How to contribute to OpenMW

=======================

Not sure what to do with all your free time? Pick out a task from here:

https://gitlab.com/OpenMW/openmw/issues

Currently, we are focused on completing the MW game experience and general polishing. Features out of this scope may be approved in some cases, but you should probably start a discussion first.

Note:

\* Tasks set to 'openmw-future' are usually out of the current scope of the project and can't be started yet.

\* Bugs that are not 'Confirmed' should be confirmed first.

\* Often, it's best to start a discussion about possible solutions before you jump into coding, especially for larger features.

Aside from coding, you can also help by triaging the issues list. Check for bugs that are 'Unconfirmed' and try to confirm them on your end, working out any details that may be necessary. Check for bugs that do not conform to [Bug reporting guidelines](https://wiki.openmw.org/index.php?title=Bug\_Reporting\_Guidelines) and improve them to do so!

There are various [Tools](https://wiki.openmw.org/index.php?title=Tools) to facilitate testing/development.

Pull Request Guidelines

=======================

To facilitate the review process, your pull request description should include the following, if applicable:

\* A link back to the bug report or forum discussion that prompted the change. Note: when linking bugs, use the syntax ```[Bug #xyz](https://gitlab.com/OpenMW/openmw/issues/#xyz)``` to create a clickable link. Writing only 'Bug #xyz' will unfortunately create a link to the Github pull request with that number instead.

\* Summary of the changes made

\* Reasoning / motivation behind the change

\* What testing you have carried out to verify the change

Furthermore, we advise to:

\* Avoid stuffing unrelated commits into one pull request. As a rule of thumb, each feature and each bugfix should go into a separate PR, unless they are closely related or dependent upon each other. Small pull requests are easier to review, and are less likely to require further changes before we can merge them. A "mega" pull request with lots of unrelated commits in it is likely to get held up in review for a long time.

\* Feel free to submit incomplete pull requests. Even if the work can not be merged yet, pull requests are a great place to collect early feedback. Just make sure to mark it as \*[Incomplete]\* or \*[Do not merge yet]\* in the title.

\* If you plan on contributing often, please read the [Developer Reference](https://wiki.openmw.org/index.php?title=Developer\_Reference) on our wiki, especially the [Policies and Standards](https://wiki.openmw.org/index.php?title=Policies\_and\_Standards).

\* Make sure each of your changes has a clear objective. Unnecessary changes may lead to merge conflicts, clutter the commit history and slow down review. Code formatting 'fixes' should be avoided, unless you were already changing that particular line anyway.

\* Reference the bug / feature ticket(s) in your commit message (e.g. 'Bug #123') to make it easier to keep track of what we changed for what reason. Our bugtracker will show those commits next to the ticket. If your commit message includes 'Fixes #123', that bug/feature will automatically be set to 'Closed' when your commit is merged.

\* When pulling changes from master, prefer rebase over merge. Consider using a merge if there are conflicts or for long-running PRs.

Guidelines for original engine "fixes"

=================================

From time to time you may be tempted to "fix" what you think was a "bug" in the original game engine.

Unfortunately, the definition of what is a "bug" is not so clear. Consider that your "bug" is actually a feature unless proven otherwise:

\* We have no way of knowing what the original developers really intended (short of asking them, good luck with that).

\* What may seem like an illogical mechanic can actually be part of an attempt to balance the game.

\* Many people will actually <i>like</i> these "bugs" because that is what they remember the game for.

\* Exploits may be part of the fun of an open-world game - they reward knowledge with power. There are too many of them to plug them all, anyway.

OpenMW, in its default configuration, is meant to be a faithful reimplementation of Morrowind, minus things like crash bugs, stability issues and severe design errors. However, we try to avoid touching anything that affects the core gameplay, the balancing of the game or introduces incompatibilities with existing mod content.

That said, we may sometimes evaluate such issues on an individual basis. Common exceptions to the above would be:

\* Issues so glaring that they would severely limit the capabilities of the engine in the future (for example, the scripting engine not being allowed to access objects in remote cells)

\* Bugs where the intent is very obvious, and that have little to no balancing impact (e.g. the bug were being tired made it easier to repair items, instead of harder)

\* Bugs that were fixed in an official patch for Morrowind

Feature additions policy

=====================

We get it, you have waited so long for feature XYZ to be available in Morrowind and now that OpenMW is here you can not wait to implement your ingenious idea and share it with the world.

Unfortunately, since maintaining features comes at a cost and our resources are limited, we have to be a little selective in what features we allow into the main repository. Generally:

\* Features should be as generic and non-redundant as possible.

\* Any feature that is also possible with modding should be done as a mod instead.

\* In the future, OpenMW plans to expand the scope of what is possible with modding, e.g. by moving certain game logic into editable scripts.

\* Currently, modders can edit OpenMW's GUI skins and layout XML files, although there are still a few missing hooks (e.g. scripting support) in order to make this into a powerful way of modding.

\* If a feature introduces new game UI strings, that reduces its chance of being accepted because we do not currently have any way of localizing these to the user's Morrowind installation language.

If you are in doubt of your feature being within our scope, it is probably best to start a forum discussion first. See the [settings documentation](https://openmw.readthedocs.io/en/stable/reference/modding/settings/index.html) and [Features list](https://wiki.openmw.org/index.php?title=Features) for some examples of features that were deemed acceptable.

Reviewing pull requests

=======================

We welcome any help in reviewing open PRs. You don't need to be a developer to comment on new features. We also encourage ["junior" developers to review senior's work](https://pagefault.blog/2018/04/08/why-junior-devs-should-review-seniors-commits/).

This review process is divided into two sections because complaining about code or style issues hardly makes sense until the functionality of the PR is deemed OK. Anyone can help with the Functionality Review while most parts of the Code Review require you to have programming experience.

In addition to the checklist below, make sure to check that the Pull Request Guidelines (first half of this document) were followed.

First review

============

1. Ask for missing information or clarifications. Compare against the project's design goals and roadmap.

2. Check if the automated tests are passing. If they are not, make the PR author aware of the issue and potentially link to the error line on Travis CI or Appveyor. If the error appears unrelated to the PR and/or the master branch is failing with the same error, our CI has broken and needs to be fixed independently of any open PRs. Raise this issue on the forums, bug tracker or with the relevant maintainer. The PR can be merged in this case as long as you've built it yourself to make sure it does build.

3. Make sure that the new code has been tested thoroughly, either by asking the author or, preferably, testing yourself. In a complex project like OpenMW, it is easy to make mistakes, typos, etc. Therefore, prefer testing all code changes, no matter how trivial they look. When you have tested a PR that no one has tested so far, post a comment letting us know.

4. On long running PRs, request the author to update its description with the current state or a checklist of things left to do.

Code Review

===========

1. Carefully review each line for issues the author may not have thought of, paying special attention to 'special' cases. Often, people build their code with a particular mindset and forget about other configurations or unexpected interactions.

2. If any changes are workarounds for an issue in an upstream library, make sure the issue was reported upstream so we can eventually drop the workaround when the issue is fixed and the new version of that library is a build dependency.

3. Make sure PRs do not turn into arguments about hardly related issues. If the PR author disagrees with an established part of the project (e.g. supported build environments), they should open a forum discussion or bug report and in the meantime adjust the PR to adhere to the established way, rather than leaving the PR hanging on a dispute.

4. Check if the code matches our style guidelines.

5. Check to make sure the commit history is clean. Squashing should be considered if the review process has made the commit history particularly long. Commits that don't build should be avoided because they are a nuisance for ```git bisect```.

Merging

=======

To be able to merge PRs, commit priviledges are required. If you do not have the priviledges, just ping someone that does have them with a short comment like "Looks good to me @user".

The person to merge the PR may either use github's Merge button or if using the command line, use the ```--no-ff``` flag (so a merge commit is created, just like with Github's merge button) and include the pull request number in the commit description.

Dealing with regressions

========================

The master branch should always be in a working state that is not worse than the previous release in any way. If a regression is found, the first and foremost priority should be to get the regression fixed quickly, either by reverting the change that caused it or finding a better solution. Please avoid leaving the project in the 'broken' state for an extensive period of time while proper solutions are found. If the solution takes more than a day or so then it is usually better to revert the offending change first and reapply it later when fixed.

Other resources

===============

[GitHub blog - how to write the perfect pull request](https://blog.github.com/2015-01-21-how-to-write-the-perfect-pull-request/)