General Information:

I wanted to treat this project as an intellectual challenge, and I also wanted to implement many rooms with different predefined shapes, which there aren't many existing algorithms for, so I created an entirely new system. This project does not intentionally implement any existing algorithms, and uses concepts that we discussed in class, as well as concepts that I researched on my own.

Dungeon Generation:

Before generating the dungeon, every room prefab is instantiated once to get data from it, which includes available doors, the local position of all tiles in the room, and the offset required to add to the calculated position to make the doors line up properly.

After this, dungeon generation starts. The start room is instantiated in the beginning. And then, for each crawler, the DungeonController script will try to generate a random number of rooms between the specified minimum and maximum, with the first room building off of the start room. Regularly, the script picks a random room prefab from the available list, and then runs several checks to see if placing that room will work. If it is at the end of each crawler and the shop or boss room has not been generated, it will instead pick one of those.

Once a random room is picked, it checks if the room is a special room (boss or shop), and if that room has already been generated. If so, it picks a different room. When a valid room has been picked, if it is the first room of the crawler, it checks if it can be placed off of the start room. Otherwise, it checks if it can be placed off of the last room that was instantiated. It does this by querying the stored prefab data, and comparing that with the data for the last instantiated room. If the rooms have a door connection, ie left and right, or up and down, it then moves on to calculate a position for that room.

The position is calculated by finding the position of the door in the previously instantiated room that the new room is attempting to connect to. This is done by using the width of the previous room for horizontal doors, left and right, and the height for vertical doors, top and bottom. For horizontal doors, that position is also affected by height offsets set for different room prefabs. This allows for the L shapes to properly align with other rooms. This is not necessary for vertical placement.

Once that position is calculated, the script gets the local coordinates for each tile in the new room by querying the stored room data for the list of tiles, and adds the new position to each of those tiles. This converts their coordinates to world space, while maintaining the orientation of the room. Then, it checks if these tiles would overlap with any existing floorspace by getting the list of rooms, and checking against all of their floor tiles in world space. If there is no overlap, the room is placed.

Once all of these checks have been completed and the room is placed, it is added to the list of rooms, and the process continues until the crawler reaches its end, and then starts again with the next crawler.

And then, once all of the rooms have been placed, items and enemies are randomly placed on the floor tilemap of each room, within the specified range, and using the specified prefabs.

The Game Manager:

The game manager handles communication between different scripts for the processes that happen during the game. This includes updating and keeping track of the player balance when they pick up currency, their score when they defeat an enemy or pick up an item, and

managing ending and resetting the game when the player dies. It accesses the HUD scripts and updates these values, instead of directly doing that in the relevant classes, like item and enemy related classes.