Neovim

CSCI 362 Project

Brandi Durham, Misae Evans, Lauren Tubbs, John Quinn

What is Neovim?

- Neovim is a project that seeks to aggressively refactor Vim in order to:
  - Simplify maintenance and encourage contributions
  - Split the work between multiple developers
  - Maximize extensibility

What's wrong with Vim?

- Over ~20 years of life Vim has gathered about 300k lines of code that is written in C89
- C89 is scary. Few people understand it and fewer have the guts to mess with it.
- One person is responsible for Vim's code base so he has to be extra careful before accepting patches because they become his responsibility once merged

Our Tests

- Neovim doesn’t seek to rewrite Vim from scratch or transform it into an IDE so the tests we made test both the functionality that one would expect from Vim but also those that test what is expected from Neovim. (Some Neovim commands are the same as Vim, some are different.)
- We used pyautogui to simulate keyboard input for the testing

Our Tests

1. Ability to write and save file (:wq command)
2. Ability to append to an existing file (:o command)
3. Ability to open a file, enter text and close without saving (:q! command)
4. Ability to delete a character (x command)
5. Ability to undo changes to the last line (u command)

Our Tests

6. Ability to move cursor to beginning of file (gg command)
7. Ability to move cursor to end of line ($ command)
8. Ability to move cursor to end of a word (e command)
9. Ability to move cursor to beginning of word (b command)
10. Ability to move cursor to end of file (G command)

Our Tests

11. Ability to find the first instance of a pattern in text (/command)
12. Ability to find the first instance of a pattern in text (2/ command)
13. Ability to find the last instance of a pattern in text (? command)
14. Ability to find the second to last instance of a pattern in text (2? command)
15. Ability to move through found instances of a pattern in text (n command)

Our Tests

16. Test another way to delete a character (X command)

17. Ability to delete an entire line (D command)

18. Ability to undo text and redo text (u command and ctrl-R command)

19. Ability to move up in neovim text and delete text (k command and D command)

20. Ability to move down in neovim text and delete text (j command and D command)

Our Tests

21. Test another way to move down and delete text (+ command and D command)

22. Test another way to move up and delete text (- command and D command)

23. Ability to replace a char with another char (r command)

24. Ability to access replace mode (R command)

25. Ability to repeat last function (. command)

Our Testing Framework

- Our framework is set up so that a tester simply has to add two files in the appropriate folders to add a new test: a text file which contains a description of the test and sequence of Neovim commands, and a text file containing the desired result of executing those Neovim commands.

```
1  Test ability to write and save a new file (:wq command).
2  nvim <space> test01-output.txt <enter> <i> Testing the insert function <enter> <esc> :wq <enter>
```

Neovim commands file

```
1  Testing the u function
```

Correct output file

- The tester doesn't need to know anything other than how Neovim commands work.

Our Testing Framework

- Once the tester has added all the tests they want, they run a single bash script, runAllTests.sh
- The bash script iterates through all the Neovim command files the tester added. For each one:
    - The bash script calls a Python script which parses the file and uses the module Pyautogui to simulate the desired keyboard input in Neovim.
    - The Python script generates an output file with the results of the commands.
    - The bash script compares the results of the commands with the correct file input by the tester using a “diff”.
    - If the files are the same, the bash script outputs “Test passed”.
- The bash script then generates an HTML report of the testing results along with the descriptions of the tests provided by the tester.

Our Testing Framework

```
OUTDIFF=diff -B -b $TEMPFILE $COMPAREFILE
echo $OUTDIFF
if [ -z "$OUTDIFF" ];
#diff -B -b $TEMPFILE $COMPAREFILE >/dev/null 2>&1
 then
    echo " passed "
    echo "$entry passed" >> $REPORTSFILE
    TOTALPASSED=$((TOTALPASSED+1))
 else
    echo $OUTDIFF
    echo " failed"
    echo "$entry failed" >> $REPORTSFILE
    TOTALFAILED=$((TOTALFAILED+1))
 fi
 echo
done
```

Bash script sample

```
#stores each word in the list words
words = content.split()
dir_path = os.path.dirname(os.path.realpath(__file__))

#iterates through the words list
for word in words:
    #find all the <'s which are commands
    if "<" in word:
        type =word[1:]
        type = type[:-1]
        pyautogui.press(type)
    elif ".txt" in word:
        pyautogui.typewrite(temp_file)
    #if not, then plain text
    else:
        pyautogui.typewrite(word + ' ')
```

Python script sample

Fault Injection

1. **Mess with Mouse Response**

```
if (button != 0 || (ev != TERMKEY_MOUSE_PRESS && ev != TERMKEY_MOUSE_DRAG && ev != TERMKEY_MOUSE_RELEASE)) {
        Return; }
```

changed '==0' to '!=0

No change

2 . **make the coordinates of where the cursor is go up instead of down, or the opposite of what it's supposed to do.**

```
row**++**; col--; // Termkey uses 1-based coordinates
```

changed -- to ++

No change

Fault Injection

3. **cause the buttons to choose the wrong direction.**

```
if (button == 2) {
len += (size_t)snprintf(buf + len, sizeof(buf) - len, "Left");
} else if (button == 1) {
len += (size_t)snprintf(buf + len, sizeof(buf) - len, "Middle");
} else if (button == 3) {
len += (size_t)snprintf(buf + len, sizeof(buf) - len, "Right");
}
```

Switched 1 and 2

No change

Fault Injection

4. startup will be longer by 1ms.

```
for (size_t ms = 0; ms <= 100 && !tui_is_stopped(ui);) {

ms += (loop_poll_events(&tui_loop, 20) ? 20 : 1);}
```

Changed < to <=

No change

5. Switched the HL_ITALIC and HL_BOLD

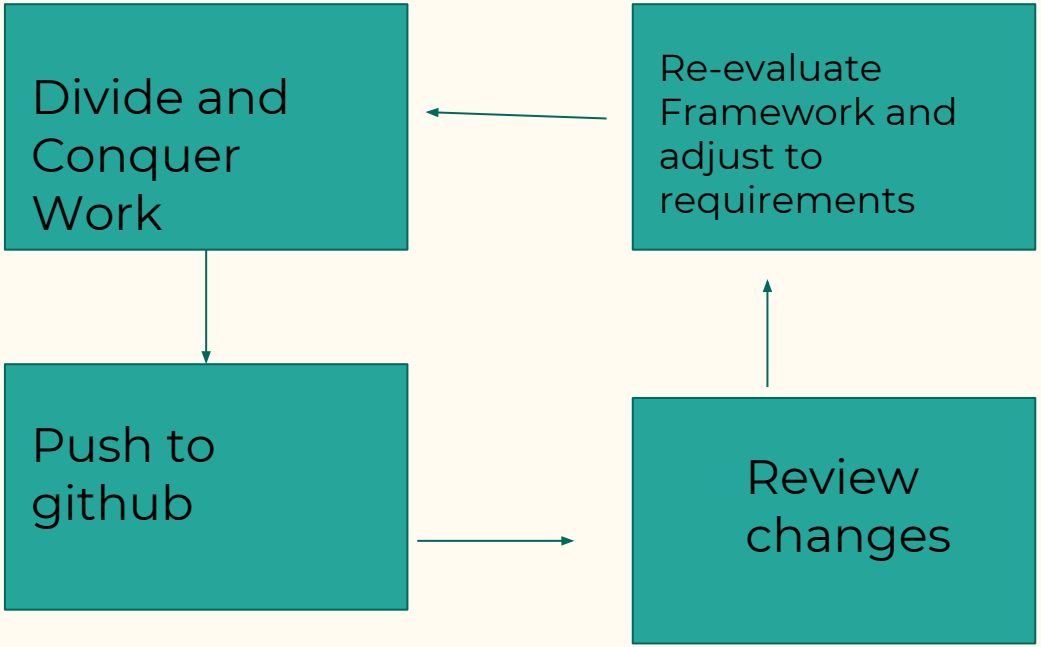
```
int attr = ui->rgb ? attrs.rgb_ae_attr : attrs.cterm_ae_attr;

bool bold = attr & HL_ITALIC;

bool italic = attr & HL_BOLD;
```

No change

Workflow


Divide and Conquer Work
Re-evaluate Framework and adjust to requirements
Push to github
Review changes