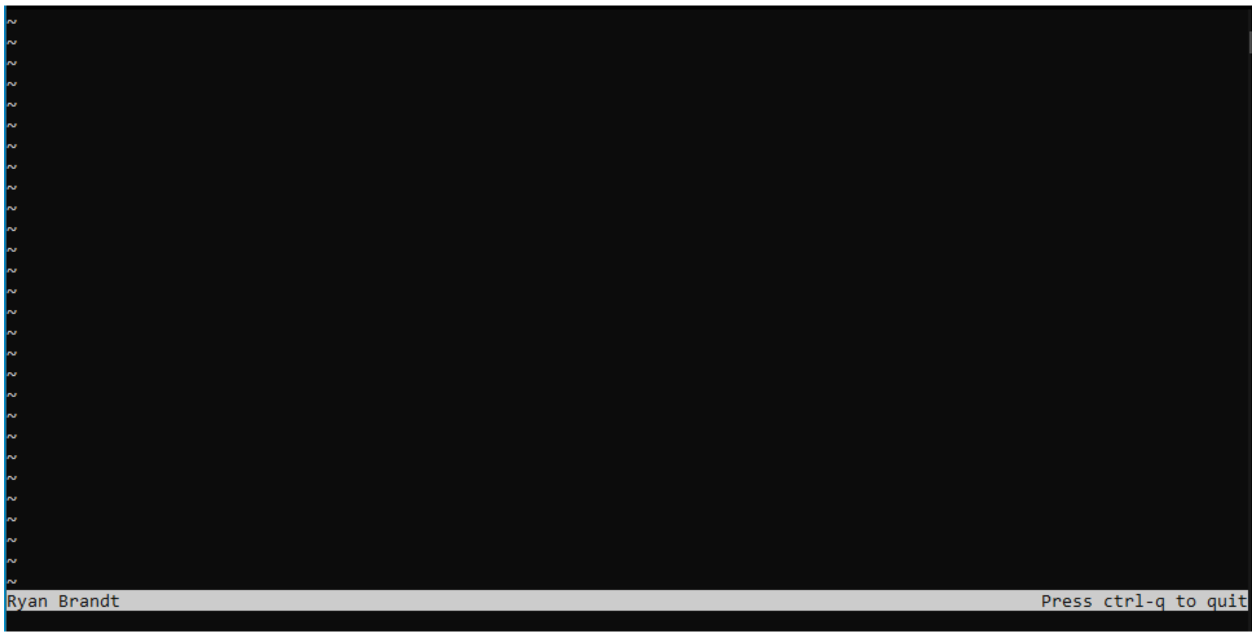
Background

The text editor was tested with an Ubuntu Linux distribution and uses a forever loop to run until the user decides to quit out of the program. An observer is attached to the view in order to monitor and make changes to the screen based on the user's input. Every time a key is pressed, a change can be made via an update function, and a refresh to the screen is called. The update function determines what change to make based on what key was pressed. Each character is stored in a vector of vectors of characters and is appended together to add each line, as a string, to the screen. The cursor gets initiated at the top left of the screen and will move around as changes are made to the text.

Initiation

The console-based text editor can be initiated by navigating to the directory of the stored files though a Linux command line. Once at this location, in the command line simply type make to initiate the compilation of the text editor. To run the editor type ./myeditor <filename>, where <filename> is the name of the desired input/output file with the .txt extension. If a file under the name is found, the contents of the file will be initiated in the editor, else a blank editor will be created. In the constructor if the editor, a view is created and the observer is attached to the view. The screen is now shown so that the user can begin editing the text. Upon quitting the program, the contents of the editor will be written to the desired file name.

Screen Layout



```
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
Ryan Brandt                                                                                     Press ctrl-q to quit
```

Each empty line is represented by a ~. When the user types in a line, the line then populates with what the user has typed. The screen will be initialized as shown above. Issuing a cursor movement command, via the arrow keys, will retrieve the contents of the input file, if there are any. The line at the bottom of the screen, with a white background, is a status bar that displays how to exit the program.

Keystrokes

Alpha numeric – inserts the pressed key at the location of the cursor

Special characters – inserts the pressed key at the location of the cursor

Spacebar – inserts a space at the location of the cursor

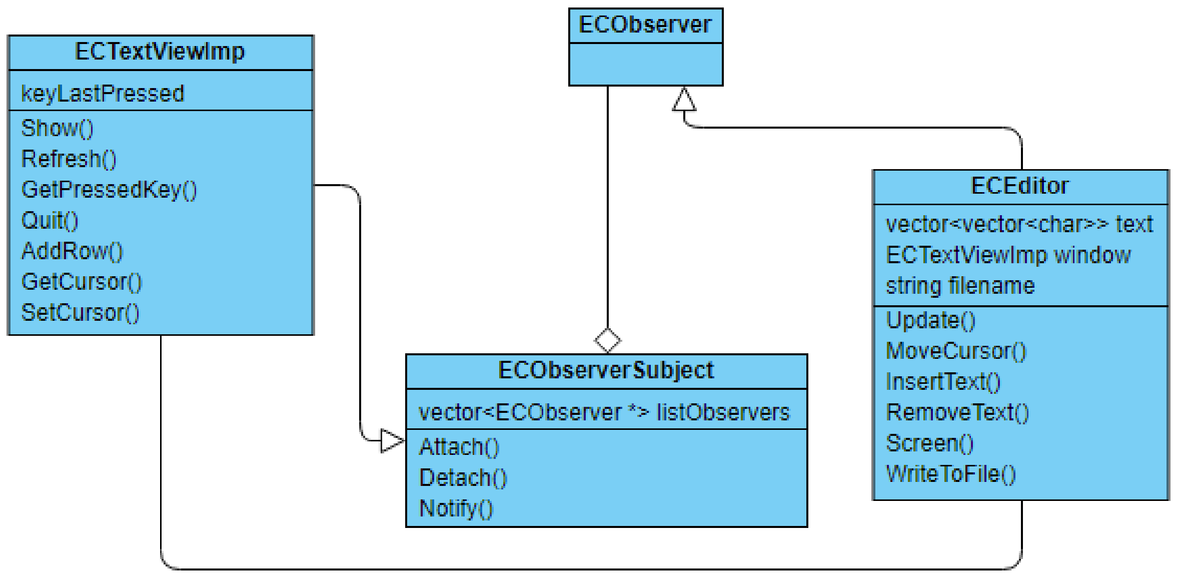
Arrow keys – moves the cursor around the bounded part of the screen by one unit per keystroke

Backspace – deletes the character to the left of the cursor location, or if at the first position, will merge with the row above

Enter – breaks the current line at the location of the cursor, or if at the last position, will create an empty row after the current row

CTRL-Q – quits the program and saves the text to a file

Design Spec



ECTextViewImp is a subclass of ECObserverSubject

ECEditor is a subclass of ECObserver

ECObserver and ECObserverSubject are aggregated, but can both exist separately

ECEditor and ECTextViewImp are in association with one another

Implementation Spec

Feature	Status	Design Pattern	User Manual
Text Entry	Implemented	Observer	Keyboard entry for all letters and symbols are supported.
Text Removal	Implemented	Observer	Backspace
Undo	Bugs/ Not Implemented	Observer	Was not fully functional, so was removed.
Redo	Bugs/ Not Implemented	Observer	Was not fully functional, so was removed.
Spaces	Implemented	Observer	Spacebar
Enter	Implemented	Observer	Enter
Arrow Keys	Implemented	Observer	Arrow Keys

Technical Issues – The window has an off by one error, due to one row not being shown on the screen, that had to be accounted for by increasing certain values by one. This issue also caused the existing file contents to not be inserted into the editor correctly, resulting in blank spaces shown on screen. The issue now is fully accounted for and does not negatively impact what is to be shown on screen. Undo and redo presented issues such as not being able to be continuously called on the same actions and were not working properly for pressing enter. Thus, both had to be removed for further production and testing.

Algorithms – The key reading algorithm listens for a key to be pressed and then calls the appropriate function to produce the desired action. The file input algorithm opens an input file stream, reads each line of the file, and populates the text field of the editor. The file output algorithm opens an output file stream, reads each line in the text field of the editor, and populates the text file with each line.

Refactoring – Originally, the update method contained all of the code for producing all of the desired actions. Upon refactoring of the code, however, each were moved to separate functions to reduce the size of the update function and to make it more modular. This improved testing of each of the features, as each function was now modular and could be removed easier to test other parts. Currently, the update method only contains the key pressed switch with each resulting function, which delegates the action to be performed to the function.

Conclusions

Learned – One of the main things I learned throughout this project was to plan more ahead of time. Doing this would have reduced the clutter of my code earlier, which would have reduced the amount of time that this project took, which was substantial. Adding comments to each section aided the readability of the code so that returning to certain aspects, after time away, was easier. With everything being very non-modular at the beginning, testing was quite difficult at times. Thus, including more object orientated design patterns could have eased this process of coding and testing each of the functionality.

Improvements – Without a doubt an improvement to my project would be to utilize more and different design patterns. These could include a composite pattern for the document structure, command pattern for undo/redo, or a chain of responsibility pattern for the key presses. Using more design patterns would also help the readability of the code so that others working on or improving the code would be able to do so more efficiently. Additional functionalities such as cut/copy/paste could be added to help the user move large sections of text around, further improving the ease of use of the program.