TP3 Updates:
Major design updates include:

- adding image underlay options
- save options (the gallery)
- maze generation + visualization.

Minor design updates include:

- Start/End node selection
- Labeled queue
- Color balance
- Adjusted random weight generation to make A* heuristic admissible
- Backgrounds for weights in Dijkstra's and A*
- App no longer crashes for disconnected/nonexistent custom graphs

TP2 Updates:

Rather than having users assign weights when creating custom graphs, I changed it so that weights would be a randomly assigned integer between 1 and 9. I also decided to forgo the idea of dropdown menus for selecting different algorithms/graphs and went with buttons instead. Algorithmically, my project remains on pretty much the same track as it was for TP1.

Project Description:

The name of my project is 'Pathfinding Algorithm Visualizer" (although I aim to think of something more creative in the next few weeks). My project will allow users to visualize at least three pathfinding algorithms (Dijkstra, BFS, A*) step by step. The users should also be able to design their own custom graphs and watch the pathfinding algorithms traverse those as well.

Competitive Analysis:

There are quite a few graph and pathfinding visualizers out there. Most notably, Clement Mihailescu's project (https://clementmihailescu.github.io/Pathfinding-Visualizer/#) and the CS Academy's graph editor tool (https://csacademy.com/app/graph_editor/). Mihailescu's project allows the user to visualize how different pathfinding algorithms traverse a grid given a source and target. It also allows the user to add features like walls, 'bombs' that must be visited before reaching the target, and weighted paths. The CS Academy's graph editor does not have any functions for visualizing pathfinding algorithms, however, it allows users to add and delete nodes, and is extremely aesthetically pleasing.

Functionally, my project will be quite similar to Mihailescu's, however, I will allow my users to step forwards and backward through the algorithm instead of automatically running it. Additionally, I plan on incorporating a 'queue' feature that allows the user to look at what nodes will be looked at next. I also plan on allowing users to add nodes and edges in a similar fashion to the CS Academy's graph visualizer, however, the custom graphs in my project will not operate in 'Force mode' and will be constrained by a grid for ease of computation.

Structural Plan:

My project will have 3 main structural components: graph-related classes, algorithms, and visualizations. The graph-related classes will just be files containing my Graph, Node (and

possibly Path) class. My algorithm files will each contain one of the pathfinding algorithms I plan on visualizing (i.e. one file for Dijkstra's one file for BFS etc...) and my visualization files will also each contain one visualization for each algorithm. All the visualizations will be imported into one main visualizer file which will be capable of switching between the different implementations depending on how the user interacts with the project.

Algorithmic Plan:

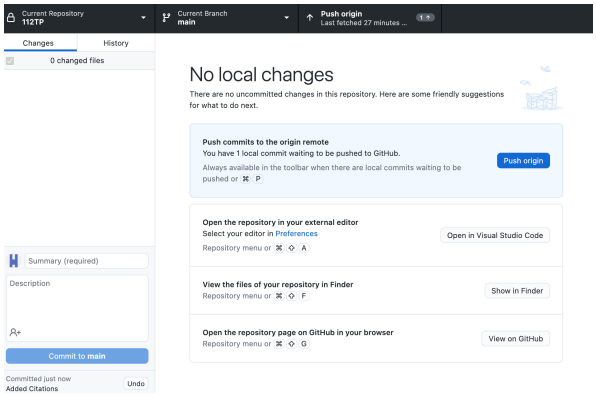
The most complex parts of my project are definitely the pathfinding algorithms (Dijkstra's, BFS, and maybe A*) themselves as well as the process of being able to 'step' through the algorithm. I will be making my task easier by doing research and fully understanding each of the algorithms before attempting to implement them. I also plan on using the pseudocode found on the Wikipedia pages for each algorithm to help structure my functions more cleanly (although those are mostly non-recursive, and I plan on implementing my functions recursively so I might need to re-haul the structure a little.) As for the process of storing and visualizing the data, since I will be writing my functions recursively, I plan on running the algorithm first, 'caching' the state of each recursive call within a list, then passing that list to the visualizer. This will allow me to easily implement the forwards/backwards step function and will prevent my project from re-calling the pathfinding algorithms unnecessarily.

Timeline Plan:

I hope to have all the algorithms fully written by tp1 and all the basic visualization features (hardcoded graphs, custom graph-making ability, stepping through the cached states, automatically running the algorithm) finished by tp2. After that, I would like to polish up the user interface by adding dynamic resizing (over Thanksgiving break) and details like sound effects and 'finishing' animations.

Version Control Plan:

I am currently backing my project up by periodically committing my edits to a GitHub repository:



Module List:

None