Breakdown

Task: Learn vim
Date: November 12
Assignee: Han and Derek

Task: Setup (Makefile, Git, GitKraken, VS Code, .h files)
Date: November 17
Assignee: Han and Derek

Major Task: Implement functionality for *Text* class (adding and deleting characters). Since *Text* depends on many other classes such as *EditorComponent* and *KeyBoard,* parts of these classes should be implemented too.

Subtask: Implement necessary code for *View* and *Model* for *Text* class to work
Date: November 20
Assignee: Derek
Subtask: Implement necessary code for *VM, EditorComponent,* and *Controller* for *Text* class to work
Date: November 20
Assignee: Han

Major Task: Implement functionality for *Cursor* (displaying and moving cursor) and *StatusLine* (display information about file and cursor/window position) class.

Subtask: Implement *Cursor* functionality
Date: November 20
Assignee: Derek
Subtask: Implement *StatusLine* class to work
Date: November 20
Assignee: Han

Task: Implement commandline view (displays after hitting ‘esc’)
Date: November 21
Assignee: Han

Task: Test and clean up code.
Date: November 22
Assignee: Han and Derek

Major Task: Implement all required VM commands.

Subtask: Implement “motion commands” ($, %, ^, fF, j, h, k, l, ;, nN, b, /?, 0 (zero))
Date: November 24
Assignee: Derek
Subtask: Implement “command” commands (rR, u, iI, oO, pP, aA, sS, J, xX, .)
Date: November 24
Assignee: Han

Subtask: Implement “operator” commands (y, yy, d, dd, c, cc)
Date: November 25
Assignee: Derek

Subtask: Implement shortcuts commands (ctrl b, ctrl d, ctrl f, ctrl g, ctrl u)
Date: November 25
Assignee: Han

Task: Test and clean up code.
Date: November 26
Assignee: Han and Derek

Task: Implement macros commands (@)
Date: November 27
Assignee: Han

Task: Implement macros commands (q)
Date: November 27
Assignee: Derek

Task: Add syntax highlighting.
Date: November 29
Assignee: Han

Task: One other feature (to be determined).
Date: November 29
Assignee: Derek

Task: Create documentation for project. Also fix bugs if necessary.

Date: November 30
Assignee: Han and Derek

Design Doc Questions

Question: Although this project does not require you to support having more than one file open at once, and to be able to switch back and forth between them, what would it take to support this feature? If you had to support it, how would this requirement impact your design?

When *./vm filename0 filename1 filename2* is called (or however many filenames), we will store the filename arguments as strings in a *vector*. *Filename0* is loaded first, as usual. When the user inputs *:n* or *:N* into the commandline to switch files, if there are unsaved changes, we warn the user about the unsaved changes and prevent them leaving. We can keep track of changes with a boolean in *Control*. Otherwise, we check if the filename is the last element in the *vector* if the command is *:n* or if the filename is the first element in the *vector* if the command is *:N*. If either of those is true, we warn the user that they are trying to switch out of the first or last file.

This requirement would not affect our design as our *Model* class already handles loading files. This is useful because once the command is inputted through the *Controller* class which will let *Model* know, we can load the next or previous file on-the-fly using the vector of filename arguments.

Question: If a document's write permission bit is not set, a program cannot modify it. If you open a read-only file in vim, we could imagine two options: either edits to the file are not allowed, or they are allowed (with a warning), but saves are not allowed unless you save with a new filename. What would it take to support either or both of these behaviours?

Edits are not allowed completely: We can pass the write permission bit as a parameter of *Model*'s constructor and use a boolean variable to indicate the behaviour. If the document is read-only, we will remove all commands that allow the user to edit the file from our list of available commands ('a', 'i', and 'R' to name a few). With this, the user will not be able to make changes to the file. If the user tries to use any of these commands, we will display a warning through *StatusLine*.

Only if new filename:

If the file does not exist, we will proceed normally. Otherwise, proceed normally until the user attempts to save the file, in which case we try to check the read-only boolean and if the filename is the same as the filename we loaded. If the name is different, this is valid so we save the text into a new file under that supplied name. Otherwise, we display a warning to the user using the *StatusLine* class to let them know they cannot overwrite this read-only file.