Neil Hemnani
April 27, 2020
CSE 3150
Yufeng Wu

Project Milestone 2: Adding More Features to the Text Editor

User Manual:

1. Navigate to the project directory.
2. Type 'make' in your terminal to compile the files.
3. Type "./ECTextEditor filename" to run the text editor
    a. You can use an existing .txt file within the directory.
    b. If the given filename does not exist, the editor will create a new file with that name within the directory.

Functionalities:

Ctrl-q	- save and quit the program
Ctrl-s	- save
Arrow keys	- move the cursor around to navigate the text
Enter/return	- add a new line
Backspace	- delete text (or go to the previous line)
Ctrl-z	- undo
Ctrl-y	- redo
Letter keys	- type in text

Report:

For this milestone, I added on the strategy pattern to provide line breaking whenever text is added or removed. Although I didn't use the Composition pattern in the sense that there was no hierarchical structure of Document -> Paragraphs -> Rows -> Words, I did create a class called Visible to encapsulate the line breaking algorithm and to separate the model in Editor from the view. Whenever an action is called in the

Editor, it calls a command on the document controller, ECTextDocumentCtrl. The document controller then uses the command pattern to perform an action on the text (such as inserting a character). After the action is completed, the Editor calls on Visible to perform linebreaking and then update the view.

Note: for now I have basic character insertion and backspace working with the new line breaking functionality. However, I haven't yet implemented a backspace at the beginning of a row, or an enter button. Although I had these in Milestone One, I am still struggling to get them working with the new line breaking. I anticipate having this done very soon, but for now the code may not work reliably. I am close, and I think I will finish it in a day or two (I have been struggling with the logic for these commands).

I also implemented file i/o for this milestone. If a file is provided, the Editor constructor simply uses the getline() function provided in C++ to read the .txt file line by line and populate its own std::vector<std::string> used to hold the data. If the file does not exist, the constructor will simply create a new file with the filename within the directory. When the program is quit, or the save command is used, the program loops through each string in the vector and overwrites the data to the filename.

Finally, I used this milestone to clean up my code a little bit; I moved over all the individual commands to Command.cpp to prevent my Editor.cpp file from being so massive and hard to read. Once I get all the functionality working, I will go back and simplify my code more to make it easier to read, as well as less bloated.

From Milestone One:

For this milestone, I used an MVC, observer, commands, and chain of responsibility. In Command.cpp/h, I implemented the command functionalities. Main.cpp is used to run the code. And Editor.cpp/h is where I wrote the bulk of the code, including the observer (or model), and controller.

The main observer is called Editor, and it is attached to an instance of ECTextViewImp. The document controller is called ECTextDocumentCtrl, and it is used to control the Editor. Whenever Update() is called on Editor, Editor calls one of its handler functions depending on what key was pressed. Then, these handler functions call on any one of the functions in ECTextDocumentCtrl, which use the command pattern to allow for undo/redo, to do text insertion, deletion, enter, or backspace.