The generation is based on creating an instance of the Grid class, which sections the game window into cells. These cells are then assigned Tiles, which are basically images with some additional information such as what kind of tile it is. The constructor of the Grid is fed information about how many rooms I want to generate. The grid then attempts to generate this amount of rooms but breaks this loop if it can't find the space to generate more rooms. This is all tweakable in the code but it is currently set to a 60x60 grid, attempting to generate 9 rooms with sizes 5x5 - 20x20. This usually results in 8 or 9 rooms.

Next, we connect the rooms, which is done by iterating over the vector of room pointers and drawing the shortest paths between the indexes between random tiles in the two rooms. This is a very simple solution, but I was quite happy with the visual result so I decided to keep it and focus on implementing more features. Since pretty much all parts of the room and corridor generation is fairly encapsulated in their own functions/steps, I could easily continue working on other things and get back to the generation parts of the code later.

After this, I get the rooms with the highest and lowest X or Y values and change a random wall tile to a door; this is also where I spawn the player. The second door is assigned a random enum which decides what key you need to pick up to exit the dungeon.

The Rooms are then decorated based on their RoomType. When deciding RoomType I make sure that the dungeon always has at least one of the following:

- Key room containing the key required to leave
- Pickup room, containing either other keys or "health pickups"
- Enemy room with one or more enemies

The rest of the rooms then have a 25% chance of becoming any one of these types, or they will be a "normal" room with some non-interactable décor. Lastly, I decorate the outer world by iterating over the remaining non-room tiles, and give them a low chance to get a more decorative sprite, making the dungeon more visually appealing.

The gameplay is pretty simple, you control your character's movement using the WASD keys. Your objective is to reach the key that matches the color of the dungeon exit. Be careful not to walk over enemies, as this will reset your position to the start of the dungeon. The exit will only open once you've obtained the correct key for that dungeon.

Initially, I experimented with a feature where exiting one dungeon would immediately transition you into another. However, implementing this proved challenging as it required significant code restructuring to ensure proper clean-up of the previous dungeon instance and avoid potential memory leaks. Therefore, for now, the game simply ends once you've successfully exited a dungeon. This is definitely something I would want to continue working on, especially for the sake of deepening my understanding of proper memory management.

Overall I would also want to make a more sophisticated implementation of the room and corridor generation. Currently rooms are always squares, sometimes they sort of blend with a corridor which can provide a more interesting shape, but rooms of other shapes would be interesting. Along with this it would be nice to spend some more time on the corridors. Something like a breadth first search to connect nearby rooms and a variable to set the

amount of connections a room could/should have. This along with doing the corridors using Manhattan distance would probably provide a cleaner look of the overall dungeon. Visually it would also be nice to implement corner sprites to the rooms.

Known problems:

- Tiles that can't be walked on block corridors/doors, rarely happens
- Janky player movement, this is because of the SDL_Delay after each movement input. Fast hacky way to slow down the player without changing refresh rate of the entire program.