Project 1: Ghosts in the Maze

16:198:520

This project has a few purposes. First, to give you experience implementing and using search algorithms to solve problems. Second, to highlight the difference between planning and execution - a plan may be based on incomplete information, and may need to be updated or adapted as it is executed. In this project you'll build an environment, an agent that has to interact with that environment in order to accomplish a goal, and you'll collect data to analyze the performance of your agent in pursuit of that goal.

1 Implementation

1.1 The Environment

The environment for this problem is a maze-like square grid. Some of the cells are open (unblocked) and some are obstructed (blocked). An agent in the environment can move among and occupy the unblocked cells, but cannot enter the blocked cells.

We want to generate many environments to test our agent in, so to do so we will generate these mazes randomly: *starting with an empty 51 x 51 square grid, iterate through each cell, and with probability 0.28 make it blocked, with probability 0.72 make it unblocked.*

However, since these mazes are generated randomly, they may not be very good (since blocks are distributed randomly, it's entirely possible every cell is blocked, for instance). So we need to check the quality of the maze, and reject ones that are too blocked. Remove any blocks present from the upper left and lower right corners, and make sure there is a path from one to the other (moving in up/down/left/right directions)

> *What algorithm is most useful for checking for the existence of these paths? Why?*

If the maze satisfies these conditions, accept it. Otherwise, reject it and start the process over again. You will want this process automated so you can easily generate lots of good mazes.

1.2 The Agent

The agent is going to start in the upper left corner, and attempt to navigate to the lower right corner. The agent can move in the cardinal directions (up/down/left/right), but only between unblocked squares, and cannot move outside the 51x51 grid. At any time, the agent can 'see' the entirety of the maze, and use this information to plan a path.

1.3 The Problem: Ghosts

Unfortunately for the agent, the maze is full of ghosts. Each ghosts starts at a random location in the maze that is reachable from the upper left corner (so that no ghost gets walled off).

> *Author's Note: I originally wrote this statement before I decided that ghosts could pass through walls.*

If the agent enters a cell with a ghost (or a ghost enters the agent's cell), the agent dies. This is to be avoided.

Each time the agent moves, the ghosts will also move. This means that whatever plan the agent initially generated to traverse the maze may at any point become blocked or invalid. This may mean the agent needs to **re-plan** its path through the maze to try to avoid the ghosts, and may have to repeatedly re-plan as the ghosts move.

The ghosts move according to simple rules: at every timestep, a ghost picks one of its neighbors (up/down/left/right); if the picked neighbor is unblocked, the ghost moves to that cell; if the picked neighbor is blocked, the ghost either stays in place with probability 0.5, or moves into the blocked cell with probability 0.5. (These rules apply even if the ghost is currently within a blocked cell.)

Every time the agent moves, the ghosts move according to the above rule. If the agent touches a ghost, the agent dies.

> *Author's Note: There are two ways this might happen - the agent enters a cell that contains a ghost, or a ghost enters a cell that contains the agent.*

1.4 The Strategies

The agent may take various strategies for navigating the maze. Your job is to implement these strategies, and compare and analyze their effectiveness.

Assume for the moment that each agent has total awareness of where all ghosts are at all times, even when they are in the walls.

Agent 1) Agent 1 plans a the shortest path through the maze and executes it, ignoring the ghosts. This agent is incredibly efficient - it only has to plan a path once - but it makes no adjustments or updates due to a changing environment.

> *Author's Note: When I say that the ghosts are ignored, I only mean as part of the execution. If the agent enters into a cell with a ghost, it will not ignore the ghost, it will die.*

Agent 2) Agent 2 re-plans. At every timestep, Agent 2 recalculates a new path to the goal based on the current information, and executes the next step in this new path. Agent 2 is constantly updating and readjusting based on new information about the ghosts. Note, however, Agent 2 makes no projections about the future. If all paths to the goal are currently blocked, Agent 2 attempts to move away from the nearest visible ghost (not occupying a blocked cell).

> *Author's Note: For convenience, let's define nearest visible ghost by shortest path to any ghost. Though if you've already implemented and generated data for nearest ghost in terms of manhattan distance or something, that's fine. Just be clear in your writeup.*

> Agent 2 requires multiple searches - you'll want to ensure that your searches are efficient as possible so they don't take much time. Do you *always* need to replan? When will the new plan be the same as the old plan, and as such you won't need to recalculate?

Agent 3) Agent 3 forecasts. At every timestep, Agent 3 considers each possible move it might take (including staying in place), and 'simulates' the future based on the rules of Agent 2 past that point. For each possible move, this future is simulated some number of times, and then Agent 3 chooses among the moves with greatest success rates in these simulations. Agent 3 can be thought of as Agent 2, plus the ability to imagine the future.

Author's Note: Given two possible moves with equal estimated survivability, you need some way to break the tie between them. Given two moves, you'd want to take the one that moves you closer to the goal - so compare the planned distance via Agent 2 for each move, and take the one with the shortest total distance. If they have equal total planned distance remaining, you can pick at random.

Also: Agent 3 is meant to mimic the behavior of Agent 2, plus a little bit extra. This means that in the absence of anything else, Agent 3 should default to Agent 2 behavior. In particular, if no move is survivable, then Agent 3 should default to running away from the nearest visible ghost.

Agent 3 requires multiple searches - you'll want to ensure that your searches are efficient as possible so they don't take much time. Additionally, if Agent 3 decides there is no successful path in its projected future, what should it do with that information? Does it guarantee that success is impossible?

Agent 4) Agent 4 is a free strategy - develop your own agent to solve the problem that beats Agent 3. How can you balance intelligence and efficiency? Note that the shortest path isn't necessarily the best one to take, if your goal is survival. (Note! An Agent 4 that works by testing Agent 3 at for every possible move before moving is not allowed! Try something else.)

Author's Note: What possible ways can you improve on the previous agents? Does each planned path really need to be the shortest possible path? How could you factor in distance to ghosts? What do they do well? What do they do poorly? Can you do it better? Challenge yourself!

2 Analysis

Along with the above implementation, you need to analyze the performance and the results of your code in a final **lab report**. The report should include the following:

- Discussion of design and algorithm choices you made, including but not limited to
  - Answers to the gray boxed questions above.
  - Description of your Agent 4, and how you implemented it.
  - Computational bottlenecks you ran into and how you dealt with them. To get this working with a large number of ghosts at a large dimension is going to take some work.
- Graphs comparing the performance of each agent for different numbers of ghosts. When does survivability go to 0?
  - Note that to get a decent graph, you'll have to repeatedly run experiments for each agent, with different mazes and each number of ghosts, and average the results together. The more times you can do this, the more accurate your estimates of success rates will be.
- How did Agents 1, 2, and 3, compare? Was one always better than others, at every number of ghosts?

> *Author's Note: If you are unable to get Agent 3 to beat Agent 2 - why? Theoretically, Agent 3 has all the information that Agent 2 has, and more. So why does using this information as Agent 3 does fail to be helpful?*

- How did your Agent 4 stack up against the others? Why did it succeed, or why did it fail?
- Redo the above, but assuming that the agent loses sight of ghosts when they are in the walls, and cannot make decisions based on the location of these ghosts. How does this affect the performance of each agent? Build an Agent 5 better suited to this lower-information environment. What changes do you have to make in order to accomplish this?