Devlin Ihmacs Code Review Materials

Architecture Review

During the architecture review, Steve suggested the following changes.

First: Make the current keychord part of the global state. I have done this. It has been helpful for echoing as you type keychords, as well as passing the typed keychord to other editing functions.

Second: Storing not the active buffer as a pointer to the buffer in a list, but rather storing it as an integer representing the index of the active buffer in a list. This has made it easier to ensure that everything is editing the correct buffer.

Architecture Changes

The amount of information that was required to be passed through to editing functions started spiraling out of control. Luckily, I made the choice to wrap the global state, as well as the controller and view, into one monolithic class. Now, I just pass the entire global state to editing functions.

This works out fine, except it does lead to the temptation of trying to dodge around the MVC architecture. I have been trying my best to make sure the editing functions manipulate the model, and do not tamper with the view. While I have forgotten at some points, I am pretty sure I cleared that all up.

In addition, many of the classes now have many more attributes storing aspects of the state I did not originally think I would need. Some of what I originally had as attributes I replaced with properties. These were things that I realized I would have to update the state of all the time, yet the state was entirely dependent on a static relationship with other attributes. For example, the modeline I originally wanted to store as a string, but now I generate it with a property.

Implementation Tasks

Additional editing commands to create a better Emacs experience (this is easy, and I already have a decent amount implemented).

File IO. The buffer class has methods for this that have a half baked implementation. They are not robust at all, only working with absolute Unix paths.

Minibuffer input. This will be required before I can implement many other features like switching buffers, renaming buffers, etc. This needs to be part of the controller. (Also a minibuffer class might be useful).

Buffer switching. Right now, I have just a single scratch buffer that is stored in the buffer list. In addition, I need a way to run editing commands on the non-active buffer (this would require a major rewrite of the editing functions I've implemented thus far).

Major Hurdles

ncurses input

Decoding keystrokes to a textual representation that I could use in my keymap dictionary was difficult. I originally wanted a no compromises solution that could decode what key was pressed, and what modifiers it was pressed with. I learned—after being shocked by how many characters it had to read to handle keystrokes involving the F keys—that I would have to settle for the ncurses keypad option, which automatically does this for you. There are some compromises that are made with this though. `C-j` is the same as `RET`, `C-i` and `TAB`, etc.

ncurses display

The view section for my architecture been a great source of difficulty. Trying to effectively tie my model to ncurses to create output has been difficult.

My current solution is full of a lot more magic numbers than I would like unfortunately.

One solution would be to have more constants in the global state which control these. Another solution, and probably better, would be to use multiple ncurses windows instead of 1. For example, the absolute hellish debugging experience that came with displaying only a certain portion of the buffer would easily be resolved by creating an curses pad window.

In addition, resizing the terminal right now is a great way to get a crash.

Not knowing exactly when to include things in different files.

This is a huge annoyance.

The first issue is the Emacs-like architecture. For example, the default global map. In the Ihmacs class file where this map is defined, I run `from basic_editing import *`. Is there a way to just import functions? I don't need to import regex or curses again, (more on that in a bit).

Importing curses multiple times has been an issue. Curses creates some global variables called in `curses.COLS` and `curses.LINES` to represent the size of the terminal. I have needed this in the view and the controller and in some editing functions, so I have imported curses several times. However, pylint gets annoyed at this (especially since curses.COLS and curses.LINES are only available after initializing the screen). Should I store these in the global state and pass it through?

Why does Pylint not like empty dicts?

Just wondering, it says they are unsafe. Why?

Debugging Plan

As of right now, I have not written any unit tests. A lot of my code is unit testable, and a lot of it isn't. But unit tests could help with a lot of the testable stuff.

As for the issues I discussed above, I hope to be able to talk to CAs, profs, other students, anyone who might know where I should start looking to resolve some of these problems.

Testing Approach

I do not have any unit tests as of now.

Readability

Overall, one of the problems I have is inconsistent local variable names for the same thing. This is across all files pretty much. There is also a lot of repeated code like store state of this, modify, restore original state. These could be abstracted into a with statement or something.

One question I have is about property docstrings. As of right now, I am typically just using a 1 line docstring for properties that describes the return value.

basic_editing.py

These are all the editing functions I've implemented thus far. I think they are reasonably understandable code with docstrings. All of these functions take the same first argument, and additional arguments are described in the docstring.

The biggest issue with this is it assumes some familiarity with using Emacs. It also has a problem where it assumes "active buffer" every time buffer is used, something which needs to be changed both in the architecture and also in the documentation.

buff.py

The buffer class. I think it's documented quite well. The difficulty in reading just comes down to how many properties and methods I have. There are some methods which I could also make private that are not.

controller.py

The controller class. With the exception of the `read_key` method (which has tons of comments explaining exactly why everything is the way it is, much of it is unnecessary rant), this is a good file.

fundamental_mode.py

The fundamental major mode for Ihmacs. This is the only major mode implemented as of now. Other modes will inherit from it. I think it's implemented in a reasonable way, although I am not sure if I should make some of these attributes public and ditch the superfluous properties. (For example, a user might want to modify the modemap for a given mode).

ihmacs.py

The main file to execute for the program. This literally exists for the sole purpose of using the `curses.wrapper` function (which allows for safe crashes that don't mess up terminal state).

ihmacs_class.py

This contains the top level Ihmacs class which holds the entire state of the editor. I think the comments and docstrings do a good enough job.

Performance

The editor feels fine, although I can see it quickly being even slower than real the canonical GNU/Emacs (which has a lot of performance issues).

Right now, every action (except for the very brief moment during the keystroke reading process), every call is blocking.

If no keystroke has been entered yet, keep on waiting for one. Once it gets one, it needs to finish executing everything before it can take another.

The process of doing that is also inefficient. Every command requires the entire buffer to be redrawn, the modeline to be redrawn, etc.

As of right now however, I've only been working with small buffers and it hasn't been an issue at all.

ncurses_test.py

This is a file I used to debug some ncurses stuff regarding input. It's still useful to find out what keycode is being returned.

view.py

This is a bit of a mess right now. Magic numbers galore. The `_redraw_buffer` method could be split up a little bit. It's too long and there are parts of it that would be useful in other locations.