Project 3

My goblin movement is found within **monsters.cpp.** The working movement in my game that allows players to be approached by the goblin is **Goblin::Move,** however it is not recursive. I have left my recursive goblin movement in my file as well, it is called **Goblin::goblinmove** and it’s recursive helper: **Goblin::reachesPlayer.**

My current **Goblin::Move** has my goblin approach the player in the same algorithm as the Bogeyman and Snakewoman. When the player is within 15 steps of the goblin, the goblin will move. Here is a high-level description of my goblin movement function:

1. The function retrieves the positions of the player and the goblin on the map.
2. Using the Manhattan distance formula, the function calculates the distance between he player and the goblin.
3. If the player is within 15 steps, the goblin moves.
   1. The direction the goblin moves is calculated based on the difference in positions between the player’s and goblin’s x coordinates, and their y coordinates. It moves according to which distance is shorter.
4. The calculated direction is checked for validity, making sure it is within the map boundaries, not occupied by a wall, or another monster, and not occupied by the player.

NOTE: the goblin move function is called by the game’s **moveMonsters** function which uses the goblin’s **CalculateDirection** function to cause movement.

My attempted **Goblin::goblinmove** recursive function. I will explain the logic behind this function. It is seen in my **monsters.cpp** file below the rest of my implementation.

1. This function uses the **reachesPlayer** function which is a recursive algorithm to determine if the Goblin can reach the player within a certain number of steps (15). It takes the goblin’s current position, the player’s position, and a list of past visited positions. All of the shortest paths of these is recorded in ‘bestPath’
   1. This function starts with base cases. If the number of visited positions exceeds the maximum 15 steps, the function returns false. If the goblin is adjacent to the player, the function adds the Goblin’s position to the “visited” vector and updates “bestPath” if the current path is shorter than the previously stored shortest path. It returns ‘true’ to indicate the player has been reached.
   2. The function checks if the goblin’s current position has already been visited through the vector, if it has already been explored, it returns ‘false’
   3. It explores all adjacent positions in the four possible directions (up, down, left, right), if the goblin can move in one of those directions, it calls itself recursively with the new position to continue searching for the valid path until the player is found.
2. Depending on the result from **reachesPlayer,** **goblinMove** updates the goblin’s position.

To generate randomized rooms and corridors that connect these rooms as well as display them in a grid with ‘#’ for walls and ‘ ‘ for movable space, I use **map.h, map.cpp, game.h, game.cpp, constants.h, coordinate.h,** and the provided utilities.

First, in **map.cpp**, the function **CreateRoomsAndCorridors** is the main function that creates the map.

1. A vector that will hold the generated rooms is initialized. The amount of rooms is generated as a random amount between 4-6, as that was the amount I found possible while playing through the provided game multiple times.
   1. NOTE: The grid is represented by a 2D grid of characters, (‘char grid[MAP\_HEIGHT][MAP\_WIDTH]’) with each cell in the grid representing a tile on the game map.
2. Using a loop that repeats until the correct number of rooms is made
   1. A room is placed randomly within the map boundaries by selecting a width and height for the room and then randomly choosing coordinates for it’s top-left corner. A room object, as declared in the map.h file, is created with these coordinates.
   2. The newly generated room is checked to ensure it does not overlap with other rooms using the **overlapsWith** bool, this function ensures that every room has atleast 1 ‘#’ as a buffer between them, and that no perimeters are shared between two rooms.
   3. If the room overlaps with a pre-existing room, the room is discarded and it tries again.
3. To generate corridors, once all the rooms are placed, these are generated to connect them.
   1. Every adjacent pair of rooms is iterated over. The midpoint coordinates between these rooms is calculated, prevX, prevY for the previous room, and currX, currY for the current room.
   2. Based on the position of the midpoints, it fills the map grids with corridor tiles to connect the rooms between them vertically or horizontal using a loop that places ‘ ‘ onto the grid coordinate.

Next, in **map.cpp,** the function **draw** is what displays the grid, shows the rooms and corridors, and is used to display the player ‘@’, any monsters, and spawn objects.

1. The function iterates over each row and column of the map grid, for each grid cell, it checks if there is a monster or game object at that position. If so, it draws the symbol.
2. If there is neither, it checks if the player is at that position, if the player is, it draws the ‘@’.
3. If none of the above is there, the map is drawn tile by tile by accessing the stored ‘#’ or ‘ ‘ at each coordinate based on the **CreateRoomsAndCorridors** function.
4. After the map is displayed, the stats are updated and displayed based on the player’s current statistics.

Functionality that I could not finish.

1. I couldn’t get my goblin recursion to fully work, so the working implementation in my game functions like the Bogeyman.
2. It is possible to view the inventory, but I cannot pick up Items from the map. While items randomly spawn & generate onto the map, the player is unable to walk on top of the item and pick it up.
3. Weapons are implemented, but scrolls are not.
4. Could not implement staircases and multiple levels.
5. Monsters are randomly generated and can be interacted with on the map, but they all spawn on level 0 for sake of seeing functionality since the further levels are not yet implemented.

Functionality that I could finish

1. Levels have randomly generated rooms and connecting corridors with walls blocking player movement, generated at execution.
2. Displays a level with a player that can move around and be blocked by walls.
3. Can encounter and interact with a monster.
4. Can fight monsters, can kill and be killed. Displays text according to actions.
5. Player is approached by bogeyman, snakewoman, and goblin when within range.
6. Player statistics are displayed and updated during attack/kill.