Contributing to this list:

- Ideas page Manual
- Example

Introduction

Below is a list of project ideas for Google Summer of Code. These projects may require familiarity with C, Makefiles, Lua or VimL ("Vim script").

Neovim is a text editor based on Vim. One of the project goals is to encourage hacking and collaboration. Effort is put into removing barriers for contributors and improving documentation. The wiki has sections on developing and contributing to help you get started.

The Neovim source has roots going back to 1987 which means libraries such as libuv were not around at that time. The codebase can be made easier to maintain and understand by using these libraries. Project ideas that involve working heavily with internals will in general be more difficult than project ideas that "simply" add new features. However working with older/complex parts of the code base can also provide valuable learning feedback for writing simpler and more maintainable code. There is a large range of skills that can be learned and working with the team to find a project that will help you the most is in your benefit.

Visit gitter.im/neovim or `#neovim` on IRC to discuss these projects with the community and our mentors. Because communication is a big part of open source development you are expected to get in touch with us before making your application.

It's also recommended to start with a smaller coding task, to get introduced to the Neovim project. Look on the issue tracker for simpler issues and submit a small PR attempting to solve the issue. See for instance the good first issue and complexity:low tags. The goal here is to get accustomed with the codebase and to our workflow for reviewing and testing PRs, rather than directly making a large code contribution.

The following list of projects are just some ideas. We are happy to hear any suggestions that you may have.

Making a proposal

The application period for GSOC is March 16-March 31 (Timeline). Send your proposal through the official GSOC page. We encourage students to send a first draft early in this period, this allows us to give feedback and and ask for more information if need. See https://google.github.io/gsocguides/student/writing-a-proposal for some guidelines for writing a good proposal.

Note: this year we will likely be able to accept 2-3 students. We expect to get more strong proposals than available slots, so we will need to turn some good proposals down.

Proposal evaluation

Proposal evaluation criteria:

http://intermine.org/internships/guidance/grading-criteria-2019/

Tips

- Anywhere a Vim concept (such as "textlock") is mentioned, you can find what it means by using the ":help" command from within neovim (`:help textlock`).
- Ask questions and post your partial work frequently (say, once every day or 2, if possible). It's OK if work is messy; just put "[WIP]" in the pull request (PR) title.

- Take advantage of the continuous integration (CI) systems which automatically run against your pull requests. When you send work to a PR, the full test-suite runs on the PR while you continue to work locally.
- The wiki contains up-to-date documentation on building, debugging, and development tips.
- Only a text editor, CMake, and a compiler are needed to develop Neovim. Ctags is very helpful also.
- The contributing guidelines are intended to be helpful, not rigid.

Projects

- New Features
  - Improve Lua configurability
  - GUI Features
  - Live preview of commands
  - Improve autoread
  - IDE "Vim mode"
  - Propose an idea!

Support for Lua plugins and configuration

Desirable Skills:

- C and related tools
- Familiarity with Lua C API is a plus.

Description:

Nvim is always built with a builtin Lua interpreter. A goal is to make Lua a first class language for plugins and user config, which can access editor functionality directly, without "shelling out" to VimL commands and functions.

Expected Result:

More functionality is directly exposed to Lua. This project idea is a bit open ended, and improvements can be made in different directions. Regardless what priorities are decided, it is expected that the project will take form as multiple smaller deliverables that will be merged separately during the duration of the project.

The project can involve calls the core editor makes to VimL, such as autocommands and mappings, where native Lua support could be added. Lua code can already use luv bindings to libuv for async io, but better integration with specific io features like rpc channels and terminal buffers could be worthwhile. Adding API functions for vim features only available via ex commands would be useful (like reading and setting :highlight definitions). Nvim is working on building a Lua standard library to make it convenient to write plugins and user config in Lua. This goal is broad, and discussion with the plugin developing community is encouraged, about what functionality would be most useful for the standard library.

Another direction is to add new extension points, such as letting Lua code control aspects of screen drawing. A starting point could be to let Lua code draw a richer completion popup menu for the TUI, by processing popup menu UI events internally.

Difficulty: Medium

Code license: Apache 2.0

Mentor: Björn Linse (@bfredl), Ashkan Kiani (@norcalli)

GUI Features

Desirable Skills:

- C and related tools
- Familiar with event-loop programming model

Description:

Nvim GUIs are implemented as processes communicating with Nvim. Originally the UI protocol exposed most functionality as a single, terminal-like screen grid. Work has been done to allow UIs (including embeddings in GUI editors, like VSCode) to render the screen layout themselves, based on semantic updates from Nvim. Some screen elements like the cmdline, popupmenu and message area has been externalized. As a result of a 2018 GSOC project, windows can be drawn on separate grids.

Expected Result:

Further improvements to the GUI protocol.

Some UIs want to render the buffer contents themselves. A solution would be a UI protocol mode, where the rendered grid is not used, rather all decorations, such as syntax highlighting, conceals, virtual text are transmitted for the current viewport. Such an UI would be able to render text without the restrictions of the builtin monospace grid. Then the UI should be able to inform nvim about usage of virtual columns and wrapping, so that vim commands such as `gj` are consistent with how text is presented to the user.

Another path is to improve the core Nvim grid model. We could allow the width and height of cells be different for each row. This would allow for heading text with different font size, and pictures rendered inside windows.

Putting forward your own ideas of UI improvements is encouraged. Read the docs for the implemented extensions as well as the tracking issue for ongoing/planned work, as a starting point.

Difficulty: Medium-Hard

Code license: Apache 2.0

Mentor: Björn Linse (@bfredl)

Live preview of commands

Desirable Skills:

- C and related tools
- Familiar with event-loop programming model

Description:

Nvim has builtin live preview of `:%s` substitution, as a result of a [[successful collaboration with students|https://medium.com/@eric.burel/stop-using-open-source-5cb19baca44d]]. This support could be extended to more commands, such as `:global` and `:normal`.

Expected Result:

More commands support interactive preview. This could be done by extending the hard-coded support for substitution preview to more commands. Alternatively, a more scalable approach could be to add core functionality that makes it easier to implement live preview as plugins in VimL and/or Lua. See [[#7370|https://github.com/neovim/neovim/issues/7370]] for some ideas.

Difficulty: Medium-Hard.

Code license: Apache 2.0

Mentor: Björn Linse (@bfredl)

Improve autoread

Desirable Skills: Any

Description: Reload file/notify user when a file being edited changes outside of `nvim`.

Expected Result:

Neovim has the 'autoread' setting that regularly checks if a file edited in neovim has been externally modified. It thus notifies the user to prevent overwriting the changes. Sadly the current mechanism isn't foolproof. This project intends to make this feature work as well as in other editors like Sublime text and across the Neovim-supported platforms. The interface should also be improved so that notifications show how different the edited and modified files are. The candidate can realize some of the difficulties involved with this proposition for linux

Difficulty: Medium

Code license: Apache 2.0

Mentor: Matthieu Coudron (@teto)

IDE "Vim mode"

Desirable Skills: Any

Description: Implement "Vim mode" in an editor/IDE (such as IntelliJ) by embedding a `nvim` instance.

Expected Result:

Full Nvim editing should be available in the editor/IDE, while also allowing the user to use the native editor/IDE features.

Examples:

- VSCode integration
- Sublime Text integration

Difficulty: Medium

Code license: Apache 2.0

Mentor: TBD

Please add your project ideas in the following format.

Title

Desirable Skills:

Description:

Expected Result:

- Item1
- Item2

Difficulty: Easy/Medium/Hard

Code license: Apache 2.0

Mentor: Mentor name (@MentorName)