**Divya Sikka**

**206222701**

**Project 3 Report**

1. My recursive goblin function is called:

int canGoblinSmell(int playerRow, int playerCol, int goblinRow, int goblinCol, int smellDist, Game\* game, int\* nextMoveRow, int\* nextMoveCol, bool visited[MAX\_ROWS][MAX\_COLUMNS]);

This is implemented in the file “Actor.cpp”.

Description of Goblin Movement:  
My goblin movement function returns an integer containing the shortest amount of steps that a goblin can take to reach the player. It starts off with a smell distance of 0.

The base case checks are:

1. If it is greater than the maximum smell distance from the game constructor. If it takes longer, it returns 1000, a large number to indicate that the player is not reachable in the max smell distance allowed using this path.
2. If the game matrix is not empty for movement (there is an actor or a wall) - so this path is not possible
3. If the goblin has already visited this location of the matrix in one of the previous frames in this recursive call, so it would be unnecessary to check this location again.

To find the shortest path, the function then recursively calls itself 4 times with a different starting row or column each time (up, down, left, right) and the smell distance incremented by 1. If it reaches the player without returning 1000, it returns the smell distance, i.e. the amount of steps it took to reach the player.

The function then compares each of the paths taken to each other and returns the smallest path.

Additionally, the function takes two pointer parameters to the row and column for the next move. While checking for the minimum distance, it resets the next step for the goblin to either up, down, left, or right depending on each recursive call. In this way, the function returns both the smallest path and the coordinates of the next step the goblin must take to reach the player.

The reason the goblin sets its locations as visited is to save time for recursive calls that have already been taken (so the program doesn’t take too long).

This function is called by a separate function that resets the visited grid to false, and checks if the shortest distance returned is less than the maximum allowed. If this is true, the goblin will move.

1. Description of level generation with rooms/corridors:  
   In order to generate a level of the game, I call the “createLevel” function in the game class.

The first thing that this function does is set each character in the 2D array of the floor to a wall. Then, I call a function to create the rooms on the level.

This function uses the randInt() function provided to generate random starting places for the rows and columns of rectangles, and uses a while loop to continue generating these open rectangles (by calling another function). It makes sure that the rectangles will not reach the edge of the grid by using if statements. Additionally, the hallways are generated by generating 2 rectangles, and then calling another function that finds a common row between the two and creates another rectangle between them. It finds a random integer between where the two rectangles overlap, and creates an open rectangle between the two of them. The rest of the level is covered by walls.

After creating the rooms, if it is level 0, I randomly place the player at an empty space with no wall or monster. If it is not the first level, I randomly place the previously generated player at a new location so that the player still retains all of their inventory and information.

If the level is *not* 4, I randomly place the staircase. If it is 4, I randomly place the golden idol. I then randomly generate 2-3 objects using a helper function with switch cases between all possible objects and place the objects in a location with no other objects or walls.

I also randomly generate and place the given amount of monsters. The helper function that helps with this movement also takes the level into account (which is a private member variable of the class) in order to choose which random monsters to generate. It places these monsters in a viable location where there is no actor or wall.

Finally, the function displays the screen for the user.