Towers 2

Due: Friday, May 10 @ 11:59pm

Objectives

1. Add AI players to Towers
2. Apply Graphs and BFS to add pathfinding to the AI

Description

The AI players will behave just like human player, except they will be controlled by your code. These AI players will be programmed to always take the shortest path to the enemy base and will ignore towers. The shortest path will be computed using the grid in each level where each square on the grid is represented as a node in a graph and the nodes are connected if a player can move between the tiles. An edge between two tiles means that they are adjacent in one of the 4 cardinal directions and neither tile contains a wall. With this representation, the shortest path can be found using BFS.

Project Structure

If you are starting from scratch:

1. Create a new project in IntelliJ
2. Pull the Scala Examples repo and copy the towers package into the src folder in your new project

If you're starting with your Towers 1 project, make the following changes to your existing code:

1. Copy the ai package from the Scala Examples repo into your project's model package
2. Add the Update case object to the MessageTypes file
3. Add the AIController to the ActorSystem and send it the Update message at a fixed interval (The last 2 lines added to TCPSocketServer in the repo. The repo sends Update 10 times/second. You can lower this if it's putting too much strain on your CPU)

Testing Objectives (0 points)

There are no testing objectives for this assignment.

Objectives (50 points)

Objective 1 (40 points)

AI Pathfinding.

Implement the getPath(String) method in the AIPlayer class to find the shortest path from this Players current location to the enemy base. The input is a JSON string containing the entire game state.

The returned path should be in the form of List of GridLocations. This list should include both endpoints (tile containing the player and the enemy base tile).

Assume that the player cannot move diagonally for this method. To use BFS for to find the shortest path you can represent each tile as a node in a graph that is connected to the 4 tiles adjacent to it, except for tile containing walls or tiles that are outside the boundary of the level.

To find the tile containing the player, take the player's location and convert their (x, y) location from doubles to ints. This effectively takes the floor of each value which results in the correct tile location.

If the player is on the same tile as the enemy base you will return a list of size 1.

Notice the AIPlayer class contains the id of the player. This can be used to find this player's location in the game state.

Objective 2 (10 points)

AI travel along a path.

Implement the pathToDirection(String, List[GridLocation]) method in the AIPlayer class to decide which direction to move the player given a path. To implement this method you'll need to find the player's current location using the game state, then return a direction that will move the player towards the next grid location in the path. If the path has size 1 (player is on the base tile) move towards the center of this tile. If the path has size > 1 move towards the center of second tile in the path.

Bonus Objective (25 points)

Ai move diagonally.

Implement the smoothPath(String, List[GridLocation]) method in the AIPlayer class to take a path and return the same path, but with diagonal movements wherever possible. The input string is a JSON string containing the entire game state (Use this to find the walls).

A diagonal movement is not possible if there is a wall adjacent to both tiles of the diagonal. For example:

Input path: (0, 0) -> (0, 1) -> (1, 1)
Wall at (1, 0)

Cannot be simplified since the path (0, 0) -> (1, 1) would have a diagonal with each end adjacent to a wall (If this were allowed, 50% of the time your AI would get stuck on the wall every time).

Note: This objective is deceptively simple. To appreciate the difficulty, consider this example:
Input path: (0, 0) -> (0, 1) -> (0, 2) -> (0, 3) -> (0, 4) -> (1, 4) -> (2, 4) -> (3, 4) -> (4, 4)
Wall at (2, 2)