespec`. That is it: your new tileset can now be selected from the menu (you may need to reopen the settings window). For now, it is identical to the original `amplio2`; we will address this in a moment.

The `.tilespec` file contains some metadata that should be updated. To do this, open it in your text editor (see the [introduction](#) for suggestions). Then, find the following line:

```
name = "Amplio2"
```

Clearly, the name of the tileset should be changed to `Tutorial`. Near this line, you will also find a short description of the tileset:

```
summary = _("Large isometric tileset.")
```

The underscore and parentheses around the quotes indicate that this value can be translated, and you should leave them in place. However, the description itself could be improved: change it, for instance, to “Tileset from the Tileset Tutorial”. Once done, you can save the file and reload the tileset in Freeciv21; the default keystroke is `Control-Shift-F5`.

Note: You may find that syntax highlighting gives strange results with Freeciv21 files: when this happens, configuring your editor to use syntax highlighting for INI configuration files usually helps.

You may have noticed that we did not need to copy any image file in order to create the new tileset. Indeed, if you scroll down in `tutorial.tilespec`, you will find that it references files from `amplio2`:

```
files =
    "amplio2/terrain1.spec",
    "amplio2/maglev.spec",
    "amplio2/terrain2.spec",
    ...
```

The ability to reference files from another tileset is very useful to organize files. This is how, for instance, nation flags are imported: there is a central list that every tileset references. In the [next chapter](#) of this tutorial, you will learn how to new sprites.

Adding units

One of the most common changes done when developing custom rules (as described in the [Rulesets Overview](#)) is to add new unit types to the game. It is important to note that Freeciv21 has no prior knowledge of which unit types will exist, and much less of what they should look like. Some collaboration between the ruleset and the tileset is thus needed to provide this information. This is achieved using *graphic tags* in the rules; for instance, one could find the following in the definition of a new unit type, the Alien:

```
[unit_alien]
name      = _("?unit:Alien")
graphic   = "u.alien"
graphic_alt = "u.explorer"
...
```

The `graphic` directive above specifies the name that the unit has from the tileset point of view, `u.alien`. The second directive, `graphic_alt`, gives another name that can be used in case the tileset does not support the first one. In this case, it is set to `u.explorer`, which the author of the rules expects to be more widely available than `u.alien` — the Explorer is one of the units used in the default rulesets. It is also not completely wrong, since in these rules the aliens are quite peaceful. To add support for the Alien unit to our tileset, we thus need to provide the graphics for `u.alien`. For this, we will use the sprites below:

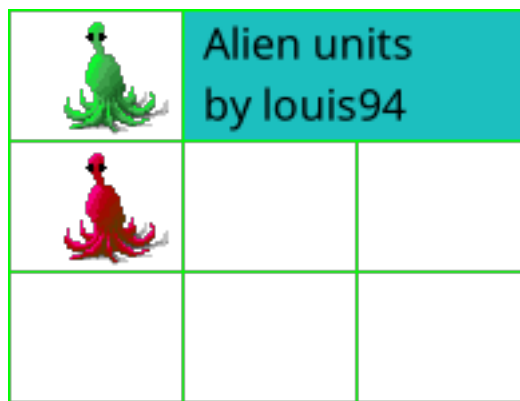


Fig. 4.2: Aliens. Aren't they cute?

Notice that a single image contains several sprites arranged in a grid. This makes it easier to send your units to friends, but is in no way mandatory: after all, a grid with a single cell is still a grid. One can adjust the width of the green lines to your liking or remove them, or even use several grids in a single image. All it takes is adjusting the configuration accordingly.

To add these sprites to your tileset, create a new folder `tutorial` next to the `tutorial.tilespec` that you created earlier. Download the following files and move them to the new folder:

- `aliens.png` — this is the same image as above.
- `aliens.spec` — we will go through it shortly.

The `aliens.spec` file tells Freeciv21 which sprites are in the image, and where in the grid. There is only one small change left to make the new sprites available in our tileset: we need to add the image to the tileset. To do this, open `tutorial.tilespec` and find the line with `files =` (around the half of the file). Insert the new `spec` file at the top of the list:

```
files =
  "tutorial/aliens.spec",
```

(continues on next page)

(continued from previous page)

```
"amplio2/terrain1.spec",
"amplio2/maglev.spec",
"amplio2/terrain2.spec",
"amplio2/hills.spec",
"amplio2/mountains.spec",
"amplio2/ocean.spec",
...
```

That is it! Your tileset will now display the Alien units for any ruleset that supports them — or rather, for any ruleset that uses the `u.alien` tag. Here is what it could look like:



Fig. 4.3: Alien invasion.

Let us now go through the newly added `aliens.spec` file. It can be opened with your text editor; go back to the *first page* if you do not know what to use.

You will notice that the file contains four sections, each opened by a name between square brackets: `[spec]`, `[info]`, `[file]`, and `[grid_main]`. The first section specifies the version of Freeciv21 supported by the tileset and will only change when moving from one version to the next. The `[info]` section is there solely to give credit to the artists. The last two are more interesting, so let's reproduce them here:

```
[file]
gfx = "tutorial/aliens"
```

This section contains the name of the image file described by the `spec`: one `spec` can only describe a single image file. Notice that the folder name, `tutorial`, is included, and that the `.png` extension is not given. Freeciv21 will figure automatically that it needs to add the `.png`. In theory, some other image formats could be used, but PNG is the only one fully supported on every system where Freeciv21 runs.

```
[grid_main]
x_top_left = 1
y_top_left = 1
dx = 64
dy = 48
pixel_border = 1

tiles = { "row", "column", "tag"
  0, 0, "u.alien"
  1, 0, "u.alien_angry"
}
```

The last section describes the layout of the sprites within the image. There is a single grid, called `main`, that starts one pixel from the top and one pixel from the left (`x_top_left` and `y_top_left`). This is because of the green outline around the file: it is not counted as being part of the grid. The next two lines specify the width (`dx`) and the height (`dy`) of the sprites in pixels. Then, we instruct Freeciv21 to skip one pixel between each sprite (`pixel_border`) to jump over the green lines separating them. With these values, Freeciv21 now has enough information to locate precisely every sprite in the grid.

The last four lines is the part that changes the most. It specifies where in the grid our two aliens are: the green one is in the first row from the top, which has index 0, and in the first column with index 0. We give the tag `u.alien` to this sprite to associate it with the corresponding unit in the ruleset. The red alien is in the second row and first column (1, 0: remember, it counts from zero), and we call it `u.alien_angry`.

A note on image editing

If you create tilesets, you will likely also want to edit graphics. We are often asked which painting program we use. We believe that the program that will work best for you depends a lot on your workflow, so we encourage you to try several of them and make your own opinion. If you don't know where to start, here are a few free (as in speech) recommendations:

- [Gimp](#) is a generic image editor with a focus on image edition. It also has great drawing capabilities.
- [Krita](#) is a digital painting application. It has a wider selection of brushes and presets than Gimp, and also supports vector layers.
- [MyPaint](#) is another program focused on digital painting, famous for its very flexible and powerful brush engine.

4.2.2 Tile Specification Files

Creating a Freeciv21 Tileset requires a mix of editing plain text `spec` files as well as creating graphics files saved in the `.png` image format. When complete, a Tileset editor will have created the canvas that Freeciv21 players play upon. Next to Rulesets, creating Tilesets is one of the most important aspects of a Freeciv21 game.

Using Graphics

To use different graphics with Freeciv21, use the `--tiles` argument to the Freeciv21 client. Eg, to use the 'engels' graphics, start the game as:

```
$ freeciv21-client --tiles engels
```

What Freeciv21 actually does in this case is look for a file called `engels.tilespec` somewhere in your Freeciv21 data path. That `.tilespec` file contains information telling Freeciv21 which graphics files to use, and what those graphics files contain.

If you do not want to use the command line to invoke Freeciv21, you can also load a different Tileset from the menu at *Game* → *Load Another Tileset*. The Longturn.net community also provides some custom Tilesets accessible from the modpack installer utility, documented in [Using the Modpack Installer Utility](#).

The rest of this file describes, though not fully, the contents of the `.tilespec` file and related files. This is intended as developer reference, and for people wanting to create/compile alternative tilesets and modpacks for Freeciv21.

Overview

The purpose of the `.tileset` file and related `.spec` files is to allow the detailed layout of the graphics within the files to be flexible and not hard-coded into Freeciv21, and to allow add-ons to conveniently provide additional graphics. The PNG graphic files are essentially a collection of arranged *sprites* combined together into a single PNG file. The `.spec` files then tell Freeciv21 how to process the sprites and display them in the game. The `.spec` files select which images are used for each Freeciv21 graphic, and the `.tileset` file controls which `.spec` files are used, and how the graphics will interact with each other.

There are two layers to the `.tileset` files:

- The top-level file is named, for example, `engels.tileset`. The basename of this file (here, ‘engels’) corresponds to the parameter of the `--tiles` command-line argument for Freeciv21, as described above.
- The top-level `.tileset` file contains general information on the full tileset, and a list of files which specify information about the individual graphics files. These filenames must be located somewhere in the data path. On Unix like operating systems this is `[install location]/share/freeciv21` and on Windows this is `[install location]/data`. Typically the second-level `.spec` and image files are in a sub-directory at the same level as the `.tileset` file. Note that with this system the number and contents of the referenced `.spec` and image files are completely flexible at this level. Here is an example file and folder view:

```
engels.tileset
engels/
  terrain1.spec
  terrain1.png
  terrain2.spec
  terrain2.png
  ...
```

An exception is that the intro graphics must be in individual files, as listed in the `.tileset` file, because Freeciv21 treats these specially. These graphics are freed after the game starts, and reloaded later as necessary.

Graphics Formats

Freeciv21 currently uses 24 or 32 bit PNGs image files. As noted before, the PNG files are a collection of images in a single image file. The smaller images are called *sprites*.

Tileset Options

In the top-level `.tileset` file you can set options for the tileset. Each of these should go within the `[tileset]` section. Currently options include:

Strings

String values are enclosed in quotes (" ")

- `options` : A capability string, this should be `+Freeciv-a.b-tileset`, where “a.b” it the current Freeciv21 version.
- `name` : The name of the tileset.
- `type` : General type of tileset, different types have quite different format. Supported types are “overhead” and “isometric”.
- `unit_default_orientation` : Specifies a direction to use for unit types in worklists etc. See [Unit Sprites](#) below.

String Vectors

- `preferred_themes` : List of preferred game themes to use with this tileset.

Integers

- `priority` : When user does not specify tileset, the game automatically loads available compatible tileset with highest priority.
- `preferred_scale` : The preferred scaling factor (zoom level), in percents.
- `normal_tile_width` : The width of terrain tiles.
- `normal_tile_height` : The height of terrain tiles.
- `unit_width` : Unit sprite width. Default is always ok, setting is provided just for symmetry with `unit_height`.
- `unit_height` : Unit sprite height if more than 1.5x terrain tile height in isometric tileset.
- `small_tile_width` : The width of icon sprites.
- `small_tile_height` : The height of icon sprites.
- `fog_style` : Specifies how fog of war is drawn.
 - `Auto` : Code automatically adds fog.
 - `Sprite` : A single fog sprite is drawn on top of all other sprites for fogged tiles. The `tx.fog` sprite is used for this.
 - `Darkness` : No fog, or fog from `darkness_style = 4`.
- `darkness_style` : Specifies how “encroaching darkness” is drawn.
 - `None` : No darkness.
 - `IsoRect` : A single sprite can be split into 4 parts, each containing the darkness for that particular cardinal direction. (Iso-view only.)
 - `CardinalSingle` : Four different sprites exist, each holding the darkness for a particular direction. Any or all of the sprites may be drawn.
 - `CardinalFull` : The sprite is chosen based on the vector sum of the darkness in all 4 cardinal directions. 15 different sprites are needed.
 - `Corner` : Corner darkness & fog, 81 sprites needed.
- `unit_flag_offset_x` : Gives an offset from the tile origin at which to...
- `unit_flag_offset_y` : Draw flags behind units or cities. With isometric...
- `city_flag_offset_x` : Tilesets this should be non-zero so that the flag...
- `city_flag_offset_y` : Is placed correctly behind the unit/city.
- `occupied_offset_x` : Gives an offset from the tile origin at which to...
- `occupied_offset_y` : Draw city occupied icon (in many tilesets placed above the flag).
- `city_size_offset_x` : Gives an offset from the full tile origin at which to...
- `city_size_offset_y` : Draw city size number.
- `unit_offset_x` : Gives an offset from the tile origin at which to...
- `unit_offset_y` : Draw units.
- `activity_offset_x` : Gives an offset from the tile origin at which to...

- **activity_offset_y** : Draw normal unit activity icons. “Auto” icons are not affected by this as they are usually wanted in different offset than real activity icons for both to appear simultaneously “Auto” icons are `auto_attack`, `auto_settler`, `patrol`, `connect`.
- **select_offset_x** : Gives an offset from the tile origin at which to...
- **select_offset_y** : Draw selected unit sprites.
- **unit_upkeep_offset_y** : Gives an offset from the unit origin at which to draw the upkeep icons when they are shown along the unit. The upkeep icons can safely extend below the unit icon itself. If this value is omitted, normal tile height is used instead;
 - Upkeep icons appear below the unit icon if the unit icons are equal to tile height (typical in overhead tileset)
 - Upkeep icons overlay lower part of the unit icon, if unit icon is higher than tile height (typical in iso tilesets)
- **unit_upkeep_small_offset_y** : Like **unit_upkeep_offset_y**, but to be used in case there is only small space for the overall icon produced. Defaults to **unit_upkeep_offset_y** — not having alternative layout.
- **citybar_offset_y** : Gives an offset from city tile origin at which to draw city bar text.
- **hex_sid** : When **is_hex** is specified (see **is_hex**, below), this value gives the length of the “extra” side of the hexagon. This extra side will be on the top/bottom of the tile if **is_isometric** (below) is given, or on the left/right of the tile otherwise. The actual dimensions of the hex tile are determined from the `normal_tile_width/normal_tile_height` of the tileset as well as the hex side. The “normal” dimensions give the X and Y offsets between adjacent tiles in the tileset — this is not the same as the dimensions of the tile itself. The dimension of the bounding box of the hexagonal tile will be equal to the “normal” dimension minus the `hex_side`. For instance, “normal” dimensions of 64x32 with a `hex_side` of 16 for an iso-hex tileset will give hexagons of size 48x32.

Booleans

Boolean values are either FALSE or TRUE.

- **is_hex** : Set to TRUE for a hexagonal tileset. If **is_isometric** is also specified then you have an iso-hex tileset. Hex tilesets should be used with topologies 8-11 and iso-hex tilesets with topologies 12-15.

String Lists

String lists are a comma-separated list of strings.

- **files** : A list of `.spec` files to scan for sprites. See “individual spec files”, below.

Extra Options

Tilespec should define style of extra graphics for each extra type in section `[extras]` like:

```
[extras]
styles =
{ "name",      "style"
  "road",      "RoadAllSeparate"
  "rail",      "RoadAllSeparate"
  "river",     "River"
  "tx.irrigation", "Cardinals"
}
```

- **RoadAllSeparate** : A single sprite is drawn for every connection the tile has. Only 8 sprites are needed.
- **RoadParityCombined** : A single sprite is drawn for all cardinal connections and a second sprite is drawn for all diagonal connections; 32 sprites are needed.

- **RoadAllCombined** : One sprite is drawn to show roads in all directions. There are thus 256 sprites (64 for a hex tileset).
- **River** : Cardinal connections are drawn, as well as delta at the coast
- **Single1** : Single sprite at layer **Special1**.
- **Single2** : Single sprite at layer **Special2**.
- **3Layer** : 3 Sprites, tagged `<name>_bg`, `<name>_mg`, and `<name>_fg`.
- **Cardinals** : Sprite for each cardinal connection.

Individual Spec Files

Each `.spec` file describes one graphics file as specified in the spec file. The graphics file must be in the Freeciv21 data path, but not necessarily in the same location as the `.spec` file. Note you can have multiple spec files using a single graphics file in different ways.

The main data described in the `.spec` file is in sections named `[grid_*]`, where `*` is some arbitrary tag (but unique within each file). A grid corresponds to a regular rectangular array of tiles. In general one may have multiple grids in one file, but the default tilesets usually only have one per file. Multiple grids would be useful to have different size tiles in the same file. Each grid defines an origin (top left) and spacing, both in terms of pixels, and then refers to individual tiles of the grid by row and column. The origin, and rows and columns, are counted as (0,0) = top left.

- **x_top_left** : X-coordinate of the leftmost pixel of the leftmost cell.
- **y_top_left** : Y-coordinate of the topmost pixel of the topmost cell.
- **dx** : Cell width.
- **dy** : Cell height.
- **pixel_border** : Number of pixels between cells, unless overridden by axis specific value.
- **pixel_border_x** : Number of pixels between cells in x-direction, overrides **pixel_border**.
- **pixel_border_y** : Number of pixels between cells in y-direction, overrides **pixel_border**.
- **tiles**: Table of tags, each line having “row”, “column”, and “tag”.

```
[grid_example]
x_top_left    = 1    ; Border (in x=0) also in left side of the entire grid
y_top_left    = 1    ; Border (in y=0) also in top side of the entire grid
dx            = 96
dy            = 48
pixel_border  = 1
tiles = { "row", "column", "tag"
0, 0, "tag1"
0, 1, "tag2"
1, 0, "tag3"
1, 1, "tag4"
}
```

Each individual tile is given a “tag”, which is a string which is referenced in the code and/or from ruleset files. A grid may be sparse, with some elements unused (simply do not mention their row and column), and a single tile may have multiple tags (eg, to use the same graphic for multiple purposes in the game): just specify a list of comma-separated strings.

If a given tag appears multiple times in the spec files, the *last* such tag is used. That is, in the order of files listed in the tilespec file, and order within each file. This allows selected graphics to be “overridden” by listing a replacement spec file near the end of the ‘files’ list in the top-level tilespec file, without having to modify earlier files in the list.

Tag Prefixes

To help keep the tags organised, there is a rough prefix system used for standard tags:

- **f.** : National flags.
- **r.** : Road/rail.
- **s.** : General “small”.
- **u.** : Unit images.
- **t.** : Basic terrain types (with `_n0s0e0w0` to `_n1s1e1w1`).
- **ts.** : Terrain special resources.
- **tx.** : Extra terrain-related.
- **gov.** : Government types.
- **unit.** : Unit overlays: hp, stack, activities (goto, fortify etc.).
- **upkeep.** : Unit upkeep and unhappiness.
- **city.** : City related (city, size, sq.-prod., disorder, occupied).
- **cd.** : City defaults.
- **citizen.** : Citizens, including specialists.
- **explode.** : Explosion graphics (nuke, units).
- **spaceship.** : Spaceship components.
- **treaty.** : Treaty thumbs.
- **user.** : Crosshairs (in general: user interface?).

In general, graphics tags hard-wired into Freeciv21 **must** be provided by the `.spec` files, or the game will refuse to start. Graphics tags provided by ruleset files (at least for the shipped rulesets) should also be provided, but generally the game will continue even if they are not, though the results may not be satisfactory for the user. To work properly tags should correspond to appropriately sized graphics. The basic size may vary, as specified in the top-level `.tilespec` file, but the individual tiles should be consistent with those sizes and/or the usage of those graphics.

Sprites

Depending on the information given here the tileset must/may contain certain sprites.

Theme Sprites

Citizen Sprites

This provides citizen graphics. Each citizen has one or more sprites which are shown in the *City Dialog*. The types of citizen are “happy”, “content”, “unhappy”, and “angry”. The tag name is `citizen.<type>_<n>`. `<type>` is one of the listed types. `<n>` is the number of the graphic (numbered starting with 0, unlike most other graphics) which allows more than one sprite to be used. No more than 6 sprites per citizen may be used.

Currently the citizen and specialist sprites may not have any transparency, as this is ignored in much of the drawing. This is considered a bug.

Specialist Sprites

These provide specialist graphics just like the citizen graphics. However, specialist types come from the ruleset and may be changed in modpacks. The sprite name is `specialist.<type>_<n>`. Again `<type>` is the type of specialist (currently “elvis”, “scientist”, “taxman”) while `<n>` is the sprite number. See “citizen sprites” above.

Progress Indicators

There are three types of progress indicator. `science_bulb` indicates progress toward the current research target. `warming_sun` indicates progress toward global warming. `cooling_flake` indicates progress toward nuclear winter. Each indicator should have 8 states, numbered 0 (least) through 7 (most). The sprite names are `s.<type>_<n>`.

Government Icons

There should be one icon for each government. Its name is `gov.<gov>`, where `<gov>` is the government name. Government types come from `governments.ruleset` (currently “anarchy”, “despotism”, “monarchy”, “communism”, “fundamentalism”, “republic”, “democracy”). Ruleset modders can create other governments, the aforementioned list is not static.

Tax Icons

One icon for each tax type. These are used to show the national budget. The sprites are `s.tax_luxury`, `s.tax_science`, `s.tax_gold`. Commonly the specialist sprites are reused for this.

Right Arrow

A sprite `s.right_arrow` is used on the panel when more units are present than can be shown.

Terrain Special Sprites

Farmland/Irrigation

`tx.farmland` and `tx.irrigation` provide the basic sprites for farmland and irrigation. Additionally, there is support for drawing continuous farmland and irrigation (as is used in Civ3). Here there are 16 irrigation sprites (and the same for farmland), starting with `tx.irrigation_n0s0e0w0` and running through `tx.irrigation_n1s1e1w1`. An appropriate sprite will be chosen depending on which adjacent tiles also have farmland/irrigation. If any of these sprites are not present, the default sprite will be used as a fallback.

Unit Sprites

Units sprites can be either unoriented or oriented, in which case the sprite that is displayed depends on the direction the unit is facing (it turns when it moves or fights).

Unoriented sprites are specified as `u.phalanx`. Oriented sprites have a direction suffix: `u.phalanx_s`, `u.phalanx_nw` and so on. For each unit type, either an unoriented sprite or a full set of the oriented sprites needed for the tileset topology must be provided. You can also provide both, see below.

The game sometimes needs to draw a sprite for a unit type that does not correspond to a specific unit, so is not facing a particular direction. There are several options for oriented tilesets:

- If the `unit_default_orientation` is specified for the tileset, the game will by default use that directional sprite. The direction does not have to be a valid one for the tileset.
- Specific unit types may override this by providing an unoriented sprite as well as the oriented ones; this does not have to be distinct, so it can point to one of the oriented sprites, allowing choice of the best orientation for each individual unit type. If `unit_default_orientation` is not specified, an unoriented sprite must be specified for *every* unit.

4.2.3 Terrain

Drawing terrain with realistic (or not) transitions between adjacent tiles is a complex problem. Freeciv21 comes with several patterns that can be used to build complex terrain schemes. They are specified in the main `.tilespec` file of the tileset, under the `[tile_*]` and `[layer*]` sections.

Terrain rendering is done in several independent passes, each of them having its own layer in the main tileset configuration file, `Terrain1` through `Terrain3`. This allows several sprites to be combined in the drawing of a single tile, which is one of the ways to break monotony.

Each terrain specified in a ruleset must have a `graphic` tag name, and optionally a `graphic_alt` tag. This information is used to search for compatible sprites in the tileset, first using `graphic` and, if it fails, `graphic_alt`.

Todo: Documentation is still missing for:

- Sprite dimensions, in particular in hexagonal mode where it is counter-intuitive.
 - Blending.
 - The mask sprite and what happens at the edge of the map.
 - A general introduction to the structure (`[layer*]` and `[tile_*]` sections). Some of this is already there at the bottom of but would need reformatting.
-

Sprite tags

The names used to reference the sprites depend on the chosen cell type and on the number of groups included in the matching. The table below summarizes the naming scheme; available options are discussed in detail in the corresponding sections.

Table 4.4: Terrain Matching Groups

Matched groups	single	corner	hex_corner
None	<code>t.l0.grassland1</code> (<i>doc</i>)	Avoid (<i>doc</i>)	Avoid
One, same as match_type	<code>t.l1.hills_n0e1s0w1</code> (<i>doc</i>)	<code>t.l0.floor_cell_u011</code> (<i>doc</i>)	<code>t.l0.floor_hex_cell_right_0_1_0</code> , <code>t.l0.floor_hex_cell_left_1_0_0</code> (<i>doc</i>)
One, different from match_type	Not implemented	<code>t.l1.coast_cell_u_g_g</code> (<i>doc</i>)	Not implemented
Two or more		<code>t.l0.cellgroup_g_g_g_g</code> (<i>doc</i>)	<code>t.l0.hex_cell_right_g_g_g</code> , <code>t.l0.hex_cell_left_g_g_g</code> (<i>doc</i>)

Sprite type single

In this mode, each tile is drawn using a single sprite. The sprites should have dimensions `normal_tile_width` times `normal_tile_height`. It is possible to augment the size by setting `layerN_is_tall` to `TRUE`, in which case the height is expanded by 50% above the tile. This can be used to render graphical elements like trees and mountains that hide terrain behind them.

It is possible to set arbitrary offsets on a per-terrain basis using `layerN_offset_x` (positive values move the sprite to the right) and `layerN_offset_y` (positive values move the sprite down). These options should be used with caution, because pixels drawn outside of the area covered by a “tall” tile will confuse the renderer and cause artifacts.

Note: `whole` is a synonym for `single`; `single` is preferred.

Without matching

The name of the sprites used by sprite type `single` depend on the number of terrain groups included in `matches_with`. When no matching is performed, sprites names are built according to the following pattern:

```
t.l<n>.<tag><i>
```

The value `<n>` is replaced with the layer number, and `<tag>` with the terrain tag. The last element, `<i>`, is a number starting from 1: if several sprites are provided with numbers 1, 2, ..., the renderer will pick one at random for every tile. This can be used to provide some variation, either by changing the base terrain sprite or by overlaying decorations on top.

Example

The following is the minimal definition for a terrain type: no matching is performed, and a single sprite is sufficient:

```
[tile_desert]
tag = "desert"
num_layers = 1
```

The base sprite would have tag `t.l0.desert1`; additional sprites called `t.l0.desert2`, `t.l0.desert3`, etc., can also be added, in which case one will be picked at random for every tile.

With matching

Sprite type `single` also supports matching against the *same* group as the represented terrain is in. For instance, if one group is used for land, a second group for sea tiles, and a third group for ice, the sprite used for ice tiles can depend on the presence of ice on adjacent tiles — but when there is no ice, one cannot know whether the other tile is land or water. In this case, the pattern is as follows:

```
t.l<n>.<tag>_<directions>
```

Like in the unmatched case, `<n>` is replaced with the layer number and `<tag>` with the terrain tag. The `<directions>` part indicated which in which directions a match has been achieved, as a list of directions followed by `0` (no match) or `1` (match). The directions depend on the tileset geometry:

- For square tilesets, they are North, East, South, and West, and thus the `<directions>` part looks like `n0e1s1w0`. There are 16 sprites in total.

- Isometric hexagonal tilesets also have South-East and North-West, and the <directions> part looks like `n0e1ne0s1w0nw0`. There are 64 sprites.
- Non-isometric hexagonal tilesets use North-East and South-West instead, for instance `n0ne0e1s1sw1w0`. There are also 64 sprites.

Matching Example

In many tilesets, the sprites used for hills and mountains depend on the presence of other hills and mountains on adjacent tiles. This is achieved by putting them in a single matching group, usually called `hills`:

```
[layer1]
match_types = "hills"
```

We use layer 1 in this example because something is typically drawn under the hills for coasts and blending. The next step is to put hills and mountains in the group and enable matching:

```
[tile_hills]
tag = "hills"
num_layers = 2
layer1_match_type = "hills"
layer1_match_with = "hills"

[tile_mountains]
tag = "mountains"
num_layers = 2
layer1_match_type = "hills"
layer1_match_with = "hills"
```

With these settings, both hills and mountains will match adjacent tiles if they have hills or mountains.

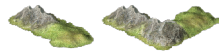


Fig. 4.4: Hills and mountains in `amplio2` use the pattern described above.

Sprite type corner

The `corner` sprite type divides each tile in four smaller parts that are adjacent to only three tiles. This allows matching with diagonal tiles, which would be impractical in `single` mode due to the large number of sprites required. Corner mode was developed primarily for square isometric tilesets, but it can also be used with other topologies, as shown in the diagram below:

For square topologies, the corner sprites (colored rectangles) cover a slice of the tile area adjacent to three other tiles. Matching takes place with respect to each of them, which enables complicated designs while requiring comparatively small numbers of small sprites. For isometric hexagonal tilesets, some slices are adjacent to two tiles and some to three; it is recommended that new tilesets use *hex_corner* instead.

The four corners are identified using the letters shown in the diagram, which stand for their location in isometric mode: up, down, left, and right. The names used in the `.spec` files depends on the number of groups listed in `matches_with` and, when a single group is listed, of the group matching is performed against. This naming scheme is explained in the next sections.

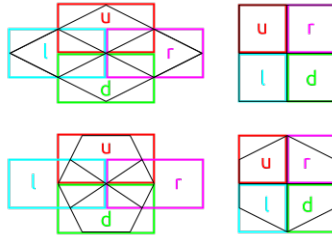


Fig. 4.5: Definition of the corners for the four tileset topologies: square isometric (top left), square (top right), hexagonal isometric (bottom left), and hexagonal (bottom right).

Without matching (single)

Warning: Using corner sprites without matching is fully equivalent to a `single` sprite, except that performance is likely to be worse and the naming convention is harder to track. Avoid using this mode.

When no matching is performed, four `corner` sprites are required for each terrain. They are expected to be half the size of a normal tile in both dimensions, and use the following naming scheme:

```
t.l<n>.<tag>_cell_<direction>
```

The value `<n>` is replaced with the layer number, and `<tag>` with the terrain tag. The last part, `<direction>`, indicates which corner the sprite refers to.

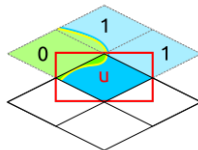
Matching with the same group

This mode is used when a single matching group is specified in the `matches_with` list, and it is the same as `match_type`. This is often used to draw beaches, because they are drawn where neighboring tiles are anything but water. 32 sprites are required for each tag, with the following naming convention:

```
t.l<n>.<tag>_cell_<direction><01><01><01>
```

The value `<n>` is replaced with the layer number, and `<tag>` with the terrain tag. Sprites must be provided for each of the four possible values of `<direction>`: `u`, `d`, `l`, and `r`, that indicate which corner the sprites are for. The three remaining parts, `<01>`, each correspond to the matching status of one of the adjacent tiles, counting clockwise. `0` means that the tile is not matched, and `1` that it is.

For instance, the suffix of `u011` corresponds to the following situation, where blue represents the group of the tile being rendered (black frame) and green is some other terrain:



Group Example

Simple coasts can be drawn as follows:

```
[layer0]
match_types = "water"

[tile_coast]
tag = "coast"
num_layers = 1
layer0_match_type = "water"
layer0_match_with = "water"
layer0_sprite_type = "corner"

[tile_floor]
tag = "floor"
num_layers = 1
layer0_match_type = "water"
layer0_match_with = "water"
layer0_sprite_type = "corner"

[tile_lake]
tag = "lake"
num_layers = 1
layer0_match_type = "water"
layer0_match_with = "water"
layer0_sprite_type = "corner"
```

This requires 96 sprites, 32 for each tile type.

Matching a pair of groups

This mode is used when a single matching group is specified in the `matches_with` list, and it is different from `match_type`: a neighbor tile matches only if it is in the specified group. This can be used in a similar role as *matching with the same group*, but is sometimes more convenient (especially when a layer starts to have many groups). This mode requires 32 sprites per tag and uses the following naming convention:

```
t.l<n>.<tag>_cell_<direction>_<g>_<g>_<g>
```

The value `<n>` is replaced with the layer number, and `<tag>` with the terrain tag. Sprites must be provided for each of the four possible values of `<direction>`: `u`, `d`, `l`, and `r`, that indicate which corner the sprites are for. The three remaining parts, `<g>`, each correspond to the first letter of a matching group of one of the adjacent tiles, counting clockwise. If there was a match, the first letter of the group in `matches_with` is used; otherwise, it is the first letter of `match_type`.

Warning: Extra care is needed when drawing sprites for this mode; see the example for guidance.

Groups Example

Suppose that you have a tileset where mountains are drawn as solid rock. It would then make sense to draw cliffs instead of beaches where the mountains meet water, as below:

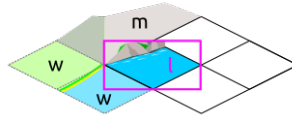


Fig. 4.6: Cliffs

This can be achieved by drawing the mountains and the sea normally in the first layer, and overlaying the cliffs in the second layer. In this example, the cliffs are drawn on top of the water (the mountains advance into the sea):

```
[layer2]
match_types = "water", "mountains"

[tile_coast]
tag = "coast"
num_layers = 2
layer1_match_type = "water"
layer1_match_with = "mountains"
layer1_sprite_type = "corner"

[tile_mountains]
tag = "mountains"
num_layers = 1
layer1_match_type = "mountains"
```

The sprite shown above would be called `t.11.coast_cell_1_w_w_m` (left side, water, water, and mountains when enumerating clockwise): even though the tile on the left is not water, it is still identified as such because it is not in the group given in `match_with`.

Because the tile on the left is identified with water, there is no way to distinguish between the following situations:

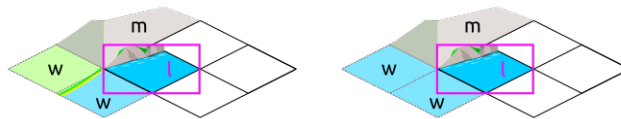
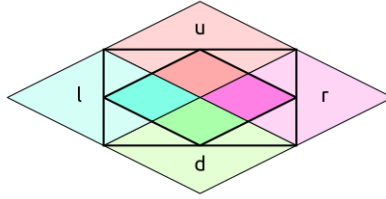


Fig. 4.7: Indistinguishable cases when using pair matching.

Because of this, sprites need to be designed to work in several cases (the tile at the bottom could also be either land or water). In the example above, the cliff vanishes at the corner, which allows it to merge with the land and is also a plausible behavior when there is only water around.

General matching

When more than one element is present in the `matches_with` list, general matching is used. This mode uses sprites that cover the intersection between four tiles:



The sprites have the same size as a normal tile, but are drawn with an offset equal to one half of a tile, such that they are centered around the meeting point of the tiles.

The sprite naming convention uses only the names of the four groups the tiles are in. Unlike with other modes, the terrain tag is not used:

```
t.l<n>.cellgroup_<g>_<g>_<g>_<g>
```

The value `<n>` is replaced with the layer number. The four remaining parts, `<g>`, each correspond to the first letter of one of the groups specified in `matches_with`, specified in clockwise order starting from top (referring to the above schema, u, r, d, and l).

Note: General matching is a very flexible mode that lets one draw very complex terrain, but this comes at the cost of a large number of sprites: for three groups, 81 sprites are needed; for four groups, it raises to 256; and to use four groups, one would need to draw 625 sprites.

Sprite type `hex_corner`

New in version 3.0: Use the `+hex_corner` option in tilesets requiring this feature.

New in version 3.1: Support for matching against the group the tile is in.

The `hex_corner` sprite type provides functionality similar to `corner`, using a geometry optimized for isometric hexagonal tilesets. Hexagonal corner sprites cover one half of the height of the hexagons and are centered vertically within the tiles. They come in two types: “left” corners cover the left hand side of an hexagon and the right hand side of the border between two others; “right” corners have a similar geometry, but are flipped horizontally. When drawn in a checkerboard pattern, left and right sprites reconstruct the complete hexagons.

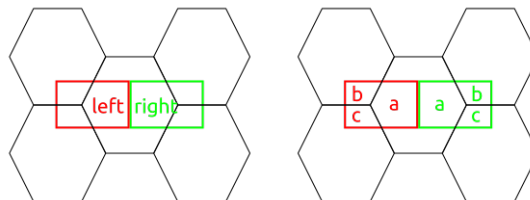


Fig. 4.8: The geometry of hexagonal corner sprites.

Matching with the same group (corner)

Similarly to *the corresponding mode for square tilesets*, this mode is used when `matches_with` only contains `match_type`. 14 sprites per terrain tag are needed to cover all the variations. They are named as follows:

```
t.l<n>.<tag>_hex_cell_left_<01>_<01>_<01>
```

For “right” sprites, simply replace `left` with `right`. The value of `<n>` gives the layer number, and the three `<01>` indicate whether the three tiles around the corner are in the specified matching group. The order is given by the letters a, b, and c in the figure above.

Matching with more than one group

This corresponds to *the general mode for square tilesets*. The naming convention for “left” sprites is as follows:

```
t.l<n>._hex_cell_left_<g>_<g>_<g>
```

For “right” sprites, simply replace `left` with `right`. The value of `<n>` gives the layer number, and the three `<g>` each correspond to the first letter of a matching group. For “left” sprites, the first group corresponds to the tile of the right, the second to the tile at the top left, and the third group is the one of the tile at the bottom left. For “right” sprites, the tile on the left comes first, followed by the one at the top right and the tile at the bottom right. The order is indicated by the letters a, b, and c in the figure above.

Multiple Group Example

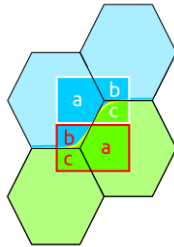


Fig. 4.9: Coasts using `hex_corner` sprites.

To draw coasts using `hex_corner`, one starts by defining two matching groups `land` and `water`:

```
[layer0]
match_types = "land", "water"
```

Each land terrain must be declared within the `land` matching group, while seas and lakes go to `water`:

```
[tile_coast]
tag = "coast"
num_layers = 1
layer0_match_type = "water"
layer0_match_with = "land", "water"
layer0_sprite_type = "hex_corner"

[tile_plains]
```

(continues on next page)

(continued from previous page)

```

tag = "plains"
num_layers = 1
layer0_match_type = "land"
layer0_match_with = "land", "water"
layer0_sprite_type = "hex_corner"

; etc

```

With these settings, the two sprites shown in the figure are called `t.10.hex_cell_right_w_w_1` for the one above (white), and `t.10.hex_cell_left_l_w_1` for the one below (red).

Terrain Options

The top-level `.tilespec` file also contains information on how to draw each terrain type (grassland, ocean, swamp, etc.). For each terrain type include a section `[tile_xxx]`. This section contains information on how to draw this terrain type. The terrain types are specified in the server `ruleset` file.

`[tile_XXX]` options:

- **tag** : Tag of the terrain this drawing information refers to. That must match the “graphic” or “graphic_alt” field given in the ruleset file.
- **blend_layer** : If non-zero, given layer of this terrain will be blended with adjacent terrains. Blending is done civ2-style with a dither mask. Only iso-view currently supports blending. Only the base graphic will be blended. The blending mask has sprite `t.dither_tile`.
- **num_layers** : The number of layers in the terrain. This value must be 1, 2 or 3. Each layer is drawn separately. The `layerN` options below control the drawing of each layer (N should be 0, 1 or 2)
- **layerN_match_type** : If 0 or unset, no terrain matching will be done and the base sprite will be drawn for the terrain. If non-zero, then terrain matching will be done. A matched sprite will be chosen that matches all cardinally adjacent tiles whose terrain has the same `match_type`.
- **layerN_match_with** : List of `match_types` to match against
- **layerN_sprite_type** : With traditional tilesets each tile is drawn using one sprite. This default `sprite_type` is “whole”. Which sprite to use may be specified using a `match_group`, and there may be multiple layers (each having one sprite). This method corresponds to `sprite_type` “single”. A more sophisticated drawing method breaks the tile up into 4 rectangles. Each rectangular cell is adjacent to 3 different tiles. Each adjacency is matched, giving 8 different sprites for each of the 4 cells. This `sprite_type` is “corner”.

Additionally the top-level `.tilespec` file should contain information about the drawing of each layer. This is needed because the way each layer is drawn must be consistent between different terrain types. You may not have more than 3 layers (either in this section or in the `[tile_XXX]` sections).

`[layerN]` Options:

- **match_types** : Gives a string list of all different match types. This list must include every possible `match_type` used by terrains for this layer. First letter of the `match_type` must be unique within layer.

4.2.4 Tileset Options

It is often possible to introduce slight variations in a tileset. One may have, for instance, multiple versions of unit sprites. In such cases, it may make sense to let the user choose between the two options. This is enabled by tileset options, which provide a list of parameters that the user can control under *Game* → *Tileset Options* in the *main menu*.

Options are always very simple yes/no questions. Tilesets can configure which questions are asked and, most importantly, load different sprites depending on the answer.

since
3.1

Adding Options

Each option is added as a section in the main `tilspec` file of the tileset. The section title must start with `option_` as this is how the engine recognizes that it defines an option. Each option must be given a name (which will be used when selecting sprites, see below), a one-line description for the user, and a default value. The following example can be used as a starting point:

```
[option_cimpletoon]
name = "cimpletoon"
description = _("Use 3D Cimpletoon units")
default = FALSE
```

This defines an option called `cimpletoon` that will be shown to the user as “Use 3D Cimpletoon units” and will be disabled when the tileset is used for the first time.

Sprite Selection

When specifying sprites for use in the tileset, option support is enabled by an additional attribute called `option`. If the attribute is not empty, it specifies the name of an option used to control the loading of the sprite, and the sprite is only loaded when the option is enabled. A simple example could be as follows:

```
tiles = { "row", "column", "option",    "tag"
          0,     0,     "",           "u.settlers"
          0,     1,     "cimpletoon", "u.settlers"
}
```

Let us decompose how this works. Regardless of whether the `cimpletoon` option is enabled, the sprite at row 0 and column 0 is always specified for Settlers units. However, when the `cimpletoon` is enabled, the second line overrides the first and the alternative sprite in column 1 is used. The end result is that the user sees a different sprite depending on the state of the option.

In the example above, the “main” and “optional” sprites are located in the same grid. This is not required and they can very well belong to different files. Remember, however, that the alternative must come *after* the main sprite in order to override it when the option is enabled.

Tileset options are very flexible and the changes they control can range from small tweaks involving a couple of sprites to complete overhauls that change the appearance of the map completely. The main limitation is that map geometry cannot change, as tiles have the same dimensions regardless of which options are enabled.

4.2.5 Tileset Debugger

New in version 3.0-alpha6.

The Tileset Debugger, accessible from the *Game* menu, lets you inspect how the map is drawn. This is very helpful when developing a tileset, to understand why something is rendering incorrectly or to understand how other tilesets work. The debugger window contains two tabs: *Messages* and *Inspector*. They are documented below.

Attention: The Tileset Debugger is still a work in progress. If you have suggestions regarding its contents and functionality, you are very welcome to let us know on [GitHub](#) — you will get a chance to shape it to your needs.

Changed in version 3.0-beta1: Added messages.

Messages

When loading a tileset, Freeciv21 will emit messages for sprites it cannot find or other errors. The *Messages* panel displays the messages for the current tileset. They are grouped in three categories, indicated by a small icon: information, warning and error messages. Errors are the most important: they indicate that something went seriously wrong when loading the tileset, and you should check it. Warnings are emitted for instance when the configuration is likely incorrect, or a feature in use will change in a future version. They may also help understand what caused an error. Information messages may give tips or provide context about the steps involved in loading a ruleset, and can usually be ignored.

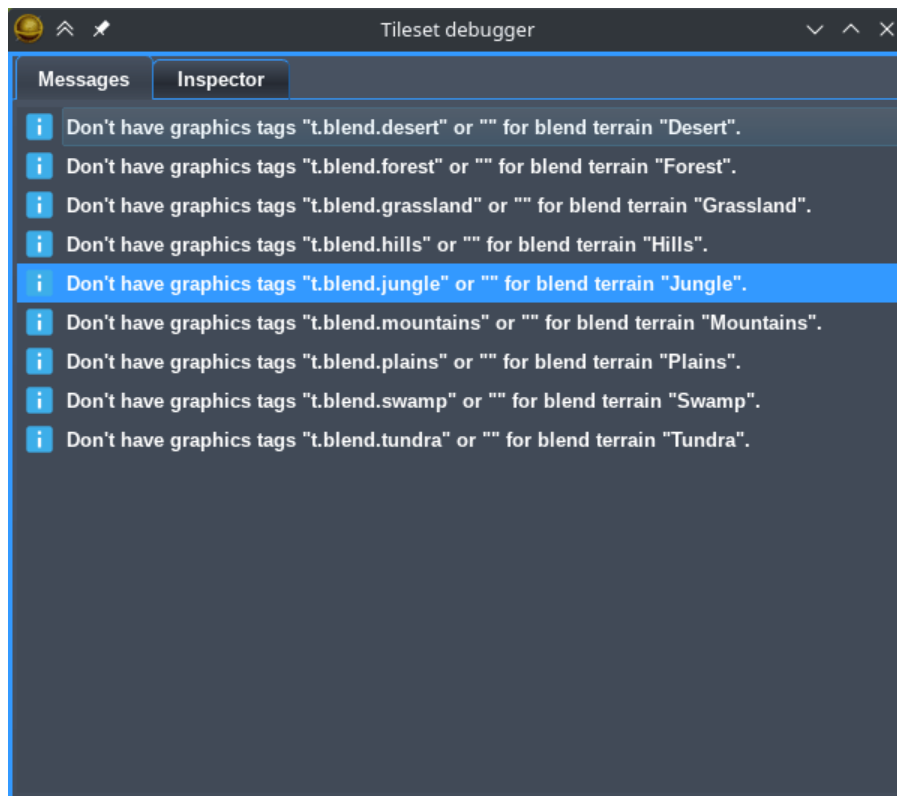


Fig. 4.10: The Tileset Debugger showing information messages from a successfully loaded tileset.

Inspector

The inspector can break out a tile into the basic sprites used to draw it. Start by clicking on the *Pick tile* button and then somewhere on the map. The window will be updated with the list of sprites used to draw the selected tile:

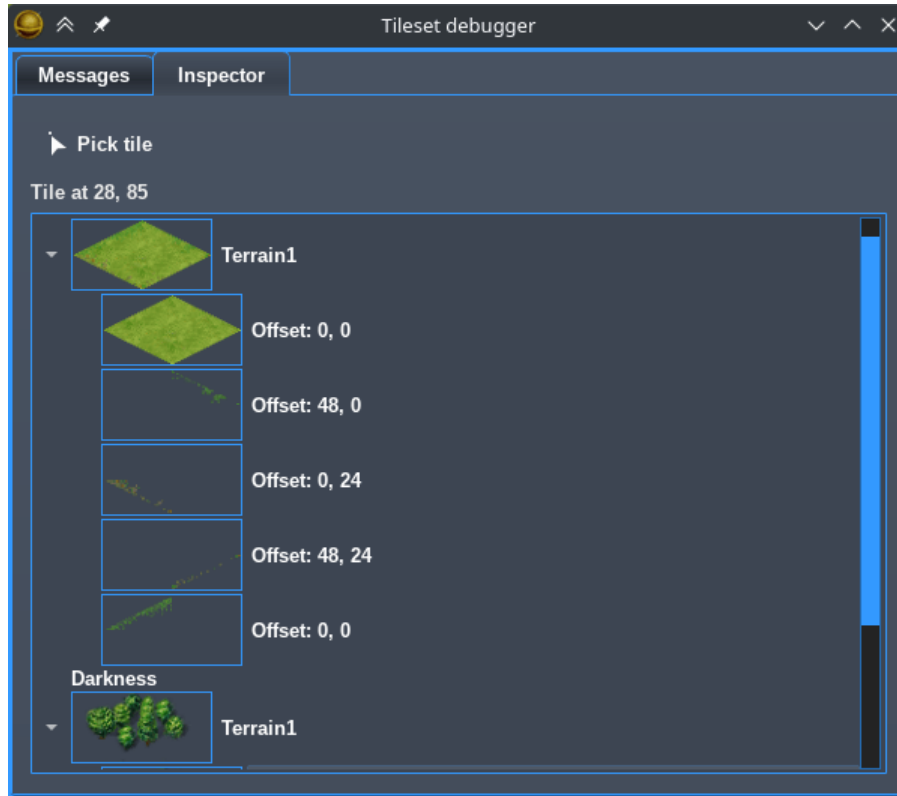


Fig. 4.11: The Tileset Debugger with a forest tile picked up.

The list has two levels. Each top level item corresponds to one layer used to draw the map. When something is drawn for a layer, its image is added next to its name and the individual sprites are added in the second level. The sprites at the top of the list are drawn first and are hidden by the ones below. Note that at the moment, only sprites that correspond to the tile (as opposed to its corners and edges) are visible.

In the picture above, which uses the *amplio2* tileset, three layers of a forest tile are shown, two of which have sprites: a terrain layer (**Terrain1**), **Darkness**, and another terrain layer (also listed as **Terrain1**). The first terrain layer is made of five sprites: one for the base texture and four that blend it with adjacent tiles. The second terrain layer has only one sprite, used to draw the trees. The offsets used to draw the sprites are also shown. The first number corresponds to the horizontal axis and runs from left to right. The second corresponds to the vertical axis that runs from top to bottom. Depending on the type of layer, these values may be computed automatically, so they do not necessarily correspond to parameters in the *tilspec* file.

4.2.6 Tileset Compatibility

The tileset format evolves when new Freeciv21 releases are published. As a rule of thumb, we try to maintain compatibility with tilesets designed for previous releases, or at least to make the update as straightforward as possible. When new features are introduced, we provide a flag that lets tilesets require them: a tileset using such a flag will not load in versions of Freeciv21 that do not have the feature. This lets tileset authors request the exact features needed by their tileset.

Feature flags are requested by adding them to the `options` string in the main `.tilespec` file. For instance, the following options require `precise-hp-bars` in addition to the main Freeciv21 capability:

```
options = "+Freeciv-tilespec-Devel-2019-Jul-03 +precise-hp-bars"
```

Notice the `+` in front of the flag name.

The rest of this page lists the available flags and their meaning.

duplicates_ok

When a graphics tag is specified appears several times, the lattermost tag is used.

precise-hp-bars

Unit *HP* bars with a precision different from 10% can be used (sprites `unit.hp_*`). The sprites must still be equally spaced: for instance, providing the following set of sprites will work as expected: `unit.hp_0`, `unit.hp_25`, `unit.hp_50`, `unit.hp_75`, `unit.hp_100`. On the other hand, specifying only the following sprites will give wrong results: `unit.hp_0`, `unit.hp_90`, `unit.hp_100`.

unlimited-unit-select-frames

The number of sprites used in the animation under the selected unit is no longer fixed to four (sprites `unit.select*`). Also introduces the setting `select_step_ms`.

unlimited-upkeep-sprites

The number of sprites used to display unit upkeep is no longer limited to 10 (sprites `upkeep.unhappy*`, `upkeep.output*`).

hex_corner

Support for this option signals the availability of the new `hex_corner` sprite type for terrain.

terrain-specific-extras

Terrain-specific extra sprites allow for different sprites to be used depending on the terrain. Currently, only rivers/roads/rails/maglev sprites may be made terrain-specific, but compatibility for specials and bases is expected to roll out soon. To use terrain-specific sprites, use the same nomenclature as before, but with the terrain name included after the type of extra. E.g. `road.road_isolated` becomes `road.road_mountain_isolated`.

options

Support for this option signals that *Tileset Options* are available.

4.2.7 Tileset Breaking Changes

This page lists in reverse chronological order breaking changes merged into the `master` branch of the GitHub repository to aid Tileset modders keep their tilesets up to date.

Post Beta.4 before Beta.5

- Removed the need for `main_intro_file` for all `*.tilespec` files. Tileset modders no longer need to include this tag in their tilesets.
- Wonder sprites removed from `data/misc/buildings.spec` and moved into `data/misc/wonders.spec`. Tileset modders will need to include a new file to get the small wonder sprites.
- Removed `data/amplio2/units.spec`. Consolidated all unit sprites into `data/misc/units.spec`. Tileset modders utilizing the Amplio2 unit sprites will want to instead use the new `units.spec` file. It is the new home of the Amplio style units utilized by the Tier 1 tilesets in Freeciv21: Amplio2 and Hexemplio. Old Amplio is no longer shipped with Freeciv21 and can be downloaded with the modpack installer.

4.2.8 Units

In addition to the unit sprite itself, the following are drawn:

- For selected units, an animated selection indicator is displayed underneath the unit. Optionally, the selected units can blink instead. *select_offset_x* and *select_offset_y*.
- The unit flag is also drawn under the unit picture (unless Freeciv21 is configured to display a solid color behind units, in which case that color is shown instead). *unit_flag_offset_x*, *unit_flag_offset_y*.
- Then, the sprite for the unit type is added. It can depend on the unit orientation. *unit_offset_x* and *unit_offset_y*.
- Loaded transports are marked as such.
- A sprite is added to represent the unit activity. *activity_offset_x* and *activity_offset_y*.
- Units on `autosettler` and `autoexplorer` commands are tagged. *activity_offset_x* and *activity_offset_y* for `autoexplore`.
- The patrol indicator is added for units with repeating orders, or the “connect” indicator for units with this kind of orders, or the “G” sprite for units on `goto` (with *activity_offset_x* and *activity_offset_y*).
- Units awaiting a decision by the players are marked with a question mark sprite. *activity_offset_x* and *activity_offset_y*.
- The battlegroup sprite is added.
- Low fuel.
- Tired.
- The “plus” symbol is added for stacked units and loaded transports.
- Veteranancy.
- *HP*.

Unless mentioned otherwise, the sprites are drawn like “full sprites”.

Unit Sprites

Units sprites can be either unoriented or oriented, in which case the sprite that is displayed depends on the direction the unit is facing (it turns when it moves or fights).

Unoriented sprites are specified as `u.phalanx`. Oriented sprites have a direction suffix: `u.phalanx_s`, `u.phalanx_nw` and so on. For each unit type, either an unoriented sprite or a full set of the oriented sprites needed for the tileset topology must be provided (you can also provide both, see below).

The game sometimes needs to draw a sprite for a unit type that does not correspond to a specific unit, so is not facing a particular direction. There are several options for oriented tilesets:

- If the `unit_default_orientation` is specified for the tileset, the game will by default use that directional sprite. The direction does not have to be a valid one for the tileset.
- Specific unit types may override this by providing an unoriented sprite as well as the oriented ones; this does not have to be distinct, so it can point to one of the oriented sprites, allowing choice of the best orientation for each individual unit type. If `unit_default_orientation` is not specified, an unoriented sprite must be specified for *every* unit.

4.3 Soundsets

Soundsets are a collection of `spec` files that allow a Modder to add sound files to varying events that happen inside the game. Events such as founding a city, or attacking a unit can have a sound associated with them. There is a huge number of events in Freeciv21 that a Modder can attach a sound file to. Soundsets will start with a top-level `.soundset` file and with a directory of the same name will have `.ogg` sound files to play in the client.

Have a look at `stdsounds.soundspec` and associated files in `/stdsounds` for an example.

4.3.1 Sound Support

The server sends the game a list of primary and secondary sound tags for certain events. The **primary** tags are those preferred by the current modpack. The game does not need to have these sounds. The **secondary** tags should refer to standard sounds that all installations of Freeciv21 should have.

Tags are used to give an easy way to change sounds. A specfile is used to indicate which tags refer to which sound files. A change of spec file, given as an option at startup, will change sounds. For example:

```
$ freeciv21-client --Sound mysounds.soundspec
```

will read sound files from `mysounds.soundspec`. You will need to download or copy or link those sounds into whichever directory is mentioned in this file first, or edit it to refer to the right files. All references are by default relative to the `data/` directory. Soundpacks can be downloaded from the Freeciv21 modpack installer.

Tags

There are two kinds of sound tags:

- defined in the rulesets
- defined in the program code

While the former can be chosen freely the latter cannot be changed.

The sound tags associated with improvements (wonders and normal buildings), unit movements, and unit combat have to be set in the rulesets. Freeciv21 just hands these sound tags over to the game where they are translated into the

filenames of the sound files via the soundspec file. Every soundspec should have generic sound tags for wonders (`w_generic`), normal buildings (`b_generic`), unit movements (`m_generic`), and unit fights (`f_generic`).

Sound tags associated with certain events are generated in the Freeciv21 code and cannot be configured from outside. The soundspec file also has to have mapping for these tags. The complete list of such tags can be found in `data/stdsounds.soundspec`. The name of the tag is C++ enum name (see `common/events.h`) in lowercase. So `E_POLLUTION` becomes the tag `e_pollution`. There is no generic event tag and no alternate tags are used.

4.4 Musicsets

Musicsets are a collection of `spec` files that allow a Modder to add Music files to play as a soundtrack inside the game. Game music follows the game based on the nation selected and the mood. The mood is essentially binary: peace or war. Musicsets will start with a top-level `.musicspec` file and with a directory of the same name will have `.ogg` sound files to play in the client.

Have a look at `stdmusic.musicspec` and associated files in `/stdmusic` for an example.

4.4.1 Musicsets Overview

A Musicset is a soundtrack for Freeciv21 and is often documented in a single `.musicspec` file with references to the media files in a separate sub-directory.

The soundtrack is broken up into time periods following the technology tree and is split by two moods: peace and combat. Peace is generally defined when you are “farming” and growing your civilization with no fighting going on at all. Any kind of fighting, including defense of your cities will trigger combat and the music mood will change. After 16 turns of no combat the music will return to peace.

At the beginning of the game, you select the style of music you want to hear by picking the city style of your civilization. The current choices are:

- European
- Classical
- Tropical
- Asian
- Babylonian
- Celtic

The city style is automatically selected for you when you pick your nation, but you can change it if you like.

The discovery of University changes the music for all city styles for the Renaissance Age. All further discoveries noted below will change the music for all city styles/nations.

The discovery of Railroad brings in the Industrial Age.

The discovery of Automobile brings in the Electric Age.

The discovery of Rocketry brings in the Modern Age.

The discovery of Superconductors brings in the Post-Modern Age soundtrack. This is the final soundtrack of the game.

This is all documented in the `styles.ruleset` file in any ruleset. The ruleset file also establishes the look and feel of the cities as they grow and technologies are learned.

Spec File Layout

The .musicspec file is quite simple. Here is a sample:

```
[musicspec]
; Format and options of this spec file:
options = "+Freeciv-2.6-musicset"

[info]
; Add information related to the musicset here, such as a list of artists

[files]
music_menu = "MusicDirectory/MenuMusic.ogg"
music_victory = "MusicDirectory/VictoryMusic.ogg"
music_defeat = "MusicDirectory/DefeatMusic.ogg"

;Ancient Times - Bronze Age / Selection of European City Style
music_european_peace_X = "MusicDirectory/EuropeanPeaceSong0.ogg"
music_european_combat_X = "MusicDirectory/EuropeanCombatSong0.ogg"
;
;Ancient Times - Bronze Age / Selection of Classical City Stlye
music_classical_peace_X = "MusicDirectory/ClassicalPeaceSong0.ogg"
music_classical_combat_X = "MusicDirectory/ClassicalCombatSong0.ogg"
;
;Ancient Times - Bronze Age / Selection of Tropical City Stlye
music_tropical_peace_X = "MusicDirectory/TropicalPeaceSong0.ogg"
music_tropical_combat_X = "MusicDirectory/TropicalCombatSong0.ogg"
;
;Ancient Times - Bronze Age / Selection of Asian City Style
music_asian_peace_X = "MusicDirectory/AsianPeaceSong0.ogg"
music_asian_combat_X = "MusicDirectory/AsianCombatSong0.ogg"
;
;Ancient Times - Bronze Age / Selection of Babylonian City Stlye
music_babylonian_peace_X = "MusicDirectory/BabylonianPeaceSong0.ogg"
music_babylonian_combat_X = "MusicDirectory/BabylonianCombatSong0.ogg"
;
;Ancient Times - Bronze Age / Selection of Celtic City Stlye
music_celtic_peace_X = "MusicDirectory/CelticPeaceSong0.ogg"
music_celtic_combat_X = "MusicDirectory/CelticPeaceSong0.ogg"
;
;Discovery of University brings in Renaissance Age
music_renaissance_peace_X = "MusicDirectory/RenaissancePeaceSong0.ogg"
music_renaissance_combat_X = "MusicDirectory/RenaissanceCombatSong0.ogg"
;
;Discovery of Railroad brings in the Industrial Age
; Much more combat here
music_industrial_peace_X = "MusicDirectory/IndustrialPeaceSong0.ogg"
music_industrial_combat_X = "MusicDirectory/IndustrialCombatSong0.ogg"
;
;Discovery of Automobile brings in the Electric Age
; At ths time, pretty much all combat
music_electricage_peace_X = "MusicDirectory/ElectricPeaceSong0.ogg"
music_electricage_combat_X = "MusicDirectory/ElectricCombatSong0.ogg"
```

(continues on next page)

(continued from previous page)

```
;
;Discovery of Rocketry brings in the Modern Age
music_modern_peace_X = "MusicDirectory/ModernPeaceSong0.ogg"
music_modern_combat_X = "MusicDirectory/ModernCombatSong0.ogg"
;
;Discovery of Superconductors brings in the Post-Modern Age
music_postmodern_peace_X = "MusicDirectory/PostModernPeaceSong0.ogg"
music_postmodern_combat_X = "MusicDirectory/PostModernCombatSong0.ogg"
```

The value of X is a number from zero (0) up. For more than one song of each type, add a row and give the option another number +1 from the last. You can reuse song files in different areas.

4.5 Scenarios

Scenarios are custom saved games that a player can load and play. A Modder will use the map editor to create a map of the scenario and enable/change varying aspects of the game to set up the game scenario.

Have a look at the scenarios shipped with Freeciv21 in /scenarios for some examples.

4.5.1 Scenarios Overview

Overview

Freeciv21 scenario-files are essentially savegames with a specific game situation. The savegame format allows some things that are useful only when the savegame in question is a scenario. There is a [scenario] section with some parameters applicable to scenario saves only, and possibility to include some lua script for scripting the scenario. The deciding difference between regular savegames and scenario savegames is that latter have `is_scenario` set to TRUE in [scenario] section. Freeciv21 saves current game situation as a scenario file, with the full [scenario] section, when one saves by the /scensave server command.

Compatibility

Freeciv21 can load savegames saved by older versions. How far back this is supported depends on version. Scenario files are often not updated to the latest savegame format version, but can be of such older version. When editing such older format savegames, one needs to be consistent with the format version. One cannot use features of the newer format in older format savegame even though the Freeciv21 server can load both older format and newer format. This document describes the current format of the Freeciv21 version. Some things described here should not be used when editing savegames with older format.

The [scenario] section

Savegame [scenario] section can have following fields:

- `is_scenario` : Always TRUE in a scenario savegame.
- `name` : Name of the scenario.
- `authors` : Text listing authors of the scenario.
- `description` : Description of the scenario.
- `save_random` : Whether random number state is saved to the scenario.

- `players` : Whether player information is saved to the scenario, or is it map-only scenario.
- `startpos_nations` : Whether nations should use start positions defined in the scenario map.
- `prevent_new_cities` : Is founding new cities prevented in the scenario, locking it to cities present on the map.
- `lake_flooding` : Should ocean tiles connecting to a lake tile flood the lake tile.
- `handmade` : Set to TRUE when savegame file is manually edited. This has no gameplay effect, but it indicates that scenario might have some properties that are not yet supported in the editor. When editing such a scenario in the editor, warning is shown that some properties might get lost.
- `allow_ai_type_fallback` : Is it ok to fallback to another player *AI* type if one defined in the scenario file is not available when scenario is loaded.
- `ruleset_locked` : Is the scenario locked to one specific ruleset. Default is TRUE.
- `ruleset_caps` : When scenario is not `ruleset_locked`, this capability string defines required capabilities of compatible rulesets.
- `datafile` : Luadata to load. See section *Luadata* for details.

Luadata

Some rulesets have a lua script that can parse special luadata file to adjust their behavior. Such luadata file can be provided by the ruleset itself, or by the scenario using the ruleset. This is controlled by `datafile` field in the scenario savegames `[scenario]` section.

- If the field is omitted, ruleset's own luadata is used, if present.
- If field has value 'None' (case insensitive), luadata is not used at all, as if it was not present even in the ruleset.
- Any other value of `datafile` is considered prefix part of a filename of form `<prefix>.luadata` to use as the luadata file. The file must be found from the Freeciv21's savegame path, usually from the same directory where the scenario savegame itself is.

4.6 Installer

Refer to a document on how to serve your own modpack set. If you are interested in how to use the modpack installer, refer to *Using the Modpack Installer Utility*

4.6.1 Serving Modpacks for the Modpack Installer

This document discusses how to set up a web server so that users can download modpacks you publish.

To host modpacks, you need a web server that can host plain static files. You do not need to run any custom code or frameworks on that web server, just to publish files with a specific layout, detailed below.

On the modpack server, there are up to three layers of files required:

1. A list of available modpacks (optional, advanced).
2. One control file per modpack.
3. The individual files comprising the modpacks.

Each of these layers is described in detail below.

Each layer refers to files in the next layer down. References can be with relative URLs (so that a modpack or set of modpacks can be moved without changing any file contents), or with absolute URLs (so that the different layers can be hosted on different web servers).

Almost all of these file formats are specific to one major version of Freeciv21. This document only describes the formats for the major version of Freeciv21 it is shipped with.

1. List of Modpacks

This is only needed if you want to let users browse a list of available modpacks before choosing one to install. To look at your modpack list instead of the standard one, users will usually have to start the modpack installer with non-standard arguments (see above).

The modpack list is a standard JSON file with a specific structure.

Here's an example:

```
{
  "info": {
    "options": "+modpack-index-1.0",
    "message": "Example modpack list loaded successfully"
  },
  "modpacks": [
    {
      "name": "Example",
      "version": "0.0",
      "license": "WTFPL",
      "type": "Modpack",
      "url": "https://example.com/example.json",
      "notes": "This is an example"
    }
  ]
}
```

This file uses the modpack index format version 1.0, as is indicated in the `info.options` field. The optional `info.message` is displayed on the status line when the modpack installer starts up. It should be kept to one line.

The modpacks list contains a list of modpacks. This example contains just one modpack. Each modpack may contain the following fields:

“name”, “version”, “type”

These three fields should match those in the `.json` file which URL links to.

“subtype”

Optional free text. For tilesets or scenarios, conventionally indicates the map topology with one of `overhead`, `iso`, `hex`, or `hex & iso` (and these will be localised). Otherwise use `-`.

“license”

Free text summarising the distribution terms for the modpack content, by naming a well-known license, not quoting the full license text! Consider using SPDX identifiers (<https://spdx.org/licenses/>).

“URL”

The URL to a `.json` file for the individual modpack. The URL can be either relative in which case it's relative to the URL of `modpack.list`, or absolute - which can be on some other web server.

“notes”

Optional free text; usually shown as a tooltip.

2. Control File: Defining an Individual Modpack

This is the core control file for a modpack, specifying what files it contains, where to download them from, and where they are installed.

Some modpack authors will just publish the URL of an `.json` file directly, for users to give to the modpack installer tool. There doesn't have to be a `modpack.list` file anywhere that refers to the `.json` file.

Again, this is a file in standard JSON format. Its filename must end in `.json`.

Here is an example of a modpack control file:

```
{
  "info": {
    "options": "+modpack-1.0",
    "base_url": ".",
    "name": "Some ruleset",
    "type": "Modpack",
    "version": "0.0"
  },
  "files": [
    "some_ruleset.serv",
    "some_ruleset.tilespec",
    "some_ruleset/nation/german.ruleset",
    "some_ruleset/nation/indian.ruleset"
  ]
}
```

The `info` section has overall control information:

“options”

Defines the version of the file format. Should be exactly as shown in the example.

“name”

A short name for the modpack. This is used for version and dependency tracking, so should not contain minor version information, and should not change once a modpack has been released for a given major version of Freeciv21. Case-insensitive.

“version”

Textual version information. If another modpack uses this one as a dependency, this string is subject to version number comparison (using the rules of Freeciv21's `cvercmp` library, which should give sensible results for most version numbering schemes).

“type”

This must be one of the following:

- **Ruleset:** `foo.serv`, `foo/*.ruleset`, `foo/*.lua`, etc.
- **Tileset:** `foo.tilespec`, `foo/*.png`, etc.
- **Soundset:** `foo.soundspec`, `foo/*.ogg`, etc.
- **Musicset:** `foo.musicspec`, `foo/*.ogg`, etc.
- **Scenario:** `foo.sav`; installed to a version-independent location.
- **Modpack:** Conventionally used for modpacks that contain more than one of the above kinds of material

- **Group:** Contains no files but only depends on other modpacks. At the moment, only `Scenario` causes special behavior.

“base_url”

URL to prepend to the `src` filenames in the `files` list. May be relative to the `.json` file – starting with `./` – or absolute in which case the files can be on some web server different to where the `.json` file lives.

The **files** list defines the individual files comprising your modpack. It must list every file individually; any files in the same directory on the webserver that are not listed will not be downloaded. Entries can be strings as shown above, in which case the same file name is used for downloading relative to `info.base_url` and installing relative to the data directory. If the installed name is different from the name on the server, the following syntax can be used instead:

```
{  
  "url": "some-remote-file",  
  "dest": "where-to-install-it"  
}
```

The URL can be either relative (to `info.base_url`) or absolute. The two syntaxes can be mixed in the same modpack.

Note: Forward slash `/` (and not backslash `\`) should be used to separate directories.

Some advice on the structure of files in modpacks:

- You should generally install files in a directory named after the modpack, with a few exceptions (`.serv`, `.tilespec`, `.soundspec`, and `.musicspec` files must be installed to the top level, and should reference files in your subdirectory). Individual files and directories install names should usually not embed version numbers, dates, etc., so that when a new version of modpack X is installed, it cleanly overwrites the old version, rather than leaving both cluttering up the user’s installation.
- The modpack installer does not stop different modpacks overwriting each other’s files, so published modpacks should be disciplined about namespace usage. If you’ve derived from someone else’s modpack, you should probably give your derivative new filenames, so that both can be installed simultaneously.
- There is no *white-out* facility to delete files from a user’s installation – if a newer version of a modpack has fewer files than an old one, the old file will persist in some users’ installations, so your modpacks should be designed to be tolerant of that.
- At the moment, there is no restriction on what kind of files a given *type* can install, but modpacks should stick to installing the advertised kinds of content. It’s **OK** to install extra files such as documentation in any case (`LICENSE/COPYING`, `README.txt`, etc.).
- If your modpack contains a ruleset, you should usually install a `.serv` file at the top level (which can be a one-line file consisting of `rulesetdir <name>`, as this is needed for the server to enumerate the available rulesets).

In some cases, a modpack may depend on other modpacks, for instance if it reuses some of their files. This can be handled by declaring a dependency with respect to the other modpack. Dependencies are listed in the optional `dependencies` list of the JSON file. Each entry in that list must contain the following object:

```
{  
  "modpack": "...",  
  "url": "...",  
  "type": "...",  
  "version": "..."  
}
```

The keys are explained below:

- **modpack:** What the dependency modpack calls itself when installed (that is, name from its `.json` file).

- **url**: URL to download modpack if needed. Can be relative or absolute.
- **type**: Must match `type` from dependency's `.json` file.
- **version**: Minimum version of dependency (as declared in its `.json` file). Subject to version number comparison algorithm.

If the modpack installer thinks the required version, or a newer version, of the dependency is already installed, it will do nothing, otherwise it will download the dependency modpack, and any of its own dependencies, recursively.

3. Individual Modpack Files

These are the files comprising the modpack (`*.ruleset`, `*.png`, etc.), that will be copied verbatim to the user's Freeciv21 profile directory and read by the Freeciv21 client and server. The modpack installer does not modify the files in any way.

The files must be hosted individually on the web server; the modpack installer tool cannot unpack any archives such as `.zip` files. Individual scenarios can be compressed (e.g. `.sav.gz`, as the Freeciv21 engine can uncompress these files).

Because the `*.json` file can change the file paths / names on download, the layout on the modpack server doesn't have to correspond with the installed layout. An individual file can be shared between multiple modpacks, if you want.

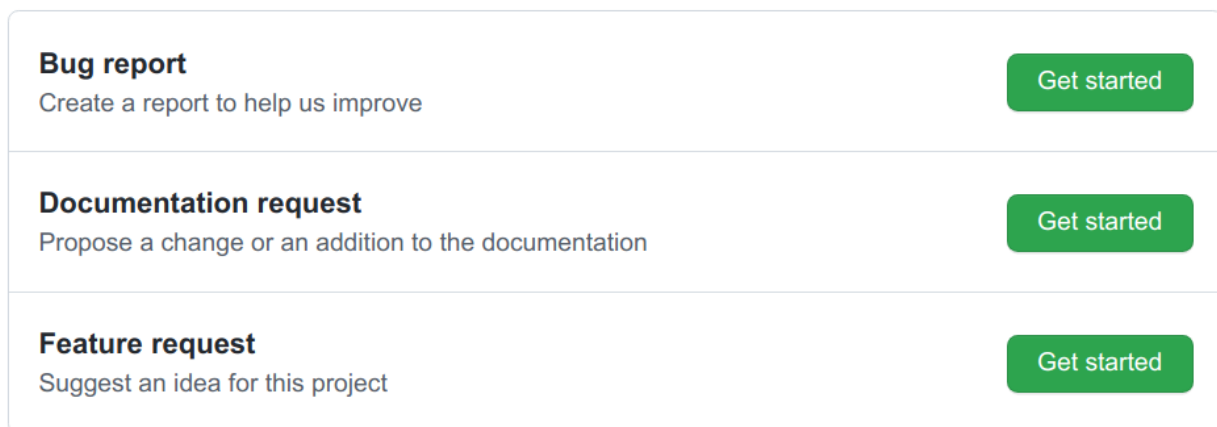
CONTRIBUTING

The Contributing category is where documentation editors provide instructions on how to contribute to the Freeciv21 project.

5.1 Filing Bugs and Enhancement Requests

Freeciv21 certainly contains some bugs and definitely needs improvements. Please report all you find or want to report to <https://github.com/longturn/freeciv21/issues/new/choose>.

The project team has implemented GitHub Issue Templates to make the process as easy as possible. We currently have 3 different ones: Bug Report, Documentation Request, and Feature Request. Each template has prompts to help you fill in the details. See an example screenshot below.



Bug report Create a report to help us improve	Get started
Documentation request Propose a change or an addition to the documentation	Get started
Feature request Suggest an idea for this project	Get started

Fig. 5.1: GitHub Issue Templates

5.1.1 Reporting an Issue

The Freeciv21 project uses GitHub issues to track bugs, enhancement requests and documentation requests. Take note of the following when thinking of submitting an issue.

- Check that it is not listed as a known issue. For a continuously updated list, see: <https://github.com/longturn/freeciv21/issues>
- Check the Freeciv21 Releases page at <https://github.com/longturn/freeciv21/releases>, to ensure you are playing the latest version. We may have already fixed the problem.
- If the first two items do not work, then feel free to submit a new issue. We have implemented GitHub issue templates which will walk you through the major aspects of a new issue. It is important to note that the more details you can provide, the better. If a screenshot is available, then by all means add it to the issue.
 - If you are an advanced hacker, you can also provide a core dump or stack trace to better help the developers of Freeciv21 understand the nature of the issue.
 - If there is an issue in a translation, please report it. We are still getting organized with *internationalization* (i18n) support for the client and server. The more we know the better.

5.2 Set up a Development Environment

Contributing code to the Freeciv21 project or contributing to any of the games that the Longturn community manages requires a bit of setup. This document should get you up and running with the basics. It should be noted, that you typically only have to go through this process once unless you setup a new workstation.

5.2.1 Workstation

Freeciv21 can be developed on Linux, Windows, and macOS. Any current version of these OS's is acceptable. For Windows, you will need to setup either the MSYS2 environment or Visual Studio to do development. Refer to *Setting up MSYS2 for Windows* or *Visual Studio* for more information. It should be generally understood that **Linux** is the preferred development platform. For a complete set of installation steps, you can refer to *Compiling Freeciv21*.

Technically all you need is a text editor of some kind to edit the files, but most people prefer to use an integrated development environment (IDE).

All platforms can use *KDevelop*. However, there are some caveats:

- On Windows, due to the nature of the integration with MSYS2, native compilation and debugging is not supported. This is resolved with *Visual Studio* as the IDE.
- On macOS, KDevelop is still considered experimental. Many Mac users use *XCode*.

For the best results, especially if you are editing game code and not just Longturn game rulesets or documentation, you will want **Linux** to be your workstation OS. Many of the current developers use a Debian variant such as Ubuntu. Instructions for getting all of the tools needed for Debian Linux can be found in *Compiling Freeciv21*. Do not follow the steps to clone the repository (e.g. the `git clone` command), that will happen in a bit.

5.2.2 GitHub

The Longturn Community uses the online source code control and revision system known as [GitHub](#). To contribute, you will need an account on this platform. There is no cost.

With an account, you can go to the [Longturn](#) community repository page and **fork** a repository (such as the Freeciv21 repository) to your personal GitHub account. Go to the main page of the repository you want to fork and you will find a **fork** button in the upper-right corner.

In order to get code pushed to the forked repository to your local workstation, you need to setup an SSH key pair to share with GitHub. Follow these [instructions](#).

With SSH set up, now it is time to clone the forked repository from your personal GitHub account to a local copy on your workstation.

First make a working directory to place the files in:

```
$ mkdir -p $HOME/GitHub
$ cd $HOME/GitHub
```

You can get the appropriate path by going to your forked copy in a browser, click the code button and then select the SSH option as shown in this sample screenshot:

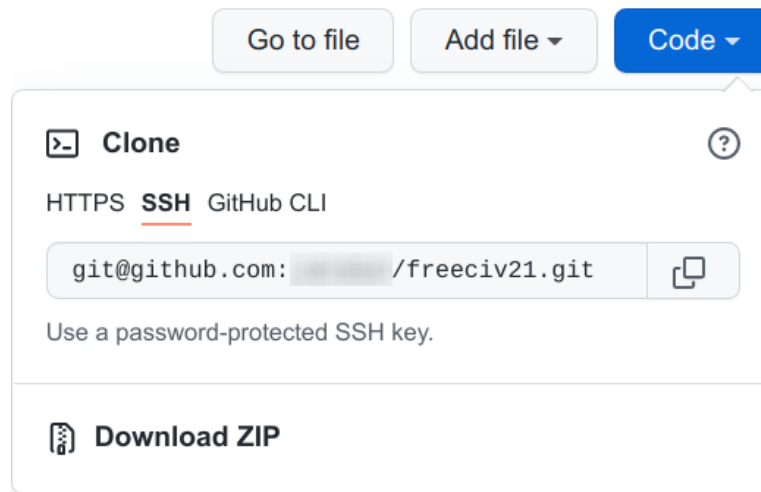


Fig. 5.2: GitHub Clone SSH

Once you have the proper path, here is the command to clone the repository:

```
~/GitHub$ git clone git@github.com:[username]/freeciv21.git
```

This will clone the forked repository to the ~/GitHub/freeciv21 directory on your computer.

The final repository setup item is to link the original Longturn project repository to your local area on your computer:

```
~/GitHub/freeciv21$ git remote add upstream https://github.com/longturn/freeciv21.git
```

You will also need to set a couple global configuration settings so git knows a bit more about you.

```
~/GitHub/freeciv21$ git config --global user.email [email address associated with GitHub]
~/GitHub/freeciv21$ git config --global user.name [your first and last name]
```

5.2.3 KDevelop Project Setup

On Linux, the easiest way to install KDevelop is with a command such as this:

```
$ sudo apt install kdevelop clang-format-11
```

Once installed, you can then import the Freeciv21 project into it. Follow these steps:

1. Open Kdevelop
2. *Project* → *Open/Import Project*
3. Find `freeciv21/CMakeLists.txt`
4. *Session* → *Rename Current Session* to Freeciv21
5. *Project* → *Open Configuration* → *Language Support*. Click on the *Language Support* tab and ensure that the C++ Profile is *c++17*, the C Profile is *c99*, the OpenCL C Profile is *CL1.1*, the CUDA C Profile is *c++11*, and finally the Compiler for Path is *GCC*.
6. Allow kdevelop to parse all of the code. This can take a while. Eventually you will see a full tree of the code in the Projects tab on the left.

Here are some useful help/documentation links:

- <https://userbase.kde.org/KDevelop5/Manual>
- <https://docs.kde.org/trunk5/en/kdevelop/kdevelop/kdevelop.pdf>

Now you are ready to edit some code! When ready, follow the steps to submit a pull request here: [How to Submit a Pull Request](#).

5.2.4 Qt Creator

Freeciv21 uses the Qt framework for many things, especially the game client. There are many `.ui` files in the `client` directory. If you want to edit those in a user interface (versus editing the raw XML), you will want to install Qt Creator. You do so with a command such as this:

```
$ sudo apt install qtcreator
```

Once installed you will get access to a program called **Qt Designer**. This tool is a graphical interface you can use to edit the `.ui` files. You also gain access to Qt documentation inside of KDevelop as well. You can read the Qt Designer Manual for more help here: <https://doc.qt.io/qt-5/qtdesigner-manual.html>

5.3 Setting up MSYS2 for Windows

This document is about building and compiling Freeciv21 with MSYS2 on Windows. MSYS2 is available from <https://www.msys2.org>. The Longturn Community builds the Windows Installer packages using MSYS2. If you are looking for a first class development environment on Windows, have a look at [Visual Studio for Windows](#).

5.3.1 Setup

This chapter is about creating a new MSYS2 build environment.

1. Download MSYS2 from their homepage: <https://www.msys2.org/#installation>
2. Run the downloaded file to install MSYS2 on you build system. This author recommends that you install the files in a directory off of the root of your C:\ Drive. For example: C:\Tools
3. Launch `msys2_shell` from the installation directory. Issue the following command to update the base set of packages.

```
> pacman -Syuu
```

4. The `msys2_shell` will close when this is complete. Run it again and install following packages with `pacman -Su <package name>`. The packages needed for building Freeciv21 are numerous. These packages are needed even if you do not plan to make the installer, update documentation, but only build Freeciv21 for local use.
5. Install these arch independent packages that are needed for building Freeciv21. With these packages it is possible to build a Freeciv21 source tree that was created via release tarball or zip file.

- `pkgconf`
- `git`
- `automake`
- `libtool`
- `autoconf`
- `patch`
- `ed`
- `python3`
- `python-pip`
- `make`
- `clang`

6. Install these packages for building common parts.
 - `mingw-w64-x86_64-gcc` (will give you `bzip2`, `readline` and `python3`)
 - `mingw-w64-x86_64-gdb`
 - `mingw-w64-x86_64-curl`
 - `mingw-w64-x86_64-lua`
 - `mingw-w64-x86_64-tlua`
 - `mingw-w64-x86_64-SDL2_mixer`
 - `mingw-w64-x86_64-cmake`
 - `mingw-w64-x86_64-ninja`
 - `mingw-w64-x86_64-libunwind`
 - `mingw-w64-x86_64-make`
 - `mingw-w64-x86_64-python-pip`

7. Packages for building the client and/or Ruledit.

- mingw-w64-x86_64-qt5
 - mingw-w64-x86_64-qt5-svg
 - mingw-w64-x86_64-karchive-qt5
8. Package needed for building Windows installer package.
- mingw-w64-x86_64-nsis
9. Packages needed for documentation development.
- mingw-w64-x86_64-python-sphinx
10. Add some environment variables to the `.bash_profile` file. The code sample assumes `x86_64`.

```
export PATH=/mingw64/bin:${PATH}
export MSYSTEM=MINGW64
export PKG_CONFIG_PATH=${PKG_CONFIG_PATH}:/mingw64/lib/pkgconfig:/mingw64/share/pkgconfig
```

5.3.2 Premade Environment

Visit <https://github.com/jwrober/freeciv-msys2> to get a set of scripts and instructions to quickly build an environment. The scripts create an `x86_64` development environment.

5.3.3 Build

Now that you have the environment setup. You can follow the steps in *Compiling Freeciv21*. MSYS2 in a Linux emulation environment on Windows, so the commands for `git`, `cmake`, etc. work the same.

5.4 Visual Studio for Windows

Freeciv21 can be compiled using Microsoft Visual Studio® and `clang-cl`. This page will help you get version 2022 up and running.

Warning: Visual Studio and the corresponding dependencies require a great deal of HDD space on your computer. Be sure to have at least 80GB of available space before starting this process.

5.4.1 Base Installation

Start by downloading the Community Edition installer from <https://visualstudio.microsoft.com/vs/community>.

Double-click the `VisualStudioSetup.exe` file in your Downloads folder and follow the prompts to get the setup process started. At some point you will be prompted to select the type of workload you want to install. Select *Python Development* and *Desktop development with C++*. On the right panel, under *Installation Details*, uncheck *vcpkg package manager* and check *Windows 10 SDK 10.0.20348.0*.

Note: We remove the `vcpkg` package manager inside of Visual Studio in order to install one that is better managed and easily updated later down this page.

Next click on the *Individual Components* tab and select the following options: *Git for Windows*, *C++ Clang Compiler for Windows (17.0.3)*, and *MSBuild support for LLVM (clang-cl) toolset*.

When ready click *Install*. Depending on your Internet connection speed, this may take a while. There is a *Start after installation* check box that you can uncheck as we do not need to open Visual Studio post installation.

When the installation process is finished you may close the installer.

5.4.2 Install Python

FreeCiv21 development needs access to a Python Interpreter. We also need Python for the `git clang-format` command used as part of adding a [pull request](#) to our GitHub repository.

Right-click the *Start Menu* and either open an Admin level PowerShell or Terminal depending on the version of Windows you have installed. From now on we will refer to this as the **Terminal**.

From the prompt enter `python` and press enter. Windows should open the Windows App Store to Python 3.x (as of this writing its 3.12). Click *Get* to install it. Once the installation is finished, you can close the Windows App Store and return to the Terminal. Enter `python` a second time and you should now get a python interpreter like below. You can type `quit()` to exit the interpreter.

```
PS C:\> python
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb 6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)]
> on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

5.4.3 Set Up VCPKG

`vcpkg` is a library package manager developed by Microsoft. It makes downloading and installing third-party libraries much easier. FreeCiv21 uses `vcpkg` in this way via the manifest file `vcpkg.json`. Documentation is available here: <https://github.com/microsoft/vcpkg>

First, create a directory on your computer to install `vcpkg` into. For example: `C:\Tools`. The directory can be anywhere, however the author prefers to **not** install things in the root of the `C:\` drive.

In the Terminal window change to the directory.

```
PS C:\Windows\System32> cd c:\tools
PS C:\Tools>
```

Follow these steps to install `vcpkg`. The `setx` command should be altered to be the correct path that you installed `vcpkg` into. The forward slashes are correct.

```
Tools> git clone https://github.com/microsoft/vcpkg
Tools> .\vcpkg\bootstrap-vcpkg.bat
Tools> setx VCPKG_ROOT C:/Tools/vcpkg /M
Tools> exit
```

Warning: The `vcpkg` website/readme will ask for you to run a `vcpkg integrate install` command to fully integrate all the packages installed into Visual Studio. **Do Not** run this command as it actually breaks Visual Studio's ability to find and use the `clang-cl` compiler, which we need.

5.4.4 GitHub

The Longturn.net Community uses the online source code control and revision system known as [GitHub](#). To contribute, you will need an account on this platform. There is no cost.

With an account, you can go to the [Longturn](#) community repository page and **fork** a repository (such as the Freeciv21 repository) to your personal GitHub account. Go to the main page of the repository you want to fork and you will find a **fork** button in the upper-right corner.

In order to get code pushed to the forked repository to your local workstation, you need to setup an SSH key pair to share with GitHub. Follow these [instructions](#).

With SSH set up, now it is time to clone the forked repository from your personal GitHub account to a local copy on your workstation. You can get the appropriate path by going to your forked copy in a browser, click the code button and then select the SSH option as shown in this sample screenshot:

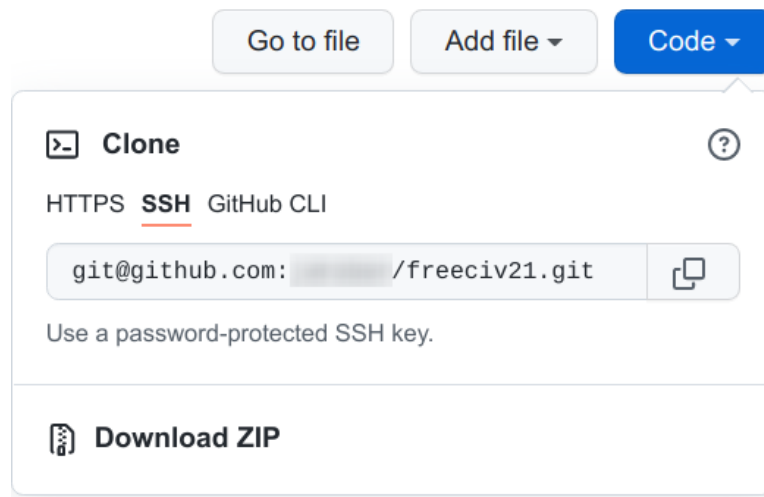


Fig. 5.3: GitHub Clone SSH

Open up Visual Studio. A menu will appear. Go to the very bottom right and click the link to **continue without code**. Select *View* → *Terminal* to open a terminal in the IDE if one is not already shown at the bottom. Once you have the proper GitHub path, here is the command to clone the repository:

```
PS C:\Users\[user]\Source\Repos> git clone git@github.com:[username]/freeciv21.git
```

This will clone the forked repository to the C:\Users\[User]\Source\Repos\freeciv21 directory on your computer.

The final repository setup item is to link the original Longturn project repository to your local area on your computer:

```
Repos> cd freeciv21
Repos\freeciv21> git remote add upstream https://github.com/longturn/freeciv21.git
Repos\freeciv21> git fetch upstream
Repos\freeciv21> git pull upstream master
```

You will also need to set a couple global configuration settings so git knows a bit more about you.

```
freeciv21> git config --global user.email "[email address associated with GitHub]"
freeciv21> git config --global user.name "[your first and last name]"
```

5.4.5 Build Visual Studio Solution

Now let us get Visual Studio set up. Select *Git* → *Local Repositories* → *Open Local Repository* and then select the `freeciv21` folder. Visual Studio will take a minute to parse the source tree.

Select *File* → *Open* → *Folder*. Select the `freeciv21` directory and then Visual Studio will parse the files in there. It can take a few minutes for this to complete. When complete, Visual Studio will open a **CMake Overview Pages** tab.

Microsoft provides documentation on CMake in Visual Studio – <https://docs.microsoft.com/en-us/cpp/build/cmake-projects-in-visual-studio?view=msvc-170>

5.4.6 Qt Visual Studio Tools

There is an available extension called **Qt Visual Studio Tools**. This extension allows you to use the Qt Designer and other Qt tools within the Visual Studio IDE. This is especially useful if you want to edit any of the client `.ui` files.

Start by [downloading](#) the installer.

1. Double-click the downloaded file and login with your Qt account. If you do not have a Qt account, there should be a link in the installer window to create one.
2. Click *Next*, agree to the GNU License and state that you are an individual, not a business.
3. Click *Next* and pick to either send or not send usage statistics to Qt.
4. Click *Next*. Note that installing to `C:\Qt` is fine.
5. Ensure *Custom Install* is selected and then click *Next*.
6. On the right, uncheck *latest supported release* and check *Archive*, click *Filter*.
7. Expand *Qt* and check the `5.15.2` option.
8. Click *Next* and agree to the Microsoft license.
9. Click *Next* and note that **Qt** for the Start Menu is fine.
10. Click *Next* and *Install* to begin the process.

Note: You can reduce the size of the Qt Tools install by expanding the `5.15.2` option and unchecking **WebAssembly**, **MSVC 2015 64-bit**, **MSVC 2019 32-bit**, **MinGW 8.1.0 32-bit**, **MinGW 8.1.0 64-bit**, **UWP***, and **Android**. Unless you intend to develop for those platforms, you do not need to download and install those components for Freeciv21.

To install the extension. In Visual Studio, click on *Extensions* → *Manage Extensions* → *Online*. Search for **Qt Visual Studio Tools** and install it.

To set the version of Qt in the extension's options. Click on *Extensions* → *Qt VS Tools* → *Qt Versions*. Add version `5.15` and in the path enter: `C:\Qt\5.15.2\msvc2019_64\bin`.

5.4.7 Final Steps and Notes

At this point Visual Studio is ready for you to configure, compile (build), and install Freeciv21.

To configure Freeciv21 in Visual Studio you first have to select either the `windows-release` or `windows-debug` configuration preset as defined in `CMakePresets.json`. On the *Standard* toolbar, which is enabled by default, you will click the *Target System* drop down box and pick the **Local Machine** option. Next you will click the *Configuration* drop down box and pick the **windows-release** or **windows-debug** option (debug is preferred for development).

With these options set you will lastly click *Project* → *Configure Cache*. When this process is complete you can then compile (build) by clicking *Build* → *Build All*. Visual Studio will compile all targets for Freeciv21 and place the output into the `build-vs` directory. If you want to install Freeciv21 to test any work you are doing, you can go to *Build* → *install Freeciv21*. When complete, you should find a fully functional install in the `build-vs/install` directory.

Note: The preferred *Configuration* is **debug-windows**, especially if you want to troubleshoot code with the built-in debugger and also if you plan to use the unit test feature CTest.

Note: The first time you run the Configure Cache command (from *Project* → *Configure Cache*) or ask Visual Studio to generate the C++ Intellisense data, Visual Studio will invoke the `vcpkg` installation process to download and compile all of the project dependencies listed in the manifest file: `vcpkg.json`. **This will take a very long time.** On a fast computer with a good Internet connection it will take at least 3 hours to complete. Everything will be downloaded and compiled into the `C:\Tools\vcpkg` directory, or wherever you configured `vcpkg` earlier. Binaries for the packages will be copied into the `./build-vs/` directory inside of the main Freeciv21 directory and reused for subsequent builds.

Attention: As documented in [Compiling Freeciv21](#), there is a `--target package` option available to build an installable package for Windows. This is only available to the `MSYS2` environment. This does not mean that you can not test an install using Visual Studio. After going to *Build* → *install Freeciv21* you can still manually start up the client or a server as needed to debug. To do this you will start up either the client, the server, or both and then in Visual Studio go to *Debug* → *Attach to Process*

Notes about Clang-Cl vs MSVC

Freeciv21 has two custom presets specifically for Windows and Visual Studio because Visual Studio defaults to the MSVC compiler instead of LLVM's Clang-Cl compiler that we installed earlier. The main reason why we do this is because the code requires proprietary extensions that MSVC does not implement. Instead the Longturn community supports the open source LLVM Clang-Cl compiler on Windows in Visual Studio as it supports the required proprietary extensions (note that GNU GCC is used in [MSYS2](#) on Windows). If you select any preset and allow Visual Studio to configure with the Ninja generator and MSVC compiler you will receive a great number of errors and the build will fail.

If you are interested in configuring and compiling from the command line instead of the GUI, you can use these commands:

```
cmake --preset windows-debug -S . -B build-vs
cmake --build build-vs --target install
```

The first command configures Visual Studio to compile a Debug version of the programs and places the install location to be a sub-directory of the `build-vs` directory for use during debugging and testing purposes. This is the same as selecting the `windows-debug` preset configuration. The second and third command then “builds” and “installs” the configured code solution. You will need to manually start the client and/or server to test.

5.5 How to Submit a Pull Request

A Pull Request (PR), also commonly known as a Merge Request, is a mechanism to “pull” code from one repository into another one and merge the change into the source repository.

Before we go much further, let us take a few minutes to describe what happened when you *Set up a Development Environment* and to define a few terms that will help with further instructions. Starting with a diagram of the setup of the three locations where code resides:

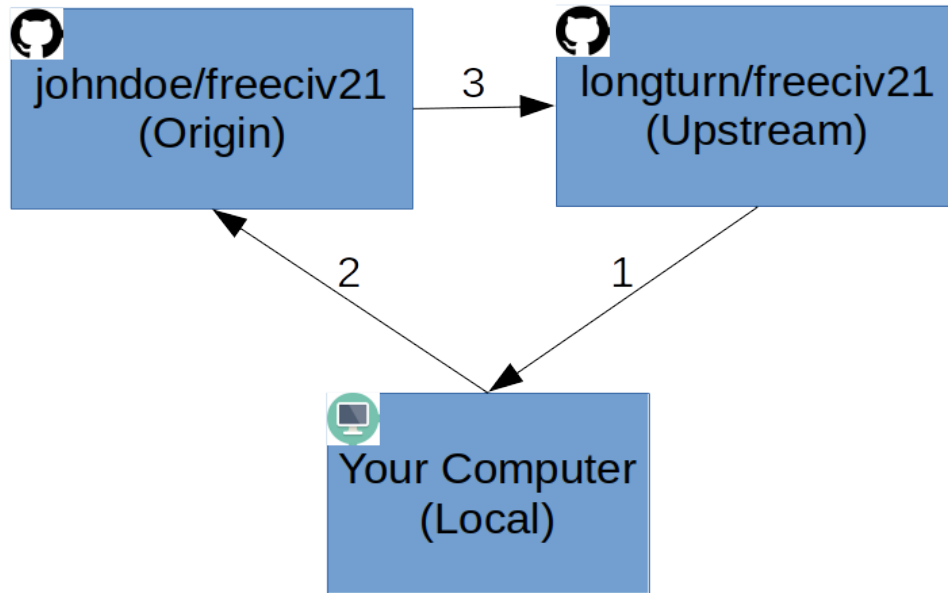


Fig. 5.4: Diagram of GitHub Repositories

As part of the *Compiling Freeciv21* process, one of the programs installed is `git`. This program is used to interact with the varying places that code can reside. The `git` program has a relatively standard way of referring to things that are in the diagram. If you have not worked with `git` much, it is highly recommended you read the first three chapters of the *Pro Git Book* before continuing.

- **Upstream:** This refers to the source (original) repository. For us this is the Longturn Freeciv21 or Games repositories.
- **Origin:** This refers to the fork you made in your own personal GitHub account.
- **Local:** This is not a `git` term per se, but is a well understood industry standard way of referring to the copy of code on a person’s computer. This is where the changes are made and eventually make it back up to origin and over to upstream via a Pull Request.

The arrows represent the path that updates (e.g. changes) of files occur.

1. File updates from *Upstream* are downloaded (pulled) to *Local*.
2. After changes are made, file updates from *Local* are uploaded (pushed) to *Origin*.
3. File updates from *Origin* are transferred to *Upstream* laterally via a Pull Request.

With this in mind, submitting a Pull Request generally entails these steps:

1. Update Local with the latest from Upstream.
2. Create a branch from Upstream.
3. Edit files as needed in Local.

4. Push the changes in a commit to Origin.
5. Requesting a Pull Request from Origin to Upstream.

5.5.1 1. Update Local with the Latest Updates from Upstream

Assuming you are in the appropriate Local directory, issue these commands to get the latest code from Upstream:

```
~/GitHub/freeciv21$ git checkout master
~/GitHub/freeciv21$ git pull upstream master --ff-only
```

The first command sets the working branch to `master` on Local. The second command downloads the changes from Upstream to Local.

Note: If you want `git pull` to always do a “fast-forward” (which is preferred), you can set a local repository configuration option with this command: `git config --local pull.ff only`. You only need to do this once. After the config is set, you can drop the `--ff-only` option from the last command above.

5.5.2 2. Create a Branch from Upstream

Now that things are all up to date with the latest code, let’s create a branch to do your work in.

```
~/GitHub/freeciv21$ git checkout -b [some_feature] upstream/master
```

The tag `[some_feature]` should be replaced by something that matches what you are planning to work on. For example, it could be as easy as `issue_123`, meaning that this branch is changes to resolve GitHub Issue 123 in the Freeciv21 Issues list.

Attention: It is recommended to never make changes in the `master` branch. Best practice is to make changes in a different branch. This makes it much easier to evaluate changes as part of the Pull Request evaluation process and keeps things much cleaner from a repository management perspective.

5.5.3 3. Edit Files as Needed in Local

This step is as simple as it sounds. Edit the files in the Local area as needed to complete the work you want to work on. Depending on what you are doing, it is likely that you will want to compile the code to test your work. Here are some quick steps that you can use to run through quick compiles of code and/or documentation updates.

```
~/GitHub/freeciv21$ rm -rf build
~/GitHub/freeciv21$ cmake . -B build -G Ninja -DCMAKE_INSTALL_PREFIX=$PWD/build/install -
  -DCMAKE_BUILD_TYPE=Debug
~/GitHub/freeciv21$ cmake --build build
~/GitHub/freeciv21$ cmake --build build --target install
~/GitHub/freeciv21$ cmake --build build --target docs
```

The first command cleans out the `build` directory to start with a fresh configuration. The second command runs a configure process and then sets things up to do the install portion inside the same `build` directory, keeping work in progress all in one place.

The third and fourth commands compile and install the code. The last command is used to generate the documentation that you are reading right now. You can open `./build/docs/index.html` to see a locally generated copy of the documentation prior to pushing a change up.

Note: You do not have to start from the top of the list of commands every time you want to check on things. If you have already done a full pass to install and then make some more changes, you can go straight to the build and install commands. The `cmake` program is smart enough to determine what changed and only compile those files again. It's much faster to do things this way!

One more thing to note here, this section is only talking about making changes to the Freeciv21 repository and has not really talked about the Games repository, nor talked about Rulesets or Tilesets. In this scenario, there is not any compiling going on. The Ruleset or Tileset editor is editing files and testing locally. The Longturn Games repository is effectively a repository of Rulesets.

5.5.4 4. Push the Changes in a Commit to Origin

Now the moment arrives, you have a fully completed set of changes you want to share! The next step is to evaluate what all has changed and build a commit. A commit is an organized set of changes all bundled up together. There are lots of ways to organize commits, for this set of instructions we are going to assume that all of your changes are going to be bundled into one commit. If you want to learn how to build more than one commit into a Pull Request then you can read about [git add](#) and [git commit](#). Most of the work is handled with `git add`.

```
~/GitHub/freeciv21$ git status
~/GitHub/freeciv21$ git add --all
~/GitHub/freeciv21$ git clang-format --binary /usr/bin/clang-format-11
~/GitHub/freeciv21$ git add --all
~/GitHub/freeciv21$ git commit
```

The `git status` command is used to list out all the changes that `git` has seen. Changed files, new files, moved files, etc. are all listed. This command comes in handy when you want to organize your Pull Request into more than one commit. It's also very useful to ensure that what you see as changed follows along with what you THINK you have actually changed. Sometimes in the course of editing you may inadvertently made a change to another file that you did not intend to actually change. If you find that you accidentally changed a file, you can use the `git restore <file>` command. This is also shown on the `git status` command output.

The `git add --all` command then adds all the changed files into a single commit. The `git clang-format` command then runs a pre-processor to format any code changes to support the Longturn community's code standards. We use `clang-format-11`. If your system's version of `clang-format` is different, you can use the binaries for `clang-format-11` using the command: `git clang-format --binary <path-to-clang-format-11>`. If `git clang-format` modifies any files, it will tell you so. If that happens, then you need to run a subsequent `git add --all` command to get those files added back into the commit.

Once everything looks good from a `git status` perspective, then issue the last command. The last command, `git commit`, will open a text editor (in Debian based systems this is often `nano`). Put a message at the bottom below all of the hashes (`#`) and then save.

You now have a commit of changes that you need to push to Origin.

5.5.5 5. Requesting a Pull Request from Origin to Upstream

This is the last major step in the process. To push the commit to your fork, issue this command:

```
~/GitHub/freeciv21$ git push origin
```

You will be prompted for your SSH passkey and then the changes in the branch you created in 2. *Create a Branch from Upstream* will be pushed to your Fork. Once complete, you can open your Fork in a web browser and GitHub will notice the change and prompt you to open a Pull Request. The complete steps are documented [here](#).

5.6 Evaluating a Pull Request

This page mostly targets the Longturn Admins who will be asked to review (evaluate) a submitted Pull Request. It is the policy of the Longturn community that all (100%) Pull Requests have at least one reviewer complete an evaluation of a change and either approve or make suggestions for improvements before a merge into the master branch.

5.6.1 Code PRs

For each Pull Request, a set of tests are run automatically. When the tests do not pass, the author is expected to fix the problems before someone tries to test the code.

This page assumes the user knows how to use `git`, compile Freeciv21 and use GitHub.

Create A Testing Branch

```
$ git fetch upstream master
$ git checkout -b testing/pr_[pr-number] upstream/master
```

Get A Copy Of The Pull Request

```
$ wget https://github.com/longturn/freeciv21/pull/[pr-number].diff -O pr[pr-number].diff
```

Apply The Downloaded Update To Local

```
$ git apply pr[pr-number].diff
```

Ensure The Build Directory Is Empty

```
$ rm -rf build_[pr-number]
```

Configure And Compile The Code

```
$ cmake . -B build_[pr-number] -G Ninja -DCMAKE_INSTALL_PREFIX=$PWD/build_[pr-number]/
→install
$ cmake --build build_[pr-number]
$ cmake --build build_[pr-number] --target install
$ cmake --build build_[pr-number] --target package           # MSYS2 and Debian Linux Only
```

Run tests

```
$ cmake --build build_[pr-number] --target test
```

Read The Issue's Notes

Ensure you understand what is being reported. Test the scenario as written. Make notes/comments in the PR. Approve the review request if warranted. If not, state why. If further commits are added to the PR, you will probably have to re-download the diff and run another test.

Run an autogame

If it is a big change, it might be worthwhile to run an entire game with just AI to make sure it does not break anything. You can compile the code, with additional checks such as address sanitizer with `$ cmake . --preset ASan`. Once the code is compiled, you can run the autogame with `$./build_[pr-number]/freeciv21-server -r ./data/test-autogame.serv`. You can also observe the game with `$./build_[pr-number]/freeciv21-client -a -p 5556 -s localhost`. ASan by default halts on every error, this is sometimes useful to developers to fix the errors sequentially. If you'd rather prefer listing all the errors at once, set the environment variable using `$ export ASAN_OPTIONS="halt_on_error=0"`

Cleanup

- Remove the downloaded diff files: `$ rm *.diff`.
- Remove any untracked files: `$ git status`. Look for any untracked files and delete them
- Stash changes: `$ git stash`.
- Checkout the master branch and delete the testing branch:

```
$ git checkout master
$ git branch -d testing/pr_[pr-number]
```

5.6.2 Art PRs

If a Pull Request includes art (graphics, music, etc), you should check not only the inner quality of the art, but also how it fits within what is already there. It is sometimes preferable to use lower quality sprites if they fit better with the general style of a tileset.

A recurring issue with graphics and sound assets is their licensing and attribution. Much more than code, images and music files get copied over, merged, or renamed, and authorship information is quickly lost. Make sure that the author of the PR understands where the files come from and who authored them. If possible, ask the original author directly if we can include their art.

We request that all asset files be accompanied with license and copyright information in the form of a [license file](#). You will find many examples in the repository. The license should be [compatible with version 3 of the GPL](#).

Warning: Please be extra careful when submitted graphics are present on [Freeciv-Web](#), as doubts have been repeatedly raised about the validity of some of the copyright claims made by the main developer of that project. We were also asked not to use their graphics, and we will respect this even if it would be allowed by law. As a rule, we only accept assets present on FCW if we can prove that they were taken from somewhere else — and in that case, we refer to the original source for licensing information.

5.7 The Release Process

The developers of Freeciv21 release new versions of the software following a cadence. The cadence is loosely defined as: 30 non-documentation pull requests (PR's), 200 commits, or 60 days, whichever comes first. We use the term “loosely” as there is a large amount of discretion with regard to the cadence. We can speed up a release at any time to fix a nasty bug or slow a release down to get in a PR a developer is really close to completing and wants it in the next release. However, we do aim to release around 6 to 8 times per year.

Pre-releases are done from the `master` branch, while release candidates and stable releases use the `stable` branch, which contains bug fixes backported from master. See [the dedicated page](#) for how this is done.

These are the general steps to prepare and finalize a release:

1. A release manager will open a draft release notes page from: <https://github.com/longturn/freeciv21/releases>. The easiest way to do this is to copy the contents of the last release, delete the bullet points, and add a single bullet of `* Nothing for this release` to each section. **DO NOT** add a tag at this time. This prevents an accidental release.
2. A release manager will update the draft as PR's are committed to the repository to help keep track of the release cadence progress. As PR's are added to sections the `* Nothing for this release` is removed.
3. Multiple PR's can be combined on a single bullet line. Documentation updates are often done this way.
4. When we are getting close to crossing one of the release candidate thresholds, the release manager will post to the `#releases-project` channel in the LT.DEV section on the Longturn Discord server. The main purpose of the post is to alert the developers of a pending cadence threshold and to gauge if we need to delay in any way or if we are good to proceed as normal.
5. A release manager or another regular contributor will update the `vcpkgGitCommitId` value in the CI/CD build file (`.github/workflows/build.yaml`). Grab the Commit ID of the most recent release from <https://github.com/microsoft/vcpkg/releases>. Commit and push a PR. Ensure all of the CI/CD Action runners complete successfully.
6. When it is time, the release manager will finalize the release notes and ask for an editorial review in the `#releases-project` channel. Updates are made as per review.
7. If the release will be the **first release candidate** towards a stable release, the release manager will:
 1. Delete the existing `stable` branch on Github's [branches page](#).
 2. From the same page, create a new `stable` branch from `master`.
 3. Update `cmake/AutoRevision.txt` with the hash of the last commit in `master` and `v[major version].[minor version]-dev.0` with the version of the **next stable release**, then open a PR for this change to `master`. This way, development builds from `master` will immediately use the version number of the next stable.
8. If the release is a **release candidate** for a **stable release**, the release manager will make sure that the *Target* branch in the release draft is set to `stable`.
9. The release manager will add a tag to the release notes page and then click *Publish Release*. The format of the tag is `v[major version].[minor version]-[pre-release name].[number]` for pre-releases and `v[major version].[minor version].[patch version]` for stable versions. For example: `v3.0-beta.6` or `v3.1.0`. **The format is very important** to the build configuration process.
10. After a few minutes the continuous integration (CI) will open a PR titled `Release Update of AutoRevision.txt`. The release manager will open the PR, click on the *Close pull request* button, and then click *Open pull request* button. This is a necessary step to handle a GitHub security feature. GitHub requires a human to be involved to merge CI created PR's.

11. While inside the `Release Update` of `AutoRevision.txt` PR, the release manager will enable an automatic rebase and merge.
12. The release manager will open an issue titled `Review workarounds after <version> release` with the following text:

We should review the workarounds in the source code and check that they are still needed. Some workarounds are documented here: [Workarounds](#).
13. When all the CI actions are complete, the release manager will make a post in the `#news-channel` on the Longturn.net Discord server.
14. The release manager will download the Windows i686 and x86_64 installer packages and use their Microsoft Account to submit the files for Microsoft SmartScreen analysis. The instructions are [provided here](#). We do this to help our Windows-based users have an easier time downloading the game in the Microsoft Edge browser.
15. The release manager mentions user `@Corbeau` on Discord `#releases-project` channel giving the new URL to update his blog page once all of the GitHub action runners are complete.
16. The release manager mentions user `@panch93` on Discord `#releases-project` channel so he can update the Arch AUR with the latest release.

5.7.1 Behind the Scenes

This section describes how the Continuous Integration (CI) / Continuous Delivery (CD) is setup for Freeciv21 on GitHub.

GitHub's CI/CD is called *Actions* and is enabled via YAML files in this directory in the repository: <https://github.com/longturn/freeciv21/tree/master/.github/workflows>.

There are two files that are integral to the release process: `build.yaml` and `release.yaml`. The `build.yaml` file is the main CI/CD file. It is what runs all of the action “runners” every time a PR is opened or updated with a commit to the repository. You can see the status of the runners on the actions page at: <https://github.com/longturn/freeciv21/actions>. When a release is published, we have code in the file to upload the installers generated by the operating system runner. In this file we generate the binary packages for Windows x86_64 (64 bit), Debian, and macOS. The `release.yaml` file is triggered when we publish a release. This file generates the `.zip` and `.tar.gz` source archives as well as the Windows i686 (32 bit) installer. All of these files are automatically uploaded and attached to the release notes page as assets at the bottom.

5.8 Workarounds

This page lists workarounds in the code. They should be checked from time to time and removed once they are no longer needed.

1. The file `cmake/FindSDL2_mixer.cmake` is needed for `SDL2_mixer 2.0`, which didn't have CMake support. CMake support was introduced in `SDL2_mixer 2.5`, but Ubuntu 22.04 is stuck with `SDL2_mixer 2.0.4`.
2. The macOS CI installs a fixed Python version to work around a bug in Meson. (`.github/workflows/build.yaml`)

5.9 Maintaining the Stable Branch

There are two branches that the development team maintains in our GitHub repository: `master` and `stable`. The `master` branch is the *development* branch and as the name implies, the `stable` branch is for *stable* releases.

Primary development occurs on the `master` branch. Due to the nature of development, it is never guaranteed that the `master` branch will work at all times. While we strive to keep `master` working, there may be occasions where we are introducing major breaking changes that take time to resolve. This is why we have a `stable` branch. As development occurs on the `master` branch, there are going to be times when we want to back-port a commit (a single patch) or a Pull Request (a collection of commits) over to the `stable` branch.

The `stable` branch is created when preparing for the first Release Candidate towards a stable release, see [The Release Process](#). This page documents the rules and procedures for maintaining the `stable` branch.

5.9.1 Requirements for a Back-Port

Only specific categories of commits will be approved for a back-port from `master` to `stable`. They are:

- **Bug Fixes** – Not 100% of bug fixes should be back-ported, but certainly most should be considered. *Breaking* bugs that are found and patched should always be back-ported.
- **Documentation** – We can support a `latest` and `stable` version in our documentation system. If there are issues found in our documentation that explicitly targets `stable`, then we should back-port or author a Pull Request with a target of `stable`.

Things we will not back-port

- **New Features or general improvements** – This is what `master` branch is for.

As we often do, we will allow common sense to dictate any deviations from these rules. However, it should be generally understood that there should be a careful consideration of what to back-port from `master` to `stable`.

5.9.2 Tagging Commits for Back-Port

When a community member authors and publishes a *Pull Request (PR)*, they can add text to the primary comment that the PR is a back-port candidate. If the whole PR is not a candidate, but a specific commit within the PR is, then this distinction should be highlighted as part of the comment. The full Commit ID is the definitive reference point to reduce ambiguity.

As is customary, all PRs should target the `master` branch.

5.9.3 Who Approves a Back-Port Request?

As is our standard, every Pull Request (PR) must have a peer review approval before a merge. The person assigned this task has the authority to approve or deny the back-port request. This authority comes with a catch; the peer reviewer must add the commit(s) to a back-port project that the admins will use to keep track of patches that need to be back-ported.

The back-port project is here: <https://github.com/orgs/longturn/projects/3>. You can find it by going to the main Longturn GitHub repository page and click on the projects tab.

As part of the review process, simply tag the PR to the project and add a `back-port` label to aid tracking. Once the PR has been merged into `master`, go get the commit ID(s) and add those to the appropriate column in the project. Multiple Commit IDs should be separated by a space.

5.9.4 Maintaining Tracked Back-Port Commit Candidates

On a periodic basis, for example every two or three weeks, an administrator will open the project and use the information to build a commit cherry-pick file for git to use.

Follow these steps:

1. Checkout stable on your local: `git checkout stable`
2. Copy and paste the values of the Commit ID column to a plain text file, such as `commits.txt`
3. Run `git cherry-pick -x $(cat commits.txt)`
4. Push stable to upstream: `git push upstream`
5. Update the project sheet to denote which commits were back-ported.

5.10 Documentation Style Guide

The Longturn community uses the Python based Sphinx system to generate the documentation available on this website. Sphinx takes plain text files formatted with a super set of markdown called reStructuredText (ReST). Sphinx reStructuredText is documented here: <https://www.sphinx-doc.org/en/master/usage/restructuredtext/index.html>

This style guide is mostly meant for documentation authors to ensure that we keep a consistent look and feel between authors.

5.10.1 Headings

Headings create chapter breaks between sections of a document as well as provide the title.

Heading 1

Heading 1 is used to for the title of the page/document. The asterisk is used to denote Heading 1 like this:

```
This is an awesome page title
*****
```

Heading 2

Heading 2 is used to create chapter markers or other major breaks in a document. The equal sign is used to denote Heading 2 like this:

```
This is a chapter marker
=====
```

Heading 3

Heading 3 is used to break a chapter into sub-chapters in a document. Heading 3 is not shown on the left side table of contents since we specifically only show 2 levels in the `conf.py` configuration file. The dash is used to denote Heading 3 like this:

```
This is a sub-chapter marker
-----
```

Heading 4

Heading 4 is used to break a sub-chapter down into a further section. This is useful for large pages, such as the *Freeciv21 Hacker's Guide*. The caret is used to denote Heading 4 like this:

This is a sub-chapter break
AAAAAAAAAAAAAAAAAAAAAAAAAAAA

5.10.2 Attribution

As with our code, the documentation is written by contributors and proper attribution should always be given. We follow the [SPDX](#) standard for attribution in all of our documentation, music, and art files. For documentation files, we can add the proper SPDX header to the very top of the page. For music and art we add a text license file along with the primary file. For example, if we have `art.png`, then there should be a corresponding `art.png.license` file to give proper attribution. We use the GPL v3.0 or later license. Here is what the SPDX header should look like in all scenarios:

```
.. SPDX-License-Identifier: GPL-3.0-or-later
.. SPDX-FileCopyrightText: [contributor name or handle] <[contributor email address]>
```

Note: We do not add a date (e.g. year) to our attribution blocks. There is recent commentary that this is not needed and leaving the date off makes keeping header blocks up to date easier.

If the file you are working with came from legacy Freeciv, please add this line to the SPDX header for proper attribution:

```
.. SPDX-FileCopyrightText: Freeciv21 and Freeciv Contributors
```

5.10.3 Interpreted Text Roles

Interpreted text roles are special code blocks that are inserted in line with regular text to create user interface markup elements to bring attention to something or make it more obvious to the reader what you want to do. Interpreted text roles are simply a code word surrounded by a colon on both sides and the text you want to alter is placed inside back-ticks.

- `:doc:` – Doc is used to create a hyperlink reference between documents in the documentation system.
- `:ref:` – Create a cross-reference link to an anchor in another document. This is similar to `:doc:`, except it allows you to go to a specific location within a page, instead of the top of the page. To use `:ref:`, you add an anchor in a page such as `.. _My Anchor:` and then refer to it like this: `:ref:`My Anchor``. Notice that the anchor has an underscore at the beginning. This is required for sphinx to recognize it. Also notice the use of the anchor in `:ref:` leaves the underscore off.
- `:numref:` – Create a cross-reference to a named figure.
- `:table:` – Create a named table reference. Place an anchor (e.g. `.. _My Anchor:`) above to enable `:numref:`.
- `:figure:` – Create a named figure reference. Place an anchor (e.g. `.. _My Anchor:`) above to enable `:numref:`.
- `:emphasis:` – Emphasis is used to *bring attention to something*.
- `:file:` – File is used for file names and paths such as `~/local/share/freeciv21/saves`.
- `:guiabel:` – GUI Label is used to bring attention to something on the screen like the *Next* button on the installer wizard.
- `:literal:` – Literal is used when you want to note a text element in its raw form. This is equivalent to using two back-ticks: ```text```.

- `math` and `.. math::` – Used to insert mathematics, see *Formulas*.
- `:menuselection:` – Menu Selection is used to give the path of menu clicks such as *Game → Local Options*. To denote submenus, use a test arrow like this: `-->` between the selection items.
- `:strong:` – Strong is used to **bold some text**. A good use of `:strong:` is to highlight game elements.
- `:title-reference:` – Title Reference is used to notate a *title entry* in the in-game help or to refer to a page in the documentation without giving an actual hyperlink reference (see `:doc:` above).
- `.. versionadded::` – Used at the paragraph level to document the first version in which a feature was added.

The docutils specification allows for custom Interpreted Text Roles and we use this feature. The docutils documentation on this feature is available here: <https://docutils.sourceforge.io/docs/ref/rst/directives.html#custom-interpreted-text-roles>

- `:unit:` – This provides an opportunity to highlight a Freeciv21 unit, such as the Musketeer
- `:improvement:` – This provides an opportunity to highlight a Freeciv21 building or city improvement, such as the Granary.
- `:wonder:` – This provides an opportunity to highlight a Freeciv21 small or great wonder, such as the Pyramids.
- `:advance:` – This provides an opportunity to highlight a Freeciv21 technology advance, such as Ceremonial Burial.

5.10.4 Admonition Directives

Admonitions are specially marked “topics” that can appear anywhere an ordinary body element can. Typically, an admonition is rendered as an offset block in a document, sometimes outlined or shaded, with a title matching the admonition type. We use some of the standard admonitions in our documentation as well.

- `.. attention::` – Use Attention to bring a very important high profile item to the reader’s attention.

Attention: This is a really important message! Do not forget to eat breakfast every day.

- `.. todo::` – Use To Do as a reminder for documentation editors to come back and fix things at a later date.

Todo: Come back and fix something later.

- `.. note::` – Use the Note as the way to give more information to the reader on a topic.

Note: It is important to note that Freeciv21 is really fun to play with groups of people online.

- `.. code-block:: rst` – The code block is an excellent way to display actual code or any pre-formatted plain text. The tag `rst` can be replaced by `sh`, `cpp`, and `ini` as well to give different types of markup for shell commands, C++ code, and ini file formatting.

This is a code block showing some pre-formatted text.

5.10.5 Language Usage Elements

The documentation is written mostly in US English (en_US), however elements of Queen’s English (e.g. en_GB) are also found in the documentation. The two forms of English are close enough that we do not worry too much if one author uses “color” and another uses “colour”. Any reader or language translator will be able to figure out what the author is trying to say. However, there are some standards that documentation authors do need to adhere to, so the documentation is consistently formatted and certain language elements are always used the same way.

The Oxford Comma

The Oxford Comma is the usage of a comma when listing multiple items and placing a comma before the “and” or “or” at the end of the list. For example: You need to follow these steps: Click on *Menu*, then click on *Options*, and finally click on *Interface*. Notice the comma usage before the word “and”, that is the Oxford comma and its usage is expected in our documentation.

Capitalization

For consistent formatting, the following should always use “Title Case” rules:

- Page and section headings (e.g. the 4 documented above).
- Image captions, when they act as a title to the image.
- The names of specific game items such as units, city improvements, technologies, wonders, etc. Some of them even have special text roles (:unit:, :improvement:, and :wonder:). [See here for a list.](#)

This is particularly useful with words that are used ambiguously in the game, such as “granary” which is both the amount of food a city needs before growing and an improvement in many rulesets. Another example is “transport” which covers both the movement of units on a ship and the particular unit type of Transport.

When describing elements of the user interface, use the same capitalization as in the game and wrap the text inside markup elements with the :guilabel: or :menuselection: roles. They are rendered as follows: “the *Turn Done* button”, “select *Help* → *Overview* in the menu”.

List of Capitalized Terms

This page contains a list of terms that should be capitalized when used in the documentation. It is most likely incomplete. Please refer to the [Documentation Style Guide](#) page for more information.

Buildings and Wonders

Use them with :improvement: or :wonder:, such as Granary and Pyramids.

- | | | |
|------------------------------|---------------------------|--------------------------|
| • 2nd Palace | • Coinage | • Hanging Gardens |
| • Airport | • Colosseum | • Harbor |
| • Amphitheater | • Colossus | • Hoover Dam |
| • Apollo Program | • Copernicus’ Observatory | • Hospital |
| • Aqueduct | • Courthouse | • Hydro Plant |
| • A.Smith’s Trading Co. | • Cure For Cancer | • Internet |
| • Atlantic Telegraph Company | • Darwin’s Voyage | • Isaac Newton’s College |
| • Bank | • Ecclesiastical Palace | • J.S. Bach’s Cathedral |
| • Barracks | • Eiffel Tower | • King Richard’s Crusade |
| • Barracks II | • Factory | • Leonardo’s Workshop |
| • Barracks III | • Granary | • Library |
| • Cathedral | • Great Library | • Lighthouse |
| • City Walls | • Great Wall | • Magellan’s Expedition |
| • Coastal Defense | • Hal Saffieni Hypogeum | • Manhattan Project |

- Marco Polo's Embassy
- Marketplace
- Mass Transit
- Mausoleum of Mausolos
- Mercantile Exchange
- Mfg. Plant
- Michelangelo's Chapel
- Nuclear Plant
- Offshore Platform
- Oracle
- Palace
- Police Station
- Port Facility
- Power Plant
- Pyramids
- Recycling Center
- Research Lab
- SAM Battery
- SDI Defense
- Secret Police
- SETI Program
- Sewer System
- Shakespeare's Theater
- Solar Plant
- Space Component
- Space Module
- Space Structural
- Statue of Liberty
- Statue of Zeus
- Stock Exchange
- Sun Tzu's War Academy
- Super Highways
- Supermarket
- Temple
- Temple of Artemis
- The Internet
- Trade Company
- Training Facility
- Transportation
- United Nations
- University
- Verrocchio's Workshop
- Women's Suffrage

Diplomacy

Only when referring to the in-game diplomatic state. For instance: “establishing an Alliance allows an increased level of cooperation”, but “one should always be prepared for war”.

- Alliance
- Armistice
- Cease-fire
- Peace
- Shared Vision
- War

Disasters

- Earthquake
- Famine
- Fire
- Flood
- Global Warming
- Industrial Accident
- Nuclear
- Accident
- Nuclear Winter
- Plague

Governments

- Anarchy
- City-States
- Communism
- Democracy
- Despotism
- Federation
- Fundamentalism
- Monarchy
- Republic
- Tribal

Resources

- Buffalo
- Coal
- Fish
- Fruit
- Furs
- Game
- Gems
- Gold
- Iron
- Ivory
- Oasis
- Oil
- Peat
- Pheasant
- Re-
- sources
- River
- Silk
- Spice
- Whales
- Wheat
- Wine

Specialists

- Entertainer
- Scientist
- Taxman

Terrains

- Desert
- Glacier
- Hills
- Lake
- Ocean
- Swamp
- Forest
- Grassland
- Jungle
- Mountains
- Plains
- Tundra

Technologies

Use them with :advance:, such as Advanced Espionage.

- Advanced Espionage
- Advanced Flight
- Agriculture
- Alphabet
- Amphibious Warfare
- Astronomy
- Atomic Theory
- Automobile
- Aviation Endurance
- Banking
- Bridge Building
- Bronze Working
- Ceremonial Burial
- Chemistry
- Chivalry
- Code of Laws
- Combined Arms
- Combustion
- Communism
- Computers
- Conscription
- Construction
- Currency
- Democracy
- Economics
- Electricity
- Electronics
- Engineering
- Environmentalism
- Espionage
- Explosives
- Feudalism
- Flight
- Fundamentalism
- Fusion Power
- Genetic Engineering
- Guerilla Warfare
- Gunpowder
- Horseback Riding
- Industrialization
- Invention
- Iron Working
- Labor Union
- Laser
- Leadership
- Literacy
- Machine Tools
- Maglev
- Magnetism
- Map Making
- Martial Law
- Masonry
- Mass Production
- Mathematics
- Medicine
- Metallurgy
- Miniaturization
- Mobile Warfare
- Monarchy
- Monotheism
- Mysticism
- Navigation
- Nuclear Fission
- Nuclear Power
- Philosophy
- Physics
- Plastics
- Polytheism
- Pottery
- Radio
- Recycling
- Refining
- Refrigeration
- Religion
- Road Building
- Robotics
- Rocketry
- Sanitation
- Seafaring
- Space Flight
- Specialist Training
- Stealth
- Steam Engine
- Steel
- Superconductors
- Supermodule
- Tactics
- Railroad
- The Corporation
- Theology
- Theory of Evolution
- Theory of Gravity
- The Republic
- The Wheel
- Trade
- University
- Warrior Code
- Writing

Terrain Alterations

- Airbase
- Buoy
- Fallout
- Farm-land
- Fortress
- Irriga-tion
- Mine
- Minor Tribe
- Village
- Maglev
- Oil Well
- Pollu-tion
- Rail-road
- Road
- Ruins

Units

Use them with `:unit:`, such as Settlers.

- | | | | |
|--------------------|---------------------|----------------------------|-------------------------|
| • AEGIS Cruiser | • Destroyer | • Infantry | • Riflemen |
| • Alpine Troops | • Diplomat | • Intercontinental Missile | • Scholar |
| • Archers | • Dragoons | • Ironclad | • Scribe |
| • Armor | • Elephants | • Knights | • Settlers |
| • Artillery | • Engineers | • Legion | • Spy |
| • AWACS | • Explorer | • Longboat | • Square-Rigged Caravel |
| • Barbarian Leader | • Fanatics | • Marines | • Stealth Bomber |
| • Barge | • Fighter | • Mech. Inf. | • Stealth Fighter |
| • Battleship | • Flagship Frigate | • Migrants | • Stealth Spy |
| • Bomber | • Freight | • Militia | • Submarine |
| • Cannon | • Frigate | • Missile | • Swordsmen |
| • Caravan | • Fusion Armor | • Musketeers | • Tank |
| • Caravel | • Fusion Battleship | • Nuclear | • Transport |
| • Cargo Aircraft | • Fusion Bomber | • Nuclear Bomb | • Trireme |
| • Carrier | • Fusion Fighter | • Nuclear Submarine | • Leader |
| • Catapult | • Galleon | • Operative | • Tribal Workers |
| • Cavalry | • Helicopter | • Paratroopers | • Workers |
| • Chariot | • Horsemen | • Partisan | • Warriors |
| • Cruise Missile | • Howitzer | • Phalanx | |
| • Cruiser | • ICBM | • Pikemen | |
| • Crusaders | • Immigrants | | |

Unit Classes

- | | | | | |
|--------------|--------------|------------|--------------|-----------|
| • Air | • Big Land | • Land | • Sea | • Trireme |
| • Amphibious | • Big Siege | • Merchant | • Small Land | |
| | • Helicopter | • Missile | | |

Language Contractions

Language Contractions are when two words are combined together with an apostrophe ('). For example, the word “don’t” is a contraction of “do not”. Not all language translators, and especially non-native English speakers can get confused if contractions are used. To aid the readability of our documentation, **the usage of contractions is not advised** and should be used sparingly.

The Use of Person

In English there are three types of person: first, second, and third. First person is possessive – “I took a walk down the street”. Second person is about speaking to someone – “You took a walk down the street”. Third person is non-specific – “They took a walk down the street”. In our documentation we use the second person form. We want to be conversational with our readers and speak to them about the game, features, actions, etc.

This page provides a good overview of the use person for US English: <https://www.grammar-monster.com/glossary/person.htm>

Double Negatives / Negations

To aid the readability of our documentation, we want to stay away from using double negatives. A double negative is where two negative words are combined together that end with a positive. For example: “The guidelines are not bad”. The last two words are negative – “not bad”. It is better to use positive language. For example the first sentence is better written as: “The guidelines are good”.

Figure Numbers

Diagrams, Screenshots, and Tables are **expected** to be numbered using the `numfig` feature of Sphinx. For example see this code block for a figure:

```
.. _Start Screen:
.. figure:: /_static/images/gui-elements/start-screen.png
   :scale: 65%
   :align: center
   :alt: Freeciv21 Start Screen
   :figclass: align-center

Start Screen with NightStalker Theme
```

The first line `.. _Start Screen:` is a label for the figure. The `numfig` feature of Sphinx will automatically give the figure a number in the order they are found in the page. You can then provide a link to the figure in your text with `:numfig:`Label``

5.10.6 Formulas

The Freeciv21 documentation supports inserting mathematics. This feature should be used sparingly, ideally only on technical pages or in sections that less math-savvy users can skip. When math formulas are used on non-technical pages (such as any one of the manuals), the reasoning should be relatively simple following [elementary algebra](#). Contrary to ordinary math textbooks, it is best to avoid single-letter symbols in the documentation. Full-text names should be used instead, wrapping them with `\text{}`:

$$\text{happy} \geq \text{unhappy} + 2 \times \text{angry}.$$

There may be exceptions to this rule on primarily technical pages: quantities that exist as variables in the code could be typeset in monospace with `\texttt{}`, or defining a few symbols may come handy when writing a long reasoning. The main guideline for formulas is to take your time to make them as readable as possible.

Formulas use the `:math:` role or the `.. math::` directive. These blocks support most of the LaTeX [mathematics syntax](#). The `:math:` role is used for inline math in a paragraph. For instance, `:math:`a+b=1`` becomes $a + b = 1$. The directive is used for longer or more important formulas that come on their own line:

$$a + b = 1.$$

This is rendered using an `align` environment, so alignment directives (`&`) can be used.

Warning: When editing formulas, checking both the HTML and the PDF output is heavily recommended.

CODING

The Coding category is a set of technical documentation of varying aspects of the Freeciv21 code base. The audience of this documentation is software developers seeking information on how the code is setup and architected.

6.1 Architecture

Freeciv21 uses a client/server model where two programs interact over a network connection. The *client* (`freeciv21-client`) is the program that you see on the screen. It handles user commands and forwards them to the second program, that we call the *server* (`freeciv21-server`). The server checks the commands, computes their result and sends updates back to the client. The client is capable of managing a server for its own needs, which is used for single player games. It can also connect to a server managed externally, which is how multiplayer games work.

The Freeciv21 code is old and evolved from a program written in C. It still bears a lot of this history, but this is being worked on. New code is written in C++, using modern constructs when it makes sense and relying heavily on the [Qt Framework](#) for platform abstraction. This is currently most visible in the client, but the rest of the code base is slowly being converted as well.

6.1.1 Repository Organization

The source code is organized in directories at the root of the repository. The client and server share a lot of code found in the `common` and `utility` folders. The first one contains code to manage the game state, of which both the client and server have a copy, and the second is home to various lower-level utilities that do not have an equivalent in Qt or the C standard library. Freeciv21 also ships with a couple of external dependencies under `dependencies`.

The client code is found in the `client` folder. The server code is located under `server`, with the exception of the computer (*AI*) players which is under `ai`. The code for other programs bundled with Freeciv21, such as the [Modpack Installer](#), is located under `tools`.

All the assets used by the client and server are grouped under `data`. This includes among others [rulesets](#) and [tilesets](#). Localization files are located under `translations`.

There are a few additional folders that you will touch less often. The table below describes the complete structure of the repository:

Table 6.1: Freeciv21 Code Repository Organization

Folder	Usage
ai	Code for computer opponents.
client	Client code.
cmake	Build system support code.
common	Code dealing with the game state. Shared by the client, server, and tools.
data	Game assets.
dependencies	External dependencies not found in package managers.
dist	Files related to distributing Freeciv21 for various operating systems.
docs	This documentation.
scripts	Useful scripts used by the maintainers.
server	Server code.
tools	Small game-related programs.
translations	Localization.
utility	Utility classes and functions not found in Qt or other dependencies.

Note: Some folders do not follow this structure. Their contents should eventually be moved.

6.2 Freeciv21 Hacker's Guide

This guide is intended to be a help for developers, wanting to mess with Freeciv21 programming.

6.2.1 The Server

Server Autogame Testing

Code changes should always be tested before submission for inclusion into the [GitHub](#) source tree. It is useful to run the client and server as autogames to verify either a particular savegame no longer shows a fixed bug, or as a random sequence of games in a while loop overnight.

To start a server game with all *AI* players, create a file (below named `civ.serv`) with lines such as the following:

```
# set gameseed 42      # repeat a particular game (random) sequence
# set mapseed 42       # repeat a particular map generation sequence
# set timeout 3        # run a client/server autogame
set timeout -1         # run a server only autogame
set minplayers 0       # no human player needed
set ec_turns 0         # avoid timestamps in savegames
set aifill 7           # fill to 7 players
hard                  # make the AI do complex things
create Caesar         # first player (with known name) created and
                      # toggled to AI mode
start                 # start game
```

The commandline to run server-only games can be typed as variations of:

```
$ while( time build/freeciv21-server -r civ.serv ); do date; done
--- or ---
$ build/freeciv21-server -r civ.serv -f buggy1534.sav.gz
```

To attach one or more clients to an autogame, remove the `start` command, start the server program and attach clients to created *AI* players. Or type `aitoggle <player>` at the server command prompt for each player that connects. Finally, type `start` when you are ready to watch the show.

Note: The server will eventually flood a client with updates faster than they can be drawn to the screen, thus it should always be throttled by setting a timeout value high enough to allow processing of the large update loads near the end of the game.

If you plan to compare results of autogames the following changes can be helpful:

- define `__FC_LINE__` to a constant value in `./utility/log.h`.
- deactivation of the event cache (set `ec_turns 0`).

6.2.2 Old Lists

For variable length list of units and cities Freeciv21 uses a `genlist`, which is implemented in `utility/genlist.cpp`. By some macro magic type specific macros have been defined, creating a lot of trouble for C++ programmers. These macro-based lists are being phased out in favor of STL containers. In the meantime, we preserve here an explanation of how to use them.

For example a `tile` struct (the pointer to it we call `ptile`) has a `unit` list, `ptile->units`; to iterate though all the units on the tile you would do the following:

```
unit_list_iterate(ptile->units, punit) {
    // In here we could do something with punit, which is a pointer to a
    // unit struct
} unit_list_iterate_end;
```

Note that the macro itself declares the variable `punit`. Similarly there is a

```
city_list_iterate(pplayer->cities, pcity) {
    // Do something with pcity, the pointer to a city struct
} city_list_iterate_end;
```

There are other operations than iterating that can be performed on a list; inserting, deleting, or sorting etc. See `utility/speclist.h`. Note that the way the `*_list_iterate` macro is implemented means you can use “continue” and “break” in the usual manner.

One thing you should keep in the back of your mind. Say you are iterating through a unit list, and then somewhere inside the iteration decide to disband a unit. In the server you would do this by calling `wipe_unit(punit)`, which would then remove the unit node from all the relevant unit lists. However, by the way `unit_list_iterate` works, if the removed unit was the following node `unit_list_iterate` will already have saved the pointer, and use it in a moment, with a segfault as the result. To avoid this, use `unit_list_iterate_safe` instead.

6.2.3 Graphics

Currently the graphics is stored in the PNG file format.

If you alter the graphics, then make sure that the background remains transparent. Failing to do this means the mask-pixmaps will not be generated properly, which will certainly not give any good results.

Each terrain tile is drawn in 16 versions, all the combinations with a green border in one of the main directions. Hills, Mountains, Forests, and Rivers are treated in special cases.

Isometric tilesets are drawn in a similar way to how civ2 draws (that is why civ2 graphics are compatible). For each base terrain type there exists one tile sprite for that terrain. The tile is blended with nearby tiles to get a nice-looking boundary. This is erroneously called “dither” in the code.

Non-isometric tilesets draw the tiles in the “original” Freeciv21 way, which is both harder and less pretty. There are multiple copies of each tile, so that a different copy can be drawn depending on the terrain type of the adjacent tiles. It may eventually be worthwhile to convert this to the civ2 system or another one altogether.

6.2.4 Map Structure

The map is maintained in a pretty straightforward C array, containing $X \times Y$ tiles. You can use the function `struct tile *map_pos_to_tile(x, y)` to find a pointer to a specific tile. A tile has various fields; see the struct in `common/map.h`.

You may iterate tiles, you may use the following methods:

```
whole_map_iterate(tile_itr) {  
    // do something  
} whole_map_iterate_end;
```

for iterating all tiles of the map;

```
adjc_iterate(center_tile, tile_itr) {  
    // do something  
} adjc_iterate_end;
```

for iterating all tiles close to `center_tile`, in all *valid* directions for the current topology (see below);

```
cardinal_adjc_iterate(center_tile, tile_itr) {  
    // do something  
} cardinal_adjc_iterate_end;
```

for iterating all tiles close to `center_tile`, in all *cardinal* directions for the current topology (see below);

```
square_iterate(center_tile, radius, tile_itr) {  
    // do something  
} square_iterate_end;
```

for iterating all tiles in the radius defined `radius` (in real distance, see below), beginning by `center_tile`;

```
circle_iterate(center_tile, radius, tile_itr) {  
    // do something  
} square_iterate_end;
```

for iterating all tiles in the radius defined `radius` (in square distance, see below), beginning by `center_tile`;

```
iterate_outward(center_tile, real_dist, tile_itr) {
    // do something
} iterate_outward_end;
```

for iterating all tiles in the radius defined `radius` (in real distance, see below), beginning by `center_tile`. Actually, this is the duplicate of `square_iterate`, or various tricky ones defined in `common/map.h`, which automatically adjust the tile values. The defined macros should be used whenever possible, the examples above were only included to give people the knowledge of how things work.

Note that the following:

```
for (x1 = x-1; x1 <= x+1; x1++) {
    for (y1 = y-1; y1 <= y+1; y1++) {
        // do something
    }
}
```

is not a reliable way to iterate all adjacent tiles for all topologies, so such operations should be avoided.

Also available are the functions calculating distance between tiles. In Freeciv21, we are using 3 types of distance between tiles:

- The `map_distance(ptile0, ptile1)` function returns the *Manhattan* distance between tiles, i.e. the distance from `ptile0` to `ptile1`, only using cardinal directions. For example, $|dx| + |dy|$ for non-hexagonal topologies.
- The `real_map_distance(ptile0, ptile1)` function returns the *normal* distance between tiles, i.e. the minimal distance from `ptile0` to `ptile1` using all valid directions for the current topology.
- The `sq_map_distance(ptile0, ptile1)` function returns the *square* distance between tiles. This is a simple way to make Pythagorean effects for making circles on the map for example. For non-hexagonal topologies, it would be $dx^2 + dy^2$. Only useless square root is missing.

Different Types of Map Topology

Originally Freeciv21 supports only a simple rectangular map. For instance a 5x3 map would be conceptualized as

```
<- XXXXX ->
<- XXXXX ->
<- XXXXX ->
```

and it looks just like that under “overhead” (non-isometric) view. The arrows represent an east-west wrapping. But under an isometric-view client, the same map will look like:

```
<-  X    ->
<-  X X   ->
<-  X X X  ->
<-  X X X  ->
<-  X X X  ->
<-   X X   ->
<-    X    ->
```

where “north” is to the upper-right and “south” to the lower-left. This makes for a mediocre interface.

An isometric-view client will behave better with an isometric map. This is what Civ2, SMAC, Civ3, etc. all use. A rectangular isometric map can be conceptualized as

```
<- X X X X X ->
<- X X X X X ->
<- X X X X X ->
<- X X X X X ->
```

North is up and it will look just like that under an isometric-view client. Of course under an overhead-view client it will again turn out badly.

Both types of maps can easily wrap in either direction: north-south or east-west. Thus there are four types of wrapping: flat-earth, vertical cylinder, horizontal cylinder, and torus. Traditionally Freeciv21 only wraps in the east-west direction.

Topology, Cardinal Directions and Valid Directions

A *cardinal* direction connects tiles per a *side*. Another *valid* direction connects tiles per a *corner*.

In non-hexagonal topologies, there are 4 cardinal directions, and 4 other valid directions. In hexagonal topologies, there are 6 cardinal directions, which matches exactly the 6 valid directions.

Note that with isometric view, the direction named “North” (DIR8_NORTH) is actually not from the top to the bottom of the screen view. All directions are turned a step on the left (e.g. $\pi \div 4$ rotation with square tiles and $\pi \div 3$ rotation for hexagonal tiles).

Different Coordinate Systems

In Freeciv21, we have the general concept of a “position” or “tile”. A tile can be referred to in any of several coordinate systems. The distinction becomes important when we start to use non-standard maps (see above).

Here is a diagram of coordinate conversions for a classical map.

map		natural		native		index
ABCD		ABCD		ABCD		
EFGH	<=>	EFGH	<=>	EFGH	<=>	ABCDEFGHIJKL
IJKL		IJKL		IJKL		

Here is a diagram of coordinate conversions for an iso-map.

map		natural		native		index
CF		A B C		ABC		
BEIL	<=>	D E F	<=>	DEF	<=>	ABCDEFGHIJKL
ADHK		G H I		GJI		
GJ		J K L		JKL		

Below each of the coordinate systems are explained in more detail. Note that hexagonal topologies are always considered as isometric.

Map (or “Standard”) Coordinates

All of the code examples above are in map coordinates. These preserve the local geometry of square tiles, but do not represent the global map geometry well. In map coordinates, you are guaranteed, so long as we use square tiles, that the tile adjacency rules

```
| (map_x-1, map_y-1)  (map_x, map_y-1)  (map_x+1, map_y-1)
| (map_x-1, map_y)    (map_x, map_y)    (map_x+1, map_y)
| (map_x-1, map_y+1)  (map_x, map_y+1)  (map_x+1, map_y+1)
```

are preserved, regardless of what the underlying map or drawing code looks like. This is the definition of the system.

With an isometric view, this looks like:

```
|                      (map_x-1, map_y-1)
|              (map_x-1, map_y)          (map_x, map_y-1)
| (map_x-1, map_y+1)          (map_x, map_y)          (map_x+1, map_y-1)
|              (map_x, map_y+1)          (map_x+1, map_y)
|                      (map_x+1, map_y+1)
```

Map coordinates are easiest for local operations (e.g. ‘square_iterate’ and friends, translations, rotations, and any other scalar operation) but unwieldy for global operations.

When performing operations in map coordinates (like a translation of tile (x, y) by (dx, dy) -> (x + dx, y + dy)), the new map coordinates may be unsuitable for the current map. In case, you should use one of the following functions or macros:

- `map_pos_to_tile()`: return NULL if normalization is not possible;
- `normalize_map_pos()`: return TRUE if normalization have been done (wrapping X and Y coordinates if the current topology allows it);
- `is_normal_map_pos()`: return TRUE if the map coordinates exist for the map;
- `CHECK_MAP_POS()`: assert whether the map coordinates exist for the map.

Map coordinates are quite central in the coordinate system, and they may be easily converted to any other coordinates: `MAP_TO_NATURAL_POS()`, `MAP_TO_NATIVE_POS()`, or `map_pos_to_index()` functions.

Natural Coordinates

Natural coordinates preserve the geometry of map coordinates, but also have the rectangular property of native coordinates. They are unwieldy for most operations because of their sparseness. They may not have the same scale as map coordinates and, in the iso case, there are gaps in the natural representation of a map.

With classical view, this looks like:

```
(nat_x-1, nat_y-1)  (nat_x, nat_y-1)  (nat_x+1, nat_y-1)
(nat_x-1, nat_y)    (nat_x, nat_y)    (nat_x+1, nat_y)
(nat_x-1, nat_y+1)  (nat_x, nat_y+1)  (nat_x+1, nat_y+1)
```

With an isometric view, this looks like:

```
|                      (nat_x, nat_y-2)
|              (nat_x-1, nat_y-1)          (nat_x+1, nat_y-1)
| (nat_x-2, nat_y)          (nat_x, nat_y)          (nat_x+2, nat_y)
```

(continues on next page)

(continued from previous page)

```
|          (nat_x-1, nat_y+1)          (nat_x+1, nat_y+1)
|                                (nat_x, nat_y+2)
```

Natural coordinates are mostly used for operations which concern the user view. It is the best way to determine the horizontal and the vertical axis of the view.

The only coordinates conversion is done using the `NATURAL_TO_MAP_POS()` function.

Native Coordinates

With classical view, this looks like:

```
(nat_x-1, nat_y-1)  (nat_x, nat_y-1)  (nat_x+1, nat_y-1)
(nat_x-1, nat_y)    (nat_x, nat_y)    (nat_x+1, nat_y)
(nat_x-1, nat_y+1)  (nat_x, nat_y+1)  (nat_x+1, nat_y+1)
```

With an isometric view, this looks like:

```
|          (nat_x, nat_y-2)
|      (nat_x-1, nat_y-1)      (nat_x, nat_y-1)
| (nat_x-1, nat_y)      (nat_x, nat_y)      (nat_x+1, nat_y)
|      (nat_x-1, nat_y+1)      (nat_x, nat_y+1)
|          (nat_x, nat_y+2)
```

Neither is particularly good for a global map operation such as map wrapping or conversions to or from map indexes. Something better is needed.

Native coordinates compress the map into a continuous rectangle. The dimensions are defined as `map.xsize` x `map.ysize`. For instance, the above iso-rectangular map is represented in native coordinates by compressing the natural representation in the X axis to get the 3x3 iso-rectangle of

```
ABC      (0,0) (1,0) (2,0)
DEF  <=>  (0,1) (1,1) (2,1)
GHI      (0,2) (1,2) (3,2)
```

The resulting coordinate system is much easier to use than map coordinates for some operations. These include most internal topology operations (e.g., `normalize_map_pos`, or `whole_map_iterate`) as well as storage (in `map.tiles` and `savegames`, for instance).

In general, native coordinates can be defined based on this property; the basic map becomes a continuous (gap-free) cardinally-oriented rectangle when expressed in native coordinates.

Native coordinates can be easily converted to map coordinates using the `NATIVE_TO_MAP_POS()` function, to index using the code: `native_pos_to_index()` function and to tile (shortcut) using the `native_pos_to_tile()` function.

After operations, such as the `FC_WRAP(x, map.xsize)` function, the result may be checked with the `CHECK_NATIVE_POS()` function.

Index Coordinates

Index coordinates simply reorder the map into a continuous (filled-in) one-dimensional system. This coordinate system is closely tied to the ordering of the tiles in native coordinates, and is slightly easier to use for some operations (like storage) because it is one-dimensional. In general you cannot assume anything about the ordering of the positions within the system.

Indexes can be easily converted to native coordinates using the `index_to_native_pos()` function or to map positions (shortcut) using the `index_to_map_pos()` function.

A map index can be tested using the `CHECK_INDEX` macro.

With a classical rectangular map, the first three coordinate systems are equivalent. When we introduce isometric maps, the distinction becomes important, as demonstrated above. Many places in the code have introduced `map_x/map_y` or `nat_x/nat_y` to help distinguish whether map or native coordinates are being used. Other places are not yet rigorous in keeping them apart, and will often just name their variables `x` and `y`. The latter can usually be assumed to be map coordinates.

Note that if you do not need to do some abstract geometry exploit, you will mostly use tile pointers, and give to map tools the ability to perform what you want.

Note that `map.xsize` and `map.ysize` define the dimension of the map in `_native_` coordinates.

Of course, if a future topology does not fit these rules for coordinate systems, they will have to be refined.

Native Coordinates on an Isometric Map

An isometric map is defined by the operation that converts between map (user) coordinates and native (internal) ones. In native coordinates, an isometric map behaves exactly the same way as a standard one. See *Native Coordinates*, above.

Converting from map to native coordinates involves a $\pi \div 2$ rotation (which scales in each dimension by $\sqrt{2}$) followed by a compression in the **X** direction by a factor of 2. Then a translation is required since the “normal set” of native coordinates is defined as (x, y) where $\{x \mid 0 \leq x < \text{map.xsize}\}$ and $\{y \mid 0 \leq y < \text{map.ysize}\}$ while the normal set of map coordinates must satisfy $x \geq 0$ and $y \geq 0$.

Converting from native to map coordinates (a less cumbersome operation) is the opposite.

				EJ
	ABCDE	A B C D E		DIO
(native)	FGHIJ <=>	F G H I J <=>	CHN	(map)
	KLMNO	K L M N O	BGM	
			AFL	
			K	

Note that:

```
native_to_map_pos(0, 0) == (0, map.xsize-1)
native_to_map_pos(map.xsize-1, 0) == (map.xsize-1, 0)
native_to_map_pos(x, y+2) == native_to_map_pos(x,y) + (1,1)
native_to_map_pos(x+1, y) == native_to_map_pos(x,y) + (1,-1)
```

The math then works out to:

$$\begin{aligned} x_{\text{map}} &= \left\lceil \frac{y_{\text{nat}}}{2} \right\rceil + x_{\text{nat}} \\ y_{\text{map}} &= \left\lfloor \frac{y_{\text{nat}}}{2} \right\rfloor - x_{\text{nat}} + x_{\text{size}} - 1 \end{aligned}$$

$$y_{\text{nat}} = x_{\text{map}} + y_{\text{map}} - x_{\text{size}}$$
$$x_{\text{nat}} = \left\lfloor x_{\text{map}} - y_{\text{map}} + \frac{x_{\text{size}}}{2} \right\rfloor$$

which leads to the macros `NATIVE_TO_MAP_POS()`, and `MAP_TO_NATIVE_POS()` that are defined in `map.h`.

Unknown Tiles and Fog of War

In `common/player.h`, there are several fields:

```
struct player {
    ...
    struct dbv tile_known;

    union {
        struct {
            ...
        } server;

    struct {
        struct dbv tile_vision[V_COUNT];
    } client;
    };
};
```

While `tile_get_known()` returns:

```
// network, order dependent
enum known_type {
    TILE_UNKNOWN = 0,
    TILE_KNOWN_UNSEEN = 1,
    TILE_KNOWN_SEEN = 2,
};
```

The values `TILE_UNKNOWN` and `TILE_KNOWN_SEEN` are straightforward. `TILE_KNOWN_UNSEEN` is a tile of which the user knows the terrain, but not recent cities, roads, etc.

`TILE_UNKNOWN` tiles never are (nor should be) sent to the client. In the past, `UNKNOWN` tiles that were adjacent to `UNSEEN` or `SEEN` were sent to make the drawing process easier, but this has now been removed. This means exploring new land may sometimes change the appearance of existing land (but this is not fundamentally different from what might happen when you transform land). Sending the extra info, however, not only confused the goto code but allowed cheating.

Fog of War is the fact that even when you have seen a tile once you are not sent updates unless it is inside the sight range of one of your units or cities.

We keep track of Fog of War by counting the number of units and cities of each client that can see the tile. This requires a number per player, per tile, so each `player_tile` has a `short[]`. Every time a unit, city, or something else can observe a tile 1 is added to its player's number at the tile, and when it cannot observe any more (killed/moved/pillaged) 1 is subtracted. In addition to the initialization/loading of a game this array is manipulated with the `void unfog_area(struct player *pplayer, int x, int y, int len)` and `void fog_area(struct player *pplayer, int x, int y, int len)` functions. The `int len` variable is the radius of the area that should be fogged/unfogged, i.e. a `len` of 1 is a normal unit. In addition to keeping track of Fog of War, these functions also make sure to reveal `TILE_UNKNOWN` tiles you get near, and send information about `TILE_UNKNOWN` tiles near that the client needs for drawing. They then send the tiles to the `void send_tile_info(struct player *dest, int x, int y)` function, which then sets the correct `known_type` and sends the tile to the client.

If you want to just show the terrain and cities of the square the function `show_area()` does this. The tiles remain fogged. If you play without Fog of War all the values of the seen arrays are initialized to 1. So you are using the exact same code, you just never get down to 0. As changes in the “foginess” of the tiles are only sent to the client when the value shifts between zero and non-zero, no redundant packages are sent. You can even switch Fog of War on or off in game just by adding or subtracting 1 to all the tiles.

We only send city and terrain updates to the players who can see the tile. So a city, or improvement, can exist in a square that is known and fogged and not be shown on the map. Likewise, you can see a city in a fogged square even if the city does not exist. It will be removed when you see the tile again. This is done by 1) only sending info to players who can see a tile and 2) keeping track of what info has been sent so the game can be saved. For the purpose of 2), each player has a map on the server (consisting of `player_tile` and `dumb_city` fields) where the relevant information is kept.

The case where a player `p1` gives map info to another player `p2` requires some extra information. Imagine a tile that neither player sees, but which `p1` has the most recent information on. In that case the age of the players’ information should be compared, which is why the player tile has a `last_updated` field. This field is not kept up to date as long as the player can see the tile and it is unfogged, but when the tile gets fogged the date is updated.

There is a Shared Vision feature, meaning that if `p1` gives Shared Vision to `p2`, every time a function like `show_area()`, `fog_area()`, `unfog_area()`, or `give_tile_info_from_player_to_player()` is called on `p1`, `p2` also gets the information. Note that if `p2` gives Shared Vision to `p3`, `p3` also gets the information for `p1`. This is controlled by `p1`’s `really_gives_vision` bitvector, where the dependencies will be kept.

National Borders

For the display of national Borders (similar to those used in Sid Meier’s Alpha Centauri) each map tile also has an `owner` field, to identify which nation lays claim to it. If `game.borders` is non-zero, each city claims a circle of tiles `game.borders` in Vision Radius. In the case of neighbouring enemy cities, tiles are divided equally, with the older city winning any ties. Cities claim all immediately adjacent tiles, plus any other tiles within the border radius on the same continent. Land cities also claim ocean tiles if they are surrounded by 5 land tiles on the same continent. This is a crude detection of inland seas or Lakes, which should be improved upon.

tile ownership is decided only by the server, and sent to the clients, which draw border lines between tiles of differing ownership. Owner information is sent for all tiles that are known by a client, whether or not they are fogged.

6.3 Artificial Intelligence (AI)

This document is about Freeciv21’s default *AI*.

Warning: The contents of this page are over 20 years old and have not been reviewed for correctness.

6.3.1 Introduction

The Freeciv21 *AI* is capable of developing complex nations and provides a challenging opponent for beginners. It is, however, still too easy for experienced players, mostly due to it being very predictable. Code that implements the *AI* is divided between the `ai/` and `server/advisors` code directories. The latter is used also by human players for automatic helpers such as auto-settlers and auto-explorers.

6.3.2 Long-Term AI Development Goals

The long-term goals for Freeciv21 *AI* development are:

- To create a challenging and fun *AI* for human players to defeat.
- To create an *AI* that can handle all the ruleset possibilities that Freeciv21 can offer, no matter how complicated or unique the implementation of the rules.

6.3.3 Want Calculations

Build calculations are expressed through a structure called `adv_choice`. This has a variable called “want”, which determines how much the *AI* wants whatever item is pointed to by `choice->type`. `choice->want` is:

Table 6.2: AI Want Ranges

Value	Result
-199	<code>get_a_boat</code>
< 0	an error
== 0	no want, nothing to do
<= 100	normal want
> 100	critical want, used to requisition emergency needs
> ???	probably an error (1024 is a reasonable upper bound)
> 200	Frequently used as a cap. When want exceeds this value, it is reduced to a lower number.

These are ideal numbers, your mileage while travelling through the code may vary considerably. Technology and Diplomats, in particular, seem to violate these standards.

6.3.4 Amortize

Hard fact: `amortize(benefit, delay)` returns $\text{benefit} \times \left(1 - \frac{1}{\text{MORT}}\right)^{\text{delay}}$.

Speculation: What is better... to receive \$10 annually starting in 5 years from now or \$5 annually starting from this year? How can you take inflation into account? The function `amortize()` is meant to help you answer these questions. To achieve this, it re-scales the future benefit in terms of today’s money.

Suppose we have a constant rate of inflation, x percent. Then in five years \$10 will buy as much as $10 \left(\frac{100}{100+x}\right)^5$ will buy today. Denoting $\frac{100}{100+x}$ by q we get the general formula, N dollars, Y years from now will be worth $N \times q^Y$ in today’s money. If we receive N every year starting Y years from now, the total amount receivable (in today’s money) is $\frac{N \times q^Y}{1-q}$. This is the sum of infinite geometric series. This is exactly the operation that the `amortize()` function performs, the multiplication by some $q < 1$ raised to power Y . Note that the factor $\frac{1}{1-q}$ does not depend on the parameters N and Y , and can be ignored. The connection between the `MORT` constant and the inflation rate x is given by $\frac{\text{MORT}-1}{\text{MORT}} = q = \frac{100}{100+x}$. Thus the current value of `MORT` = 24 corresponds to the inflation rate, or the rate of expansion of your civilization of 4.3%

Most likely this explanation is not what the authors of `amortize()` had in mind, but the basic idea is correct: the value of the payoff decays exponentially with the delay.

The version of `amortize` used in the military code (`military_amortize()`) remains a complete mystery.

6.3.5 Estimation of Profit From a Military Operation

This estimation is implemented by the `kill_desire()` function, which is not perfect, the multi-victim part is flawed, plus some corrections. In general:

$$\text{Want} = \text{Operation_Profit} \times \text{Amortization_Factor}$$

where `Amortization_Factor` is a function of the estimated time length of the operation and `Operation_Profit` = `Battle_Profit` – `Maintenance`, where in turn `Maintenance` = (`Support` + `Unhappiness_Compensation`) × `Operation_Time`

Here `Unhappiness_Compensation` is from a military unit being away from home and `Support` is the number of Shields spent on supporting this unit per turn.

$$\begin{aligned} \text{Battle_Profit} = & \text{Shields_Lost}_{\text{enemy}} \times \text{Probability}_{\text{win}} \\ & - \text{Shields_Lost}_{\text{us}} \times \text{Probability}_{\text{lose}} \end{aligned}$$

That is `Battle_Profit` is a probabilistic average. It answers the question: “How much better off, on average, would we be from attacking this enemy unit?”

6.3.6 Selecting Military Units

The code dealing with choosing military units to be built and targets for them is especially messy.

Military units are requested in the `military_advisor_choose_build()` function. It first considers the defensive units and then ventures into selection of attackers (if home is safe). There are two possibilities here: we just build a new attacker or we already have an attacker which was forced, for some reason, to defend. In the second case it is easy: we calculate how good the existing attacker is and if it is good, we build a defender to free it up.

Building a brand new attacker is more complicated. First, the `ai_choose_attacker_*` functions are called to find the first approximation to the best attacker that can be built here. This prototype attacker is selected using very simple `attack_power` × `speed` formula. Then, already in the `kill_something_with()` function, we search for targets for the prototype attacker using the `find_something_to_kill()` function. Having found a target, we do the last refinement by calling the `process_attacker_want()` function to look for the best attacker type to take out the target. This type will be our attacker of choice. Note that the function `process_attacker_want()` function has side-effects with regards to the Technology selection.

Here is an example:

First the `ai_choose_attacker_land()` function selects a Dragoon because it is strong and fast. Then the `find_something_to_kill()` function finds a victim for the (virtual) Dragoon, an enemy Riflemen standing right next to the city. Then the `process_attacker_want()` function figures out that since the enemy is right beside us, it can be taken out easier using an Artillery. It also figures that a Howitzer would do this job even better, so bumps up our desire for Robotics.

6.3.7 Ferry System

The ferry (i.e. boats transporting land units) system of Freeciv21 is probably better described by statistical mechanics than by logic. Both ferries and prospective passengers move around in what looks like a random fashion, trying to get closer to each other. On average, they succeed. This behavior has good reasons behind it. It is hell to debug, but means that small bugs do not affect the overall picture visibly.

Each turn both boats and prospective passengers forget all about prior arrangements (unless the passenger is actually *in* the boat). Then each will look for the closest partner, exchange cards, and head towards it. This is done in a loop which goes through all units in random order.

Because most units recalculate their destination every turn, ignoring prior arrangements is the only good strategy. It means that a boat will not rely on the prospective passenger to notify it when it is not needed anymore. This is not very effective, but can only be changed when the prospective passengers behave more responsibly. See the Diplomat code for more responsible behavior. They try to check if the old target is still good before trying to find a new one.

When a boat has a passenger, it is a different story. The boat does not do any calculations, instead one of the passengers is given full control and it is the passenger who drives the boat.

Here are the main data fields used by the system. Value of `ai.ferry` in the passenger unit is:

- `FERRY_NONE` : means that the unit has no need of a ferry.
- `FERRY_WANTED` : means that the unit wants a `ferry >0` : id of its ferry.

Value of `ai.passenger` in the ferry unit can be either of:

- `FERRY_AVAILABLE` : means that the unit is a ferry and is `available >0` : id of its passenger.

When boat-building code stabilizes, it can be seen how many free boats there are, on average, per prospective passenger. If there are more boats than prospective passengers, it makes sense that only prospective passengers should look for boats. If boats are few, they should be the ones choosing. This can be done both dynamically, where both possibilities are coded and the appropriate is chosen every turn, and statically, after much testing only one system remains. Now they exist in parallel, although each developed to a different degree.

6.3.8 Diplomacy

The *AI*'s diplomatic behaviour is current only regulated by the `diplomacy` server setting.

AI proposes Cease-fire on first contact.

AI is not very trusting for `NEUTRAL` and `PEACE` modes, but once it hits `ALLIANCE`, this changes completely, and it will happily hand over any technologies and maps it has to you. The only thing that will make the *AI* attack you then is if you build a Spaceship.

For people who want to hack at this part of the *AI* code, please note:

- The `pplayers_at_war(p1,p2)` function returns `FALSE` if `p1==p2`
- The `pplayers_non_attack(p1,p2)` function returns `FALSE` if `p1==p2`
- The `pplayers_allied(p1,p2)` function returns `TRUE` if `p1==p2`
- The `pplayer_has_embassy(p1,p2)` function returns `TRUE` if `p1==p2`

For example, we do not ever consider a Nation to be at War with themselves, we never consider a Nation to have any kind of non-attack treaty with themselves, and we always consider a Nation to have an Alliance with itself.

The introduction of Diplomacy is fraught with many problems. One is that it usually benefits only human players and not *AI* players, since humans are so much smarter, and know how to exploit Diplomacy. For *AI*'s, they mostly only add constraints on what it can do. This means Diplomacy either has to be optional, or have fine-grained controls on who can do what Diplomatic deals to whom, which are set from rulesets. The latter is not yet well implemented.

6.3.9 Difficulty Levels

There are currently seven difficulty levels:

1. Handicapped
2. Novice
3. Easy
4. Normal
5. Hard
6. Cheating
7. Experimental

The **hard** level is no-holds-barred. **Cheating** is the same except that it has ruleset defined extra bonuses, while **normal** has a number of handicaps. In **easy**, the *AI* also does random stupid things through the `ai_fuzzy()` function. In **novice** the *AI* researches slower than normal players. The **experimental** level is only for coding. You can gate new code with the `H_EXPERIMENTAL` handicap and test **experimental** level *AI*'s against **hard** level *AI*'s.

Other handicaps used are:

Table 6.3: AI Difficulty Levels

Variable	Result
<code>H_DIPLOMAT</code>	Cannot build offensive Diplomats.
<code>H_LIMITEDHUTS</code>	Can get only 25 gold and Barbarians from Huts.
<code>H_DEFENSIVE</code>	Build defensive buildings without calculating need.
<code>H_RATES</code>	Cannot set its national budget rates beyond government limits.
<code>H_TARGETS</code>	Cannot target anything it does not know exists.
<code>H_HUTS</code>	Does not know which unseen tiles have Huts on them.
<code>H_FOG</code>	Cannot see through fog of War.
<code>H_NOPLANES</code>	Does not build air units.
<code>H_MAP</code>	Only knows <code>map_is_known</code> tiles.
<code>H_DIPLOMACY</code>	Not very good at Diplomacy.
<code>H_REVOLUTION</code>	Cannot skip Anarchy.
<code>H_EXPANSION</code>	Do not like being much larger than human.
<code>H_DANGER</code>	Always thinks its city is in danger.

For an up-to-date list of all handicaps and their use for each difficulty level see `ai/handicaps.h`.

6.3.10 Things That Need To Be Fixed

- Cities do not realize units are on their way to defend it.
- *AI* builds cities without regard to danger at that location.
- *AI* will not build cross-country roads outside of the city vision radius.
- `Locally_zero_minimap` is not implemented when wilderness tiles change.
- If no path to a chosen victim is found, a new victim should be chosen.
- Emergencies in two cities at once are not handled properly.
- Explorers will not use ferryboats to get to new lands to explore. The *AI* will also not build units to explore new islands, leaving Huts alone.

- *AI* sometimes believes that wasting a horde of weak military units to kill one enemy is profitable.
- Stop building shore defense improvements in landlocked cities with a Lake adjacent.
- Fix the *AI* valuation of Supermarket. It currently never builds it. See the `farmland_food()` and `ai_eval_buildings()` functions in `advdomestic.cpp`.
- Teach the *AI* to coordinate the units in an attack.

6.3.11 Idea Space

- Friendly cities can be used as beachheads.
- The `Assess_danger()` function should acknowledge positive feedback between multiple attackers.
- It would be nice for a bodyguard and charge to meet en-route more elegantly.
- The `struct choice` should have a priority indicator in it. This will reduce the number of “special” want values and remove the necessity to have want capped, thus reducing confusion.

6.4 Attribute Blocks

Each client player has an attribute block and the server also holds such an attribute block for every player. All attribute blocks the server holds are included in the save game. The client and server synchronize their blocks. The server sends its block to the client at game start or reload. The client sends an updated block at each end of turn to the server. Since the maximum packet size is limited to currently 4kB and the attribute block can have arbitrary size (the limit is currently 256kB), the attribute block cannot be transferred in one packet. Therefore, the attribute block is divided into attribute chunks which are reassembled at the receiver. No part of the server knows any inner structure of the attribute block. For the server an attribute block is just a block of bytes.

6.4.1 User Interface

Since an attribute block is not a good user interface the user can access the attributes through a mapping, dictionary, hashmap, or hashtable interface. This hashtable will get serialized to the attribute block and the other way around. The key of the hashtable consists of: the (real) key, x, y, and an id. The (real) key is an integer, which defines the use and format of this attribute. The values of the hashtable can have arbitrary length. The internal structure of a value is opaque to the attribute handling.

For easier access there are wrapper functions for the common case of city attributes, that are used to store *governor* data.

6.5 Coding Guidelines

This page contains a set of guidelines regarding the preferred coding style for new code. Old code follows different standards and should not be modified for the sole purpose of making it follow the guidelines.

In a nutshell, new or refactored Freeciv21 code is written in modern C++ (the standard version evolves depending on what the targeted compilers support). This coding style strives to achieve a balance between old code originally written in C and the new possibilities brought by C++, so that new and old code do not look too different. At this time we are targeting C++17.

The rules listed on this page are guidelines. They represent the preferred way of writing code for Freeciv21, but good code that does not follow the guidelines will not be rejected directly. However, you should expect suggestions for changes.

6.5.1 General

Code Formatting Rules

Freeciv21 uses an automated code formatting tool, `clang-format` version 11, to handle the most tedious parts of code formatting. All code must be formatted with `clang-format`. This relieves the admins from checking that you used the correct amount of whitespace after a comma, and you from having to fix it manually. The downside is that the tool is really picky and will sometimes want to reformat good-looking code. Even the maintainers are sometimes surprised! You will find more information about `clang-format` and detailed instructions in [How to Submit a Pull Request](#).

Formatting the code as dictated by `clang-format` is **mandatory**.

Copyright Notice

New code follows the [SPDX](#) standard:

```
/*
 * SPDX-License-Identifier: GPL-3.0-or-later
 * SPDX-FileCopyrightText: Freeciv21 and Freeciv Contributors
 * SPDX-FileCopyrightText: Author Name <how-to-contact@example.com>
 */
```

You do not need to add your name to the file, but we recommend that you do so. Even if the same information is stored in the Git history, it may be lost if someone ends up moving the file.

Header Guards

Use `#pragma once` to protect headers against multiple inclusion:

```
#pragma once
```

This is equivalent to the ancient `#ifndef/#define/#endif` idiom, but much less error-prone. The syntax is supported by all mainstream compilers.

Includes

Included headers coming from the same place should be grouped together: first all Freeciv21 headers grouped by source directory, then Qt headers, and finally headers from the Standard Library:

```
// common
#include "tile.h"
#include "unit.h"

// client
#include "layer.h"

// Qt
#include <QString>
#include <QWidget>

// std
#include <map>
```

This order forces Freeciv21 headers to include the Qt and Standard Library headers they need, facilitating their use in other files.

In `cpp` files, the header of the same name should always be included first.

Forward Declarations

Whenever possible, use forward declarations in headers instead of adding an `#include`. This can greatly speed up compilation:

```
class QLabel;  
class QLineEdit;
```

One can use a forward declaration if the class is only used to declare a pointer or reference variable, possibly as a function argument.

Documentation Comments

The function of all public entities (classes, functions, enumerations, enumerators, variables, ...) should be described in a [Doxygen](#)-enabled comment. Classes and functions should use a multiline comment:

```
/**  
 * Shows a message box greeting the user.  
 */  
void greet()  
{  
    QMessageBox::information(nullptr, _("Greetings"), _("Hello, user"));  
}
```

These comments serve two purposes:

- They help the reader understand the code.
- They act as separators between functions in `cpp` files.

Single-line comments can be used for very simple methods whose implementation is included in a class definition, as well as for less complex constructs such as enumerations and variables. The use of Doxygen [markup commands](#) to provide more detailed descriptions is welcome, but in no way mandatory.

Two highly recommended Doxygen markup commands to include are `\file` and `\class`. When writing code, especially a new `cpp` file, including a comment with the `\file` markup near the top, but below the [Copyright Notice](#) gives the reader a better understanding of what the file's purpose is. If the `cpp` file contains functions related to a class, then using the `\class` markup aids the reader in understanding what the class is doing.

Variable Declaration

Older Freeciv21 code will often have a block of variables defined all at once at the top of a function. The problem with this style is it makes it very easy to create variables that never get used or initialized.

When encountering this older style or writing new code, it is best to define the variable and give it an initial value right before it is used.

```
... some code  
... some code
```

(continues on next page)

(continued from previous page)

```
for (int i = 0; i < max_item; i++) {
    ... do something in the loop
}
```

Naming Convention

The developers have not agreed on a naming convention yet. In the meantime, most code has been following the former practice of using `all_lowercase_letters` in most cases. The only exception to this rule is for constant values (enumeration values and `const static` variables), for which `UPPERCASE` is generally used.

Private member variables should be prefixed with `m_` and be placed at the bottom of the class:

```
class something
{
public:
    explicit something();
    virtual ~something();

private:
    int m_foo;
};
```

The freeciv Namespace

The `freeciv` namespace has been used to group classes created during refactoring efforts. This code is expected to follow higher standards than the rest of the code base, such as encapsulation and having minimal side effects.

The Anonymous Namespace

Symbols that are used in a single file, as support for other functions, should be defined in the anonymous namespace:

```
namespace /* anonymous */ {

const int IMPORTANT_CONSTANT = 5; ///Very, very important

/**
 * Calculates a more important value of @c x.
 */
int some_internal_function(int x)
{
    return x + IMPORTANT_CONSTANT;
}

} // anonymous namespace
```

The compiler will generate arbitrary names for symbols in the anonymous namespace that will not clash with symbols defined elsewhere.

Premature Optimization

It is often useless to try and optimize a function before proving that it is inefficient by profiling the execution in an optimized build (Release or RelWithDebInfo). Most functions in Freeciv21 are not executed in tight loops. Prefer readable code over fast code.

6.5.2 C++ Features

C++ is a very complex language, but fortunately Freeciv21 only needs to use a relatively small subset. Qt, our main dependency, manages very well to minimize user exposure to confusing parts. If all you are doing is small changes here and there, you will most likely not need to know a lot about C++. As your projects grow in scale and complexity, you will likely want to learn more about the language. In addition to your preferred learning resources, it is useful to read guidelines written by C++ experts, for instance the [C++ Core Guidelines](#) edited by the very founder of C++.

We collect below a list of recommendations that we find useful in the context of Freeciv21.

Pass by Reference

When writing a function that takes a complex object (anything larger than a `long long`), use a constant reference:

```
QString foo(const QString &argument);  
int bar(const std::vector<int> &argument);
```

Use const

Variables that are not modified should be declared `const`. While this is more of a personal preference for variables, it is especially important for functions taking references (see above).

Functions that do not modify their argument should make them `const`. Class methods that do not modify the object should also be marked `const`.

Use Encapsulation

Classes that are more complicated than C-like `struct` should not have any public variables. Getters and setters should be provided when needed.

Use auto

The `auto` keyword is useful to avoid typing the type of a variable, especially lengthy names used in the Standard Library. We recommend to use it whenever possible. Do *not* try to use `auto` for function arguments.

```
const auto &unit = tile->units.front();
```

Use STL Containers

Containers in the Standard Library should be preferred over Qt ones:

```
std::vector<unit *> foo;
std::map<int, int> bar;
```

One notable exception is `QStringList`, which should be preferred over other constructs because it integrates better with Qt.

This can have consequences on function signatures. For instance, a common C idiom is to pass an output array and a size as function parameter, whereas this is achieved in C++ by returning a `std::vector` (or, if performance is critical, passing a reference to a vector that the function fills).

Use `<algorithm>`

The C++ Standard Library provides a set of [basic algorithms](#). Code using the standard algorithms is often more clear than hand-written loops, if only because experienced programmers will recognize the function name immediately.

Use Range-based for

Avoid using indices to iterate over containers. Prefer the much simpler range-based `for`:

```
for (const auto &city : player->cities) {
    // ...
}
```

Use Structured Bindings

Structured bindings are very useful when facing a `std::pair`, for instance when iterating over a map:

```
for (const auto &[key, value] : map) {
```

If you do not wish to use one of the variables, use `_`:

```
for (const auto &[key, _] : map) {
    // Use the key only
}
```

Use Smart Pointers

Instead of using `new` and `delete`, delegate the task to a smart pointer:

```
auto result = std::make_unique<cm_result>();
```

When facing a memory handling bug such as a double free, it is sometimes easier to rewrite the code using smart pointers than to understand the issue.

Smart pointers are rarely needed with Qt classes. The [parent-child mechanism](#) is the preferred way of handling ownership for classes deriving from `QObject`. In many other cases, Qt classes are meant to be used directly on the stack. This is valid for `QString`, `QByteArray`, `QColor`, `QPixmap`, and many others. If you are unsure, try to find an example in the Qt documentation.

Qt provides its own smart pointer for `QObject`, called `QPointer`. This pointer tracks the lifetime of the pointed-to object and is reset to `nullptr` if the object gets deleted. This is useful in some situations.

Todo: We would like to include some tips about the following topics in the future:

- Logging
 - Qt Tips
-

6.6 Internationalization

Internationalization, or `i18n` for short, refers to making sure software is ready to be adapted for use in a language and country other than the original. Freeciv21 uses American English (`en-US`) as the primary target language, but can be localized to other regions.

You should keep internationalization in mind when writing code that interacts with the user. Here are a few things to take into account:

- Text should be translated to the language of the user. Freeciv21 has facilities to handle this, see below.
- Widgets should resize automatically to fit the text. In a language like Russian, text is often twice as long as in English.
- The visual arrangement of items on screen should match the natural direction of the language. Arabic and Hebrew, for instance, write text from right to left. This is in many cases handled automatically by Qt.

Paying attention to these three points is sufficient to get internationalization right in the vast majority of the cases.

6.6.1 Translating Text

Translating user-facing text is the most work-intensive part of internationalization. As a developer, you only need to make sure that strings are marked for translation when appropriate. In Freeciv21, this is done using a set of macros based on the `gettext` library (defined in `utility/fcintl.h`). The most simple one returns a translated version of a string:

```
const char *translation = _("some text");
```

In this example, “some text” will be collected automatically and handed over to the translators. If there is a translation for the locale used by the user, it will be used automatically instead of the original English text.

Dealing with plurals requires a different function. Indeed, pluralization rules are very complicated in some languages, so this is handled in another macro:

```
int count = ...;
const char *translation = PL("%1 translation", "%1 translations", count);
const QString with_number = QString(translation).arg(count);
```

Notice how you still need to replace `%1` by yourself, since this is not handled by the macro. In actual code, the intermediate variable would be simplified away:

```
const QString with_number =
    QString(PL("%1 translation", "%1 translations", count)).arg(count);
```

In some cases, one needs to disambiguate between different uses of the same string. For instance, `None` is used in different places to mean “no technology” and “no wonder”. This is handled by `Q_`:

```
const char *translation = Q_("?wonder:None");
```

Finally, there is also a macro `N_()` that marks a string for translation without doing the translation (it returns the original). This is useful in static contexts.

6.6.2 Not Translated

While most text should be translated, there are a few cases where this is not wanted:

- **File Formats:** It must be possible to load a saved game independently of the locale that was in use when producing it. The same stands for rulesets, tilesets, and other file formats used by Freeciv21.
- **Fatal Error Messages:** If the game encounter such a critical error that it needs to abort, it may not have the resources to produce a translation.

6.6.3 Helper Comments

When translators work on strings, they are provided with a list taken out of context. They can see the original text in English and sometimes the source code, but most translators cannot read code. In some cases, the lack of context makes translation very hard. The string “Close”, for instance, can have many meanings, with the correct one being inferred from context: “near”, “closed”, “close (a door)”, “stop”, etc. Each meaning calls for a different translation. One can add special comments to the code to help translators identify the correct variant:

```
// TRANS: Close the current window
close_button->setText(_("Close"));
```

The comment will be picked up if it is on the line before the translated text. These comments are typically very useful when building text from different parts using placeholders (%1, %2, etc.). In such cases, a comment should be added to explain what the final string looks like:

```
// TRANS: <Unit> (<Home city>)
text += QString::format(_("%1 (%2)")).arg(unit_type_name).arg(home_city_name);

// TRANS: "HP: 5/20 MP: 5/5" in unit description. Keep short
text += QString::format(_("HP: %1/%2 MP: %3/%4"))
    .arg(hp)
    .arg(max_hp)
    .arg(mp)
    .arg(max_mp);
```

In complex cases, adding an example or a short explanation also makes the code easier to read.

6.6.4 Character Encodings

The way characters are encoded into strings has long been a hot topic of internationalization, and language-specific character encodings are still around on some systems. Freeciv21 always uses UTF-8 for data files and internal communication (e.g. in the network protocol). The recommended way of storing text is with Qt’s `QString` class, which uses UTF-16 internally.

The `QString` constructor performs the conversion from UTF-8 automatically when passed a `char *` argument. In the opposite direction, `qUtf8Printable()` takes a `QString` and returns a *temporary* `char *` encoded in UTF-8, which is deleted automatically at the next semicolon.

Text can be converted to the system encoding using `qPrintable` or `QString::toLocal8Bit`. This should be rarely, if ever, needed.

6.6.5 Common Difficulties

Every language is different, and there is no reason for the order of words or even sentences to be the same as in English. When possible, it is thus preferable to provide the translators with full sentences or paragraphs. If you can speak several languages, it is also useful to think about how to translate your text: you may find a way to simplify it and facilitate its translation. In this section, we describe a few issues that we have encountered.

Freeciv21 cannot handle more than one plural in the same string. Imagine the following text:

```
%1 units, %2 buildings, %3 wonders
```

Since the `PL_()` macro takes a single numeric parameter, only one of the words can be pluralized correctly. There is currently no fully satisfactory solution to this problem. A slightly better version would be an enumeration:

```
Units: %1, buildings: %2, wonders: %3
```

In English and some other languages, this form is correct even if there is only one unit.

Another difficulty shows up when dynamically inserting words in a sentence. This works extremely well in English, but in many other languages this leads to incorrect grammar, with genders and declension being common culprits. For example, consider the following simplified version of the “unit lost” message:

Legion lost to an attack by a Greek Catapult.

This message has three dynamic parts: the unit types and the nationality of the attacker. Let us now look at the correct French translation:

Légion perdue dans une attaque d'une Catapulte grecque.

If you look closely enough, you will notice that this is pretty close: “perdue” is “lost” and the order of “Greek” and “Catapult” needs to be swapped. But what if the attacker is a Cannon and the defender a Musketeer?

Mousquetaire **perdu** dans une attaque **d'un** Canon **grec**.

The words in bold in the main sentence had to be changed to match the new units. There is currently no real solution to this problem in Freeciv21, and translators resort to use incorrect grammar.

6.7 Logging

Freeciv21 has built-in support for printing debugging information at runtime. This is very valuable for developers who want to check their code or a system administrator to monitor their servers. Five levels are available:

- **fatal** reports errors that cause the application to exit immediately.
- **critical** is for recoverable errors caused by a problem in the code or an input file.
- **warning** corresponds to a couple of errors that may affect the game state. The players are also notified using in-game messages.
- **info** is used for generic messages aimed at the user.
- **debug** messages are typically not shown, but might provide additional hints when there is a problem.

The minimum level for which messages are printed can be set by passing the `-d <level>` argument to Freeciv21. The default is **info**.

Freeciv21 uses Qt for its messages, so the many Qt configuration options can be used. See [the Qt documentation](#) for details. The system can be tweaked to show only the messages you want at run time, for instance by setting `QT_LOGGING_RULES`. The complete docs are at [QLoggingCategory](#). The following categories are currently defined:

Table 6.4: Logging Parameters

Name	Used for
<code>freeciv.stacktrace</code>	Stack traces in debug builds.
<code>freeciv.assert</code>	Assertion errors.
<code>freeciv.inputfile</code>	spec file parser.
<code>freeciv.bugs</code>	When the code finds a bug in itself.
<code>freeciv.timers</code>	Various timers.
<code>freeciv.depr</code>	Deprecation warnings.
<code>freeciv.cm</code>	Citizen manager messages.
<code>freeciv.ruleset</code>	Ruleset issues.
<code>freeciv.goto</code>	Path finding debug messages.
<code>freeciv.graphics</code>	FPS counters.

For instance, one could disable stack traces and enable Go To debug message by setting:

```
export QT_LOGGING_RULES="freeciv.stacktrace*=false\nfreeciv.goto.debug=true"
```

In addition to tweaking categories, system administrators may want to set `QT_LOGGING_TO_CONSOLE` to `0` to force use of the system logging facilities.

When Freeciv21 is built in Debug mode, many messages are added to its debug output and it becomes very, very verbose. This is not the case for other build types, for example Release and RelWithDebInfo. Freeciv21 will try and print a detailed stack trace on fatal and critical errors, but this is limited by the availability of debugging symbols.

6.8 Network Protocol

All information passed between the client and server must be sent through the network. Freeciv21 uses a custom binary protocol. The data flowing between the client and server is segmented in *packets*, each with a specific meaning and data contents. In most cases, it is enough to know about the packets themselves. Unless you intend to modify the low-level details, you can safely ignore how packets are serialized to binary data and transported over the network.

6.8.1 Packets

The packets used in the protocol are defined in `common/networking/packets.def`. Each packet has a type identifier (a number) and some fields. The packet definition file is parsed by `common/generate_packets.py`, which generates a set of C++ functions to serialize them. For each `foo` packet structure, there is one send and one receive function:

```
int send_packet_foo(struct connection *pc, struct packet_foo *packet);
struct packet_foo * receive_packet_foo(struct connection *pc);
```

The `send_packet_foo()` function serializes a structure into a bytestream and adds this to the send buffer in the connection struct. The `receive_packet_foo()` function de-serializes a bytestream into a structure and removes the bytestream from the input buffer in the connection struct. The connection struct is defined in `common/connection.h`.

Each structure field in a structure is serialized using architecture independent functions such as `dio_put_uint32()` and de-serialized with functions like `dio_get_uint32()`.

A packet is constituted by a header followed by the serialized structure data. The header contains the following fields (the sizes are defined in `common/packets.cpppacket_header_set()`):

```
uint16 : length (the length of the entire packet)
uint16 : type   (e.g. PACKET_TILE_INFO)
```

For backward compatibility reasons, packets used for the initial protocol (notably before checking the capabilities) have different header fields sizes as defined in `common/packets.cpacket_header_init()`:

```
uint16 : length (the length of the entire packet)
uint8  : type   (e.g. PACKET_SERVER_JOIN_REQ)
```

To demonstrate the route for a packet through the system, here is how a unit disband is performed:

Note: This example is old and talks about packets that have been removed. The overall logic still stands, but some details have changed.

1. A player disbands a unit.
2. The client initializes a `packet_unit_request` structure and calls the packet layer function `send_packet_unit_request()` with this structure and packet type: `PACKET_UNIT_DISBAND`.
3. The packet layer serializes the structure, wraps it up in a packet containing the `packetlength` type and the serialized data. Finally, the data is sent to the server.
4. On the server the packet is read. Based on the type, the corresponding de-serialize function is called by the `get_packet_from_connection()` function.
5. A `packet_unit_request` is initialized with the bytestream.
6. Since the incoming packet is a request, the server sends a `PACKET_PROCESSING_STARTED` packet to the client. A request in this context is every packet sent from the client to the server.
7. Finally the corresponding packet-handler, the `handle_unit_disband()` function, is called with the newly constructed structure.
8. The handler function checks if the disband request is legal (i.e. the sender really the owner of the unit), etc.
9. The unit is disbanded via the `wipe_unit()` and `send_remove_unit()` functions.
10. Now an integer, containing the id of the disbanded unit is wrapped into a packet along with the type `PACKET_REMOVE_UNIT`: `send_packet_generic_integer()`.
11. The packet is serialized and sent across the network.
12. The packet-handler returns and the end of the processing is announced to the client with a `PACKET_PROCESSING_FINISHED` packet.
13. On the client the `PACKET_REMOVE_UNIT` packet is deserialized into a `packet_generic_integer` structure.
14. The corresponding client handler function is now called `handle_remove_unit()`, and finally the unit is disbanded.

Notice that the two packets (`PACKET_UNIT_DISBAND` and `PACKET_REMOVE_UNIT`) were generic packets. That means the packet structures involved, are used by various requests. The `packet_unit_request()` function is for example also used for the packets `PACKET_UNIT_BUILD_CITY` and `PACKET_UNIT_CHANGE_HOMECITY`.

The `PACKET_PROCESSING_STARTED` and `PACKET_PROCESSING_FINISHED` packets from above serve two main purposes:

1. They allow the client to identify what causes a certain packet the client receives. If the packet is framed by `PACKET_PROCESSING_STARTED` and `PACKET_PROCESSING_FINISHED` packets it is the causes of the request. If not the received packet was not caused by this client (server operator, other clients, server at a new turn)
2. After a `PACKET_PROCESSING_FINISHED` packet the client can test if the requested action was performed by the server. If the server has sent some updates the client data structure will now hold other values.

The `PACKET_FREEZE_HINT` and `PACKET_THAW_HINT` packets serve two purposes:

1. Packets sent between these two packets may contain multiple information packets which may cause multiple updates of some GUI items. `PACKET_FREEZE_HINT` and `PACKET_THAW_HINT` can now be used to freeze the GUI at the time `PACKET_FREEZE_HINT` is received and only update the GUI after the `PACKET_THAW_HINT` packet is received.
2. Packets sent between these two packets may contain contradicting information which may confuse a client-side AI (agents for example). So any updates sent between these two packets are only processed after the `PACKET_THAW_HINT` packet is received.

The following areas are wrapped by `PACKET_FREEZE_HINT` and `PACKET_THAW_HINT`:

- The data sent if a new game starts.
- The data sent to a reconnecting player.
- The end turn activities.

6.8.2 Network Improvements

In the past, when a connection send buffer in the server got full we emptied the buffer contents and continued processing. Unfortunately, this caused incomplete packets to be sent to the client, which caused crashes in either the client or the server, since the client cannot detect this situation. This has been fixed by closing the client connection when the buffer is emptied.

We also had, and still have, several problems related to flow control. Basically the problem is the server can send packets much faster than the client can process them. This is especially true when in the end of the turn the AIs move all their units. Unit moves in particular take a long time for the client to process since by default smooth unit moves is on.

There are 4 ways to solve this problem:

1. We wait for the send buffers to drain before continuing processing.
2. We cut the player's connection and empty the send buffer.
3. We lose packets (this is similar to 2), but can cause an incoherent state in the client.
4. We modify the client to deal with a fast server gracefully.

We mitigated the problem by increasing the send buffer size on the server and making it dynamic. We also added in strategic places in the code calls to a new `flush_packets()` function that makes the server stall for some time draining the send buffers. Strategic places include whenever we send the whole map. The maximum amount of time spent per `flush_packets()` call is specified by the `netwait` variable.

To disconnect unreachable clients, the server pings the client after a certain time elapses (set using the `pingtimeout` variable). If the client does not reply its connection is closed.

6.8.3 Delta

If delta is enabled for this packet, the packet-payload (after the bytes used by the packet-header) is followed by the `delta-header`. The `delta-header` is a bitvector which represents all non-key fields of the packet. If the field has changed the corresponding bit is set and the field value is also included in `delta-body`. The values of the unchanged fields will be filled in from an old version at the receiving side. The old version filled in from is the previous packet of the same kind that has the same value in each key field. If the packet's kind do not have any key fields the previous packet of the same kind is used. If no old version exists the unchanged fields will be assumed to be zero.

For a `bool` field, another optimization called `bool-header-folding` is applied. Instead of sending an indicator in the bitvector if the given `bool` value has changed, and so using 1 byte for the real value, the actual value of the `bool` is transferred in the bitvector bit of this `bool` field.

Another optimization called `array-diff` is used to reduce the amount of elements transferred if an array is changed. This is independent of the `delta-header` bit, i.e. it will only be used if the array has changed its value and the bit indicates this. Instead of transferring the whole array only a list of `index` and `new value of this index` pairs are transferred. The `index` is 8 bit and the end of this pair list is denoted by an `index` of 255.

For fields of struct type (or arrays of struct) the following function is used to compare entries, where `foo` stands for the name of the struct:

```
bool are_foo_equal(const struct foo *a, const struct foo *b);
```

The declaration of this function must be made available to the generated code by having it `#include` the correct header. The includes are hard-coded in `generate_packets.py`.

6.8.4 Compression

To further reduce the network traffic between the client and the server, the (delta) packets are compressed using the DEFLATE compression algorithm. To get better compression results, multiple packets are grouped together and compressed into a chunk. This chunk is then transferred as a normal packet. A chunk packet starts with the 2 byte `length` field, which every packet has. A chunk packet has no type. A chunk packet is identified by having a too large `length` field. If the length of the packet is over `COMPRESSION_BORDER`, it is a chunk packet. It will be uncompressed at the receiving side and re-fed into the receiving queue.

If the `length` of the chunk packet cannot be expressed in the available space of the 16bit `length` field (>48kb), the chunk is sent as a jumbo packet. The difference between a normal chunk packet and a jumbo chunk packet is that the jumbo packet has `JUMBO_SIZE` in the `size` field and has an additional 4 byte `length` field after the 2 byte `length` field. The second `length` field contains the size of the whole packet (2 byte first `length`` field + 4 byte second ```length` field + compressed data). The `size` field of a normal chunk packet is its `size` + `COMPRESSION_BORDER`.

Packets are grouped for the compression based on the `PACKET_PROCESSING_STARTED/PACKET_PROCESSING_FINISHED` and `PACKET_FREEZE_HINT/PACKET_THAW_HINT` packet pairs. If the first (freeze) packet is encountered the packets till the second (thaw) packet are put into a queue. This queue is then compressed and sent as a chunk packet. If the compression would expand in size the queued packets are sent uncompressed as “normal” packets.

The compression level can be controlled by the `FREECIV_COMPRESSION_LEVEL` environment variable.

6.8.5 Files

There are four file/filesets involved in the delta protocol:

1. The definition file: (`common/networking/packets.def`).
2. The packet generator file: (`common/generate_packets.py`).
3. The generated files: `*/*_gen.[cpp,h]` or as a list `client/civclient_gen.cpp`, `client/packhand_gen.h`, `common/packets_gen.cpp`, `common/packets_gen.h`, `server/hand_gen.h`, and `server/srv_main_gen.cpp`.
4. The overview (this document)

The definition file lists all valid packet types with their fields. The generator takes this as input and creates the generated files.

For adding and/or removing packets and/or fields you only have to touch the definition file. If you however plan to change the generated code (adding more statistics for example) you have to change the generator.

6.8.6 Changing the Definition File

Adding a packet:

1. Choose an unused packet number. The generator will make sure that you do not use the same number two times.
2. Choose a packet name. It should follow the naming style of the other packets: `PACKET_<group>_<remaining>`. The `<group>` may be `SERVER`, `CITY`, `UNIT`, `PLAYER`, and `DIPLOMACY`.
3. Decide if this packet goes from server to client or client to server.
4. Choose the field names and types.
5. Choose packet and field flags.
6. Write the entry into the corresponding section of `common/networking/packets.def`.

If you add a field which is a struct (say `foobar`), you have to write the following functions: `dio_get_foobar()`, `dio_put_foobar()`, and `are_foobars_equal()`.

Removing a packet:

1. Add a mandatory capability string.
2. Remove the entry from `common/networking/packets.def`.

Adding a field:

Option A:

1. Add a mandatory capability string.
2. Add a normal field line: `COORD x`.

Option B:

1. Add a non-mandatory capability string (i.e. “`new_version`”).
2. Add a normal field line containing this capability in an add-cap flag: `COORD x; add-cap(new_version)`

Removing a field:

Option A:

1. Add a mandatory capability string.

2. Remove the corresponding field line.

Option B:

1. Add a non-mandatory capability (i.e. “cleanup”)
2. Add to the corresponding field line a remove-cap flag

6.8.7 Capabilities and Variants

The generator has to generate code which supports different capabilities at runtime according to the specification given in the definitions with the `add-cap()` and `remove-cap()` functions. The generator will find the set of used capabilities for a given packet. Let us say there are two fields with `add-cap(cap1)` and one field with a `remove-cap(cap2)` flag. So the set of capabilities are `cap1` and `cap2`. At runtime the generated code may run under 4 different capabilities:

- Neither `cap1` nor `cap2` are set.
- `cap1` is set, but `cap2` is not.
- `cap1` is not set, but `cap2` is set.
- `cap1` and `cap2` are set.

Each of these combinations is called a variant. If n is the number of capabilities used by the packet the number of variants is 2^n .

For each of these variants a separate send and receive function will be generated. The variant for a packet and a connection is calculated once and then saved in the connection struct.

6.9 Format Description of the Scorelog

Empty lines and lines starting with `#` are comments. Each non-comment line starts with a command. The parameter are supplied on that line and are separated by a space. Strings which may contain whitespaces are always the last parameter and so extend until the end of line.

The following commands exists:

- `id <game-id> : <game-id>` is a string without whitespaces which is used to match a scorelog against a savegame.
- `tag <tag-id> <descr> :` Add a data-type (tag) the `<tag-id>` is used in the ‘data’ commands `<descr>` is a string without whitespaces which identified this tag.
- `turn <turn> <number> <descr> :` Adds information about the `<turn>` turn `<number>` can be for example year `<descr>` may contain whitespaces.
- `addplayer <turn> <player-id> <name> :` Adds a player starting at the given turn (inclusive). `<player-id>` is a number which can be reused `<name>` may contain whitespaces.
- `delplayer <turn> <player-id> :` Removes a player from the game. The player was active till the given turn (inclusive) `<player-id>` used by the creation.
- `data <turn> <tag-id> <player-id> <value> :` Gives the value of the given tag for the given player for the given turn.

GLOSSARY OF TERMS

This page contains a collection of terms and acronyms commonly used in the documentation as well as in varying community discussions.

AI

Artificial Intelligence

Freeciv21 comes with a pre-programmed artificial intelligence that you can play against in single-player games.

FC

Freeciv

Classic legacy [Freeciv](#).

FC21

Freeciv21

A new fork of Freeciv v3.0 concentrating on multiplayer games and a single game interface.

FCW

Freeciv Web.

A web client and highly customized version of the legacy Freeciv server. Not affiliated directly with Freeciv, Freeciv21, or Longturn.

FEP

For Each Player

During *The Turn Change Sequence*, activities occur for each player in a random sequence per turn.

FFA

Free For All

These are Longturn games where all the players start the game non-allied to any other player. During the game, players may form alliances. However, these alliances are often not formally reported in-game via the *Nations and Diplomacy View* until necessity requires it. This is typically when conflict occurs and allied units need to enter allied cities.

FOW

Fog of War

If you do not have a unit giving vision on a piece of the game map, your nation will not see border changes until you move a unit giving vision.

FP

Fire Power

The amount of damage an attacking unit can inflict on a defending unit within a single round of combat.

Goto

Units that are out of *MP* can be given advanced orders to take an action at the start of the next turn. There are many possibilities. See *Unit* → *Goto and ...* for some ideas.

HP

Hit Point(s)

The amount of health a unit has available. When this goes to zero the unit is killed.

LT

Longturn

LTT

Longturn Traditional

Longturn's standard *ruleset*.

LTX

Longturn Experimental

Longturn's experimental *ruleset*.

MP2

Multiplayer 2 Ruleset

There are many MP2 *rulesets* and they are numbered: MP2a, MP2b, MP2c, etc.

MP

Move Point(s)

The number of moves a unit has available.

RTS

Real Time Strategy

This is a type of game where gameplay occurs all the time. In the Longturn community, this term is also used when players are online at the same time and potentially competing against each other.

TC

Turn Change

A period of time when the server processes end of turn events in a specific *order*

UWT

Unit Wait Time

A period of time that must be exhausted before a unit can move between turns. For example, if the UWT for a game is set to 10 hours and a unit moves 1 hour prior to TC. Then the unit cannot move for another 9 hours until the UWT counter is completed for that unit.

INDEX

A

AI, 343

F

FC, 343

FC21, 343

FCW, 343

FEP, 343

FFA, 343

FOW, 343

FP, 343

G

Goto, 344

H

HP, 344

L

LT, 344

LTT, 344

LTX, 344

M

MP, 344

MP2, 344

R

RTS, 344

T

TC, 344

U

UWT, 344