**Goblin function name:**

in Actor.cpp

virtual void move();

//Move helpers

bool findPath(int curX, int curY, int playerX, int playerY, int stepsLeft);

**High level description of Goblin movement:**

The Goblin movement functions are designed to allow the Goblin to navigate temple towards the player's position within a certain "smell" distance.

1. Goblin::findPath

This function is a recursive pathfinding algorithm that determines if there is a viable path from the Goblin's current position to the player's position within a given number of steps.

- Base Cases:

- The function returns `true` if the Goblin's current position matches the player's position.

- The function returns `false` if the Goblin has no steps left or if it encounters an out-of-bounds position or a wall.

- Recursive Case:

- The function marks the current cell as visited to avoid revisiting it within the same path search.

- It then explores all four possible movement directions (up, down, left, right) by recursively calling `findPath` with the new positions.

- If any of these recursive calls return `true`, indicating a path to the player has been found, the function restores the original cell state and returns `true`.

- Restoration:

- After exploring all possible moves, the function restores the original cell state before returning `false` if no path is found.

2. Goblin::move

This function determines the optimal move for the Goblin to make towards the player if the player is within the Goblin's "smell" distance.

- Initial Positioning:

- The function retrieves the current positions of both the Goblin and the player.

- It calculates the distance between the Goblin and the player.

- Distance Check:

- If the player is out of the Goblin's smell distance, the Goblin does not move.

- Movement Evaluation:

- The function defines the four possible movement directions (up, down, left, right).

- It initializes variables to track the optimal move and the minimum distance to the player.

- For each possible move, it checks if the new position is valid and if a path exists from this new position to the player within the remaining steps.

- It updates the optimal move if the new position brings the Goblin closer to the player.

- Optimal Move Execution:

- If an optimal move is found, the function updates the game grid to reflect the Goblin's move:

- It restores the previous cell's state based on whether it was an empty space, a special object, or a special position (e.g., a staircase or golden idol).

- It moves the Goblin to the new position and updates the Goblin's internal position state.

These functions together ensure that the Goblin moves intelligently towards the player within the constraints of the game's grid, obstacles, and the Goblin's sensing range. The use of recursion in `findPath` allows for flexible pathfinding while maintaining simplicity in the `move` function to handle the Goblin's actual movement logic.

**a high-level description of how you generate a level with rooms connected by corridors**

1. Clearing Previous Level Data

- The function begins by clearing any existing room data from previous level generations to ensure a fresh start.

1. Initializing the Level Map

- The entire level map is initially filled with walls (`'#'`). This creates a solid, impenetrable layout that will be carved out to form rooms and corridors.

1. Generating Rooms

- The function generates a specified number of rooms (`numRooms`).

- For each room:

- Random dimensions are assigned within a reasonable range to ensure rooms vary in size but fit within the level's boundaries.

- A random position is chosen for the room, ensuring it fits within the level and does not overlap the edges.

- The level map is updated to carve out the room by replacing the wall tiles (`'#'`) with empty space (`' '`).

- The room's details (position and dimensions) are stored in the `rooms` array for later use.

1. Connecting Rooms with Corridors

- Once all rooms are generated, the function creates corridors to connect them:

- For each pair of consecutive rooms:

- The function calculates the center coordinates of the current room and the next room.

- A horizontal corridor is drawn from the center of the current room to the x-coordinate of the center of the next room, replacing wall tiles with empty space.

- A vertical corridor is then drawn from the center of the current room to the y-coordinate of the center of the next room.

- The function ensures that the corridors connect properly at the intersection points to create continuous pathways between rooms.

1. Finalizing the Level

- After all rooms and corridors are generated, the function increments the `currentLevel` to indicate the progression to a new level.

- It then calls `createMonsters` and `createGameObjects` to populate the new level with monsters and game objects, making the level ready for gameplay.

**a list of all functionality that you didn't finish, as well as known bugs in your implementations or serious inefficiencies,**

* Kind of slow Goblins
* Rooms sometimes will overlap, causing a less maze-like shape
* Strange stuff happens with Goblins, sometimes when the path is blocked by another Goblin the behaviour is not correct