# Contributing to lc0

These are the guidelines and standards followed by this codebase.

The language is C++, specifically C++17. As such, manual `new` and `delete` memory mangement is strongly discouraged; use the standard library tools for managing memory (such as `unique\_ptr`, `shared\_ptr` etc.).

This codebase uses semantic versioning. A release is the final commit for that version number, and all subsequent commits are development for the next version. `master` is the default branch, and the active development branch (as such, all Pull Requests go here); it always targets a minor (or major) version which succeeds the current relase. `release` is always equivalent to the latest tag.

### Style

Style is of course the first guideline on every new contributor's mind :)

This codebase largely complies with the [Google C++ style guide](https://google.github.io/styleguide/cppguide.html). The maintainers recommend the use of [Clang's auto formatter](https://clang.llvm.org/docs/ClangFormatStyleOptions.html).

Notable exceptions:

1. C++ exceptions are allowed (in fact, only `lczero::Exception`, defined in `utils/exception.h`, is allowed)

2. We use `#pragma once` instead of header guards.

3. Default function parameters are sometimes allowed.

4. Rvalue reference function params are sometimes allowed, not only for constructors and assignment operators.

For items (3) and (4), usage of those are discouraged, only use them if they benefit readability or have significant performance gain. It's possible that those exceptions (3) and (4) will be disallowed in future.

The most important rule to follow is consistency: look at the surrounding code when doing changes and follow similar style.

These are the most important parts of the codebase style (as a sort of tl;dr):

\* Comments must be full sentences, i.e. capitalized and ending in a period. (Sentences with elided subjects are fine.) Only `//` style comments are allowed, `/\* \*/` style comments aren't.

\* Braces are a variant of K&R style, as can be gleaned from existing code. All `if` statements must use braces, with the possible exception of single statement `if`s, which \*may\* omit if the braces \*if\* the conditional and following statement are on the same line. Again, see surrounding code for examples.

\* Indentation is two spaces; \t characters are disallowed.

\* Code line length is strictly capped at 80 characters.

\* Using non-`const` references as function parameters is disallowed; use pointers instead. (Using `const` references as parameters is fine.)

\* Identifier style:

- `kLikeThis` for constants and enum values

- `like\_this` for variables

- `like\_this\_` for member variables

- `LikeThis` for function and class names

\* All code should be inside `namespace lczero`

The internal code dependency structure looks like this:

\* Code in `src/utils` is not allowed to depend on any other code.

\* Code in `src/chess` only depends on `src/utils`

\* Code in `src/neural` only depends on `src/utils` and `src/chess`

\* Code in `src/mcts` only depends on `src/utils`, `src/chess` and `src/neural`

### Git history

Pull Requests are squashed when merged. This means all commits in the branch will be squashed into one commit applied onto master, so branches and their PRs should stick to \*one\* topic only. If you think changes deserve separate commits, make separate PRs for each commit.

This also means it's not possible to reuse one branch for multiple PRs; new PRs must either use entirely new branches, or else you could use `git reset --hard` on the current branch.

### Allowed features

Lc0 is still in early stages of development, and has not yet reached the point where we are ready to add small tweaks to add few points of a rating. Large code changes still happen, and having lots of small optimizations adds overhead to larger changes, slowing development.

Therefore, as a rule, search algorithm tweaks that give a gain of less than ~20 Elo points are discouraged at this point. (This limit will gradually be lowered as Lc0 code matures, eventually to 0.0 Elo).

#### Adding new command line flags/UCI parameters

Only add new parameters if users can significantly (>20 Elo) benefit by tweaking it. We don't want to make every single constant configurable (or rather, users don't want to see hundreds of parameters which don't do anything).

Try to minimize number of parameters that your feature introduces. If your feature introduces several parameters, every individual parameter should be significant (i.e. tweaking it with other fixes will give >20 Elo).

#### Adding features for testing

It is fine to temporarily commit a feature of unknown Elo gain so that people may test it. It's also fine to expose many parameters for the feature initially so that people can tune them. However, if the tweak doesn't prove to be significant, it should be removed after a few weeks.