**The name of your recursive goblin movement function and the name of the file it's implemented in, or else a statement that your goblin movement function is not recursive. (Make this the first thing, to help the grader find the function.)**

The name of my recursive goblin function is:

bool Goblin::canReachPlayer(int gobRow, int gobCol, int playerRow, int playerCol, int numberOfMovesSoFar, int maxDistance)

**a high-level description of your goblin movement function.**

In my goblin movement function, I have two base cases. The first base case is to return true if the player’s row and column matches the goblin. The second base case is to return false if we know the sum of the number of steps the goblin has taken and the shortest distance from it to the player is greater than the max number of steps we need the goblin to take to reach the player. I always want my goblin to move toward the player if possible so I calculate the difference in rows and columns to determine the direction it should take. The direction is important for the recursive part of the function so the goblin can take its next step, preferably toward the player. When the difference of the row/column is zero, I didn’t want to change that row/column so I included checks for that in the if statement. I also prioritize directions toward the player so that’s why my first two if statements check if the goblin can move toward the player. Otherwise, it moves in another direction away. I use a temporary maze to mark the spots the goblin has traversed so it doesn’t hit those spots again and call the isSafeToMove() function to see if it is possible for the goblin to move in that direction. If the goblin can move in a particular direction, I add its coordinates to a vector called “path”. If the goblin can’t move in any of the directions, then it returns false, which is placed toward the end of the function. To get the next coordinate, I get the first element in the path vector and then clear the path so that the vector can be used again in the next move.

**a high-level description of how you generate a level with rooms connected by corridors (or as much as you were able to implement).**

To create rooms, I first set a random number of rooms I want to generate. Each room is defined by its top left x coordinate, top left y coordinate, width, and height. I randomly set these parameters. I then check if they would create a valid maze in my isOverlapping() function, which checks if the rooms would overlap given their dimensions. If they do overlap, I repeat the process again, setting new parameters for my rooms in a do-while loop. Once I have the dimensions of all my rooms, I connect them with corridors by selecting rooms two at a time and creating a path that changes one room’s x-coordinate (r1X), creating a blank space in the maze at those coordinates until the two rooms have the same x coordinate. I do the same process with the y coordinates. This creates a clear pathway between all the rooms.

**a list of all functionality that you didn't finish, as well as known bugs in your implementations or serious inefficiencies, e.g. "I couldn't get goblins to move correctly, so I made them move like Bogeymen."**

I couldn’t fix my memory leaks despite writing destructors and using the delete keyword, traversing through every index in the vector.