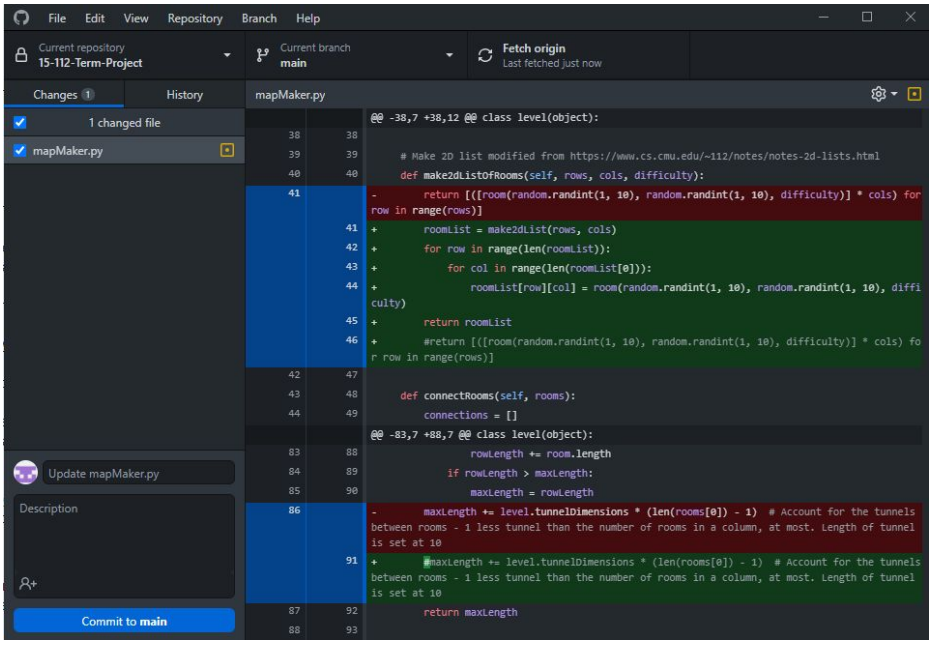
Adhvik Kanagala
15-112 Term Project Proposal

- Project Description:
    - Name (Working Title): Primrunner
    - This project is a maze crawler designed to be replayed infinitely. The player begins in a maze, fighting enemies and acquiring new powerful weapons and powerups to help them get to the end of the maze. Once the player has navigated through the maze and eliminated all enemies, they are teleported to a new level, keeping the gear they have acquired. The maze is new and larger, while the enemies have more health and are harder to fight. The goal is to get through as many rooms as possible. This game utilizes Prim's algorithm to generate mazes on demand, which are then populated with objects (weapons, enemies, etc).
- Competitive Analysis:
    - This game is inspired by an amalgamation of multiple games. In its overall structure, it is most similar to the Nintendo game The Legend of Zelda: Breath of the Wild, which contains an activity titled The Master Trials. The player begins with no tools or weapons, and is tasked with fighting through 45 rooms, each with tougher enemies and more gear. However, every single element of the map is manually created.

      Map generation in my game happens on the fly, rather than being manually premade. The algorithm used to create a new map is capable of scaling according to difficulty. Map generation is not unique to my game, and has been implemented in other games. A similar example is Destiny 2's Infinite Forest, which is designed to generate pieces of a map as players fight their way through. However, the pieces of these maps are premade, and contain locations where they are fitted together. This ensures that there is consistency in the types of obstacles players encounter, while still having a slightly different experience in each run. In my game, however, the only consistent element about each room is that it is rectangular. Everything else can change.
- Structural Plan:
    - This project makes heavy use of Object-oriented programming, with all entities as well as the level itself being stored as unique classes. The map and its associated methods and attributes are stored in one file, while the entity data (player, enemies, loot) are stored as their own classes in a separate file. Examples of loot are weapons and powerups. The bulk of the actual behavior of the game will be placed in its own file, while the graphics elements may go into a fourth file. I haven't decided yet if I need to use images, but those will be placed in a folder as necessary.
- Algorithmic Plan:
    - The most complex part of my project will likely be the map generation. Although my map is 2D and designed to be viewed from above, it is organized into rooms connected by tunnels as necessary. However, not all rooms are connected to each other. My map generation functions are capable of generating a 2D list of rooms (each with different

sizes), and then connecting them at random to create a path from the beginning to the end. The second step of this algorithm is to ensure that there is a viable path from the beginning of the map to the end, so that the player can solve it. The tunnel generation algorithm utilizes recursion to ensure that the player is not given an unsolvable map.
    - Another algorithmically complex element of this project is mapping all objects to a grid. Using a grid allows the game to have detailed knowledge of where all objects are, and allows for better abstraction of simple game actions like moving around.
        - After MVP, this grid algorithm would help me implement Dijkstra's or the A* algorithm for the enemy AI - for now, I expect that AI to be relatively simple
    - A potentially complex element of this project is rendering - rather than show the whole map at once, I plan to only show a small portion at a time. Figuring out how to do this could be a complex task.
- Timeline Plan:
    - Map generation is largely complete, and the next step for now is to map all objects to a grid.
    - Once I am able to consistently store all of my data in a grid format, I plan to focus on rendering an arbitrary level.
    - Next will be writing the game code itself - granting control of the player, simple enemy AI, interactions with loot, etc
    - The final step for MVP will be to update the rendering tool so that it can render notifications as well as focus on a specific section of the map rather than render the entire map at once.
- Version Control Plan:
    - I am utilizing Github to back up my code. I commit (and push) changes whenever I finish writing a function or class, and whenever I decide to stop working for the night.
    - 

- Module List:

- I don’t plan to use any modules that we have not already used in class. For now, I am using math, random, and CMU 112 Graphics.

TP2 Design Updates:

- Updated level generator
    - Now uses Prim's Algorithm to generate a maze, rather than a grid of randomly connected rooms
- Goal for each level is no longer to get to the opposite corner
    - Goal is to kill all enemies in the maze, in order to get restarted at the next level

TP3 Design Updates:

- Added enemy AI
    - Enemies now move towards the player
    - Enemies can fire their own weapons at the player
    - Enemy damage and health scales with difficulty
- Added graphics
    - Player, enemies, and health powerups are now represented by images
    - Weapons are drawn with TKinter
- Added pause menu and gear management system
- Added score counter