COM316: Artificial Intelligence
Problem 4: Game Playing

Derin Gezgin — Jay Nash

The Problem

Intro

Now, we will add an opponent who can move the goal a step every time you move.

Given

1. Same as Problem 3.
2. Except that we are back to knowing the entire grid. In other words, we know the location of every obstacle on the grid. We also know the current location of the goal.
3. A function (move-goal C G) that takes in the current block of the robot and the position of the goal and returns the new position of the goal. This function will stochastically choose which way to go with the highest probability being away from the robot. Your solution should not make use of this information but should just assume the goal mover is an opponent intending to get away.
4. Assume the robot moves and then the goal moves and they continue to alternate. Both the robot and goal have the option to stay in place.
5. The goal is considered captured as soon as the robot is in an adjacent space.

Problem

Using what you know about search and the tree representation of it, devise a way for the robot to find the goal. As always, efficiency is important.

Note: The point of this problem is for you to find a way to deal with an opponent. You can be the robot as stated above, with the goal the opponent. Or if you prefer to do the problem from the other perspective, you can be the goal, with the robot as the opponent. The function above will be (move-robot C G). In either case, you and the opponent are going the same speed. The opponent is trying to win (stay away from you if it is the goal; catch you if it is the robot) and you are also trying to win. The one who will win is the one that is smarter (and/or often luckier).

We met to discuss this problem on September 25th

We know that the correct answer is *MiniMax* or *Monte-Carlo Tree Search (MCTS)*, considering it's advised us to use a tree representation of the search. We can store the possible game states in a search tree and calculate the combinations accordingly. It'll consider the opponent's potential future moves and determine the best possible outcome for the player (robot/goal). However, for this homework, we'll present two different approaches to solve this problem.

Our Solutions

H Algorithm*

Our initial idea for this problem was to use the A* algorithm to decide the robot's next move. At the same time, the goal would be to choose the worst option compared to the robot. While this approach is relatively simple and seems to work in some cases, it can be exploited easily by the other side (Robot/Goal) because it'll never catch the opponent unless we trap the opponent *by luck* or the opponent makes a mistake which a fully smart agent won't do.

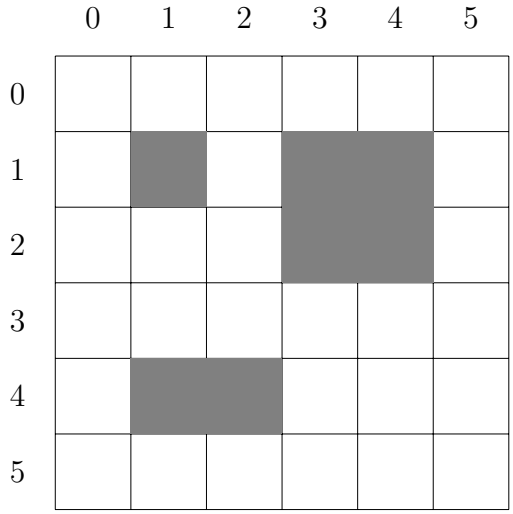
We came up with *Hunt ** or *H** algorithm, which is an A* but more cautious. Our algorithm will check the $\frac{\text{Grid Size}}{5}$ radius, and depending on what we are (goal/robot), it'll choose the best path and commit to it for $\frac{\text{Grid Size}}{5}$ moves. The point of this approach is to make fewer calculations and commit to one path for a while to prevent the other side from tricking us into wasting moves. We can not show this in a grid in this homework as we need a large grid to visualize this approach. Like the pure A* approach, this approach will also not catch the opponent if they don't make a mistake. On the other hand, it'll escape from the opponent as it will always consider the worst case and move accordingly.

Increasing the radius has advantages and disadvantages. If we increase the search radius, we can have a broader horizon for decision-making and make more informed decisions. At the same time, the robot would make fewer calculations. On the other hand, this reduces the robot's adaptability, and the opponent might take advantage of this.

It should be noted that our solution, H*, would still perform worse than MiniMax and MCTS, as we initially thought, but it can still be considered a good alternative, considering it demands less computation and can still work in non-fully smart agents.

Island Refuge

We came up with an extra solution that will only work if we play as the goal. Before explaining the solution in more detail, we first have to define what an island is:



This picture shows three different examples of islands on the grid. An island is a set of obstacles that are surrounded by open spaces. An island does not have to be a single line of obstacles; it can also be an arbitrary collection of connected nodes that satisfies this requirement.

After defining what an island is, we can move on to explaining the robot's behavior:

1. The goal would record its starting position and the robot's starting position.
2. The goal would scan through the grid -expanding from where it is- to find the closest possible island.
3. When an island is found, the goal would calculate the shortest path to the island using A* and the shortest path between the robot and the island.
4. If it's closer to the island, it'll start moving to the island.
5. Once on the island, the goal would settle and wait for the robot to come.
6. When the robot arrives on the island to catch the goal, the goal would go in the opposite direction around the island to escape from the robot. As both the robot and the goal can move only one square per move, the robot won't be able to catch the goal forever.

Despite this strategy being guaranteed to work, some crucial sub-conditions should be met for it to work.

1. There should be at least one island in the grid.
2. At least for one of the islands, the shortest path from the prey to the island should be shorter than the shortest path from the predator to the island. As an estimation, we can say it should be shorter, but depending on the size of the island, the prey-island path can be longer than the pred-island path, and this solution can still work.