**int** Goblin\_Recursion(**int** steps, **int** r, **int** c, **int** &bestDir, std::set<std::pair<**int**,**int**>> &visited, std::map<std::pair<**int**,**int**>,**int**> &m\_map){}

Class Goblins’ move function called this function

it is in the file Monster.cpp

This Goblin\_Recursion function take six arguments.

The frist one is step, it is the optimal step needs for goblin to reach the player

The int r and c is the current position of goblin

The bestDir is to determine the optimal next step the golin can take

The set called visited is the set to record if one position is visited already, like we mark on grid in homework3

The map is to map some position with the steps we need to reach that position.

In the function, we have two base case:

1. If the step need to reach the player is greater than the smile distance, return a number greater than the distance, so that the move function will not make goblin move.
2. If reach the player, stop the recursion and return the steps needed

We inert the current position to the set to mark it as visited

Then pass the current position and steps into the map to get the step needed for that specific position

We use row\_arr and col\_arr to let the for loop be able to move goblin to four different direction. In this case we don’t have to do a lot of if conditions

In side the for loop:

If we can’t move to that direction, or visited that direction already

Continue. (same as solve a maze)

If map[postion] give us 0, which means we first time reach here

or steps +1 is less than map[position], which means a better path

we recursively call the recursion to get the steps for rest

then if result < minimum step needed

set minimum needed equal to result.

Then check if (step == 0) ,

We can only reach here when the first time call the recursion, and that is what we need for the frist optimal move

After this, we remove the position from visited, because we want to unmark the set for further usage.

Return mimmum step we found for goblin to reach the player.

The function I implemented generate the map is in Temple.cpp

First I fill in the whole grid(map) by all ‘#”s.

Then I create a function to insert the room. This function takes random x1, x2, y1,y2, which are the four conner of a rectangle room.

It is between 4\*4 to 8\*8 when insert the room, just fill the grid by ‘ space ‘ for the entire rectangle.

Then I created a function to check if there is overlap between rooms

It is essentially before I insert a room, I check if the original’s grid whose the specific rectangle with four conners x1,x2,y1,y2 has ‘ space ‘ already in that place. If there is space, it means overlaps, create new position for the room to inster.

After I insert 4-6 room without overlapping

I used a stack to hold each room’s center, row\_cenetr and col\_center

Then for every two rooms, I try to connect their center. I created a connect\_corridor function, I start from the current room’s center than, I trace to next room’s center row by row, column by column. And set all grid to blank (space) on the way.