Project 1: This Bot is on Fire

16:198:520

It is another day on the deep space vessel *Archaeopteryx*, and you are a lonely bot. You're responsible for the safety and security of the ship while the crew is in deep hibernation. As an example, in the case that there is a fire on the ship, you are responsible for pressing the button to trigger the fire suppression system. Of course, you have to get to it first.

This semester I am trying something slightly new - this project writeup is a little longer than usual, because I have sprinkled through it my thoughts in boxes like this. I worry that I am giving too much away (and therefore making the project too easy), but I also worry that without additional guidance and clarity, people feel lost, hopeless, abandoned, and will likely miss the point. So in brief: read the gray boxes, and don't be scared by the length of this pdf.

*This project is meant to utilize and test the ideas from the first third of the course, in particular the ideas related to planning and search. You should anticipate having to implement various search algorithms. You should also anticipate that it will not be as simple as '**implement shortest path planning via BFS or Dijkstra**'. Additionally, one of the key concepts you should aim to wrap your head around is this: the distinction between **planning** a sequence of actions and **actually taking those actions**. We would like a sequence of actions to be optimal (or at least beneficial in some way!), and we would like **finding** that sequence of actions to be as efficient as we can make it. There is often a tradeoff between these two - less optimal solutions can be found more efficiently, more optimal solutions take more time to find.*

1 The Ship

We need an environment in which to plan actions - this is the ship. The ship is meant to be a maze that must be navigated. We would therefore like the maze to have interesting passages and hallways, and the below algorithm starts by iteratively 'opening up' the ship to produce these. However, we also need to avoid too many dead ends that can trap our bot, so the second half of the algorithm below identifies hallways that have 'dead ends', and opens them up further to increase access within the ship. This algorithm produces a maze, but with many possible loops and multiple ways to get from one location to another.

The layout of the ship (walls, hallways, etc) is on a square grid, generated in the following way:

- Start with a square grid, $D \times D$, of 'blocked' cells. Define the neighbors of cell as the adjacent cells in the up/down/left/right direction. Diagonal cells are not considered neighbors.
- Choose a square in the interior to 'open' at random.
- Iteratively do the following:
  - Identify all currently blocked cells that have exactly one open neighbor.
  - Of these currently blocked cells with exactly one open neighbor, pick one at random.
  - Open the selected cell.
  - Repeat until you can no longer do so.

- Identify all cells that are 'dead ends' - open cells with one open neighbor.
- For approximately half these cells, pick one of their closed neighbors at random and open it.

Note: In the above, I did not specify the dimension D of the ship. You should aim for $D = 40$, but if you run into difficulties because of your programs running too long, talk to me. Efficiency is one goal of this project, so try to get things running on as large a ship as you can. You should also try to make your ship generation process as efficient as you can.

Additionally: I strongly recommend that at any step where you are 'picking a cell to open from a set of possible cells', you randomize how you are selecting the cell from this set. If you do not randomize (for instance, reading through the available cells left to right and top to bottom), this can dramatically bias the distribution of hallways and dead ends within your ship, giving you very strange results.

Sanity Check: Before you open the dead ends, I estimate that about 60% of the ship should be opened by this process.

2 The Bot

The bot occupies an open cell somewhere in the ship (to be determined shortly). The bot can move to one adjacent cell every time step (up/down/left/right). Which action the bot takes will depend on the strategy being implemented, but you may assume that a) the bot has a map of the ship, and b) the bot is aware of the state of every cell of the ship.

Historically, many students have asked: can I add extra abilities to the bot, like fire suppression abilities? The answer is - no you may not. The bot moves, the bot will press one button. Your goal is to improve the bot's survivability within those constraints as best you can.

3 The Fire

At a random open cell, a fire starts. Every time step, the fire has the ability to spread to adjacent **open** cells. The fire cannot spread to blocked cells.

Note: because of the way the ship is constructed, this means that given enough time, the fire may spread throughout the entire ship.

The fire spreads according to the following rules: At each timestep, a non-burning cell catches on fire with the probability $1 - (1 - q)^K$:

- q is a parameter between 0 and 1, defining the *flammability* of the ship.
- K is the number of *currently burning neighbors* of this cell.

Every timestep, each cell is updated according to the above rules.

Note: The fact that this rule is applied to every cell means that in each timestep, multiple cells can potentially catch on fire.

Note: If q is very close to 0, $1-(1-q)^K$ will be close to 0, and the fire spreads with very low probability. If q is close to 1, then $1-q$ is close to 0, and $1-(1-q)^K$ is close to 1 - meaning that the fire spreads with very high probability. Additionally, the more neighbors a cell has that are on fire, the higher K is, and the closer $1-(1-q)^K$ will be to 1 - the faster the fire will spread. If the fire spreads incredibly quickly, even a very intelligent bot is going to struggle to avoid the fire.

Sanity Check: A common mistake here is to spread the fire to one new cell, update neighbor counts, then spread the fire again, until every cell has been processed. This is incorrect. Think about the following:

- Take the current ship
- Generate a copy
- For each open cell in the original ship (not currently on fire):
  - Count the number of currently on fire neighbors K.
  - Compute the probability this cell will catch on fire, based on K.
  - Pick a random number in [0,1] (uniformly, for instance with random.random() in python).
  - If this random number is less than the probability you calculated, set that cell on fire in the ship copy.
- Replace the current ship with the updated copy.

4 The Button

Located somewhere in the ship, in an empty cell (to be determined shortly), is a button that triggers the fire suppression system, instantly choking the fire of oxygen and putting it out. This must be pressed in order to stop the fire.

Does it matter if the cell with the button can catch on fire?

5 The Task

At time $t = 0$, place the bot, the button, and the initial fire cell at random open cells in the ship (three distinct open cells). At every time step, $t = 1, 2, 3, 4, \ldots$, the following happens in sequence:

- The bot decides which open neighbor to move to.
- The bot moves to that neighbor.

- If the bot enters the button cell, the button is pressed and the fire is put out - the task is completed.
- Otherwise, the fire advances.
- If at any point the bot and the fire occupy the same cell, the task is failed.

Ideally, the bot navigates to the button, avoids the fire, and puts the fire out. The goal of this assignment is to build and evaluate different strategies for governing how the bot decides where to go.

6 The Strategies

At any time, the bot can choose from the following actions: move to a neighboring open cell, or stay in place. The thing that differentiates the strategies is how the bot decides what to do.

For this assignment, you will implement and compare the performance of the following strategies.

- **Bot 1** - This bot plans the shortest path to the button, avoiding the initial fire cell, and then executes that plan. The spread of the fire is ignored by the bot.
- **Bot 2** - At every time step, the bot re-plans the shortest path to the button, avoiding the current fire cells, and then executes the next step in that plan.
- **Bot 3** - At every time step, the bot re-plans the shortest path to the button, avoiding the current fire cells *and any cells adjacent to current fire cells, if possible*, then executes the next step in that plan. If there is no such path, it plans the shortest path based only on current fire cells, then executes the next step in that plan.
- **Bot 4** - A bot of your own design.

Note: Your Bot 4 cannot simply be 'Bot 3 with a wider buffer on the fire cells.' This is not interesting. Be interesting.

For your Bot 4, you will want to consider the following:

- What makes a cell risky to move to? How could you measure it?
- Note that a cell being risky depends on *when* you move to it. A cell might be far away from any fire right now, but the fire may be likely to meet you by the time you get there.
- How could you think not just about where the fire is now, but where the fire likely to be in the future?
- You may assume your bot knows q and the rules for how the fire spreads, as well as the current fire, but it cannot know in advance exactly where the fire is going to be.

Note the connection to the earlier stated theme of planning vs execution. Planning a path through the ship is separate from the problem of executing it, because the fire can move as you execute a path - thus forcing you to change your plan.

Each bot will be evaluated on a number of ships, at various values of q, to establish its effectiveness.

7 Data and Analysis

In your writeup, consider and address the following:

1) Explain the design and algorithm for your Bot 4, being as specific as possible as to what your bot is actually doing. How does your bot factor in the available information to make more informed decisions about what to do next?

   - If your bot is basing its decision off a formula, you must tell us what that formula is and how to calculate it - do not make the grader go into your code for it.
   - If your bot depends on any specific values - for instance, any fire within a radius of 5 cells - you need to explain how you got that value. Why 5? Why not more? Why not less?
   - 'My bot carefully avoids the fire' - how? Be clear!

   A guiding principle should be this: if you were an ambitious grad student reading this report who did not write it or the code, would you be able to understand what was done from the report, and reproduce it yourself?

2) For each bot, explain anything you did to improve the speed and efficiency of your algorithms and data generation. It is not enough to just find a path or escape the fire - the computation needs to be done *quickly*, in order to get lots of data.

   - One thing to note here: this project is not as simple as 'for bot 4 I used A^* instead of BFS or UFCS like in the earlier bots'. Why?
   - What makes sense to use as a heuristic for this problem?
   - *Hint: There is a very useful heuristic that you can use that requires a little bit of pre-computation, if you're willing to invest the effort!*

3) For each bot, repeatedly generate test environments and evaluate the performance of the bot, for many values of q between 0 and 1. Graph the probability (or average frequency, rather) of the bot successfully putting out the fire. Be sure to repeat the experiment enough times to get accurate results for each bot, and each tested value of q.

   - Note, to get a good comparison between bots, you will need to generate a lot of data so that your graphs are smooth enough to give a valid comparison.
   - If your data is noisier than the improvements between bots, you won't be able to see the improvement! Averaging more data gives better results - but then you need to be able to generate data quickly.
   - Be sure to explain whether or not your data agrees with how you think it ought to look. What should your results look like when q is close to 0 or q is close to 1?
   - I will absolutely take off points if you do not clearly label your graphs and axes, and if you do not provide a graph of the performance for all bots plotted on a single graph for easy comparison.

4) Note that if a bot fails, it doesn't mean a better bot could have succeeded. Perhaps the fire was just such that no bot could succeed. Given a ship, bot and button positions, and the progression of a fire, write a test to determine whether that simulation was winnable. Describe your method, and plot (as a function of q), the frequency of a simulation being winnable.

> A hint, perhaps: if any of the bots succeed in a given simulation, we know for certain the simulation was winnable. But what if no tested bot succeeds?

> Does the shape of your resulting graph make sense? Why or why not?

5) Building off the previous question, plot (as a function of q), the frequency that each bot wins winnable simulations. This is similar to Part (3) above, but we are excluding simulations from our data that were not winnable.

> – Note that the better the bot, the more likely it should be to win winnable simulations.
>
> – Based on this rescaling, does your assessment of the bots and how they differ change?

6) When a bot fails to reach the button (and success was possible), *why* does it fail? Was there a better decision that could have been made that would have saved the bot? Support your conclusions.

> What I am looking for here is not 'the bot failed because it didn't avoid the fire' - what I am looking for is 'the bot failed because it made *this specific decision in this situation*, which made it impossible to escape the fire'. Additionally, if you are able, 'here is a point where it could have made a different decision'.

> *Note: This question has a bit of self-diagnosis built into it. If you can analyze why your bot fails, can you figure out a better decision that you could've made there, and thus improve your bot?*

7) Speculate on how you might construct the ideal bot. What information would it use, what information would it compute, and how? Do not worry too much about runtime.

> – *Sanity Check: If you are tempted to answer this question with, 'I would use advanced algorithms,' think twice. What algorithms? How would you process the information you have available? Be specific.*
>
> – *Sanity Check 2: If you are tempted to answer this question with, 'the ideal bot would know where the fire is going to be and act accordingly', think twice. The bot cannot know where the fire is going to be. This question is about how to use what is known about the fire to anticipate how the future might look.*