# Contributing to Wesnoth

Wesnoth was built with the hard work of volunteers all over the world! Everyone is welcome to come and contribute. TODO: list the different types of contributions that people can make.

## Contacting Us

The best place to get in touch with the development team is on our [official Discord community server](https://discord.gg/battleforwesnoth) or on the [Wesnoth forums](https://forums.wesnoth.org/).

The Discord server is mirrored to freenode IRC, channels: [`#wesnoth`](https://webchat.freenode.net/#wesnoth) (general discussions), [`#wesnoth-umc-dev`](https://webchat.freenode.net/#wesnoth-umc-dev) (questions about creating add-ons), [`#wesnoth-dev`](https://webchat.freenode.net/#wesnoth-dev) (development of wesnoth mainline).

## Art and Music

Art and music submissions are accepted usually to fill in missing or outdated assets. If you are interested in contributing, we recommend that you contact us first to determine the best resources for you to work on based on need and interest. We also commission larger projects such as character portraits, story art, and music tracks.

## Engine

Wesnoth's engine conforms to the C++14 standard. We encourage the use of standard library APIs over third-party libraries whenever possible. However, third-party libraries are preferred over adding new, custom in-engine APIs, when appropriate.

### Code Formatting

All C++, WML and Lua files are in UTF-8, as we use Gettext-style translations, and translatable strings use some punctuation that's outside of the ASCII subset. More details are in the [Typography Style Guide](https://wiki.wesnoth.org/GettextForWesnothDevelopers).

If your pull request touches the engine's C++ source code, we recommend (but don't require) you run `clang-format` on your changes before submission (Visual Studio Code gives you a handy context menu option to do so). This ensures that your code remains formatted according to our conventions. Make a local commit before running `clang-format`, in case more code than expected gets changed.

Generally, we follow these conventions in our C++ code:

```cpp

// Use pragma once instead of an include guard. Those are clumsy.

#pragma once

// Includes for files from the src/... directories should use double-quotes.

#include "help/help.hpp"

#include "gettext.hpp"

// Use angle brackets for system and external includes.

// Includes should also be sorted alphabetically.

#include <algorithm>

#include <array>

#include <iostream>

#include <string>

#include <vector>

// Classes should have scope specifiers (public, protected, private), but structs can omit them.

struct my\_struct

{

// Public members do not need a trailing underscore.

// Inline initialization is acceptable over a constructor.

bool member = false;

};

// Class names are lower-case with underscores between words.

// Put braces on new lines after class and struct declarations.

class my\_class

{

public:

// Use using directives over typedefs. They're easier to read.

using alias\_t = std::vector<my\_struct>;

// Use leading commas in the ctor list

// Use the T& foo or T\* foo reference and pointer styles, not T &foo or T \*foo.

// Use the "explicit" keyword for single-argument constructors.

explicit my\_class(alias\_t& ref)

: the\_array\_of\_doom\_()

, vec\_ptr\_(nullptr) // Use nullptr instead of NULL or 0

{

// Use C++ casts (static\_cast and dynamic\_cast) instead of C-style casts.

// Do try and avoid reinterpret\_cast and const\_cast if at all possible.

const float cast\_test = static\_cast<float>(how\_far\_to\_destination\_);

// Don't put a space after conditional keywords, and keep their opening brackets on the same line.

if(!ref.empty()) {

vec\_ptr\_ = &ref;

// Use lambdas for short functions like this.

// We also encourage the use of auto in lambdas and other places where

// type names are long and can be inferred.

std::sort(ref.begin(), ref.end(), [](const auto& a, const auto& b) {

return a.member && !b.member;

});

}

}

/\*\*

\* Keep class method brackets on their own line, and always utilize const for methods and

\* variables when possible.

\*

\* For documenting functions, we loosely follow Doxygen conventions. You don't need to document

\* every single function, but important ones should optimally have at least a one-line comment

\* explaining what it does.

\*

\* This returns a translatable string, using gettext's \_ function.

\*

\* @param speaker\_id The person speaking

\*/

t\_string exclaim(const std::string& speaker\_id) const

{

if(how\_far\_to\_destination\_ < 100) {

if(speaker\_id == "signboard") {

return \_("Oldwood — enter at own risk");

} else {

// TRANSLATORS: The lake is the small underground one in S06 Temple in the Deep

return \_("Hmm, someone has written underneath “Fire-carrying trespassers will be thrown in the lake.”");

}

} else {

return \_("Clearwater — just keep following the river");

}

}

private:

// End private class members with an underscore. Additionally, use C++ standard

// like std::array as opposed to C equivalents (such as int[])

std::array<int, 8> the\_array\_of\_doom\_;

alias\_t\* vec\_ptr\_;

// Use static or constexpr for constants. Don't use macros.

static const int how\_far\_to\_destination\_ = 1000;

};

```

## Bug Reports

Please report any bugs here on GitHub (preferred) or on the forums.

### Bugs in User-Made Content

If you encounter an engine bug such as a crash, scripting error, etc., report it here. Otherwise, issues with user-made content should be reported to their respective creators on the forums. You can usually find a thread for the add-on in question in the [Scenario & Campaign Development](http://www.wesnoth.org/forum/viewforum.php?f=8), [Faction & Era Development](http://www.wesnoth.org/forum/viewforum.php?f=19) or [Multiplayer Development](http://www.wesnoth.org/forum/viewforum.php?f=15) sections.

### Feature Requests

We accept suggestions for campaign improvements, WML or Lua API changes, and other game enhancements here on GitHub. We encourage you to attempt to implement your idea yourself and submit a pull request containing relevant information to your feature.

### Information to Include

We have several issue templates to choose from when opening a bug report. Please choose the one that best fits the bug. You do not need to include everything (we don't need screenshots for a compiling issue, for example), but the more information you can provide, the better. We need at least enough information to replicate the bug before we can track down the root cause.

## Pull Requests

TODO: stuff about PRs.