CS 440: This Maze is on *Fire*

16:198:440

A **maze** will be a square grid of cells / locations, where each cell is either empty or occupied. An agent wishes to travel from the upper left corner to the lower right corner, along the shortest path possible. The agent can only move from empty cells to neighboring empty cells in the up/down direction, or left/right - each cell has potentially four neighbors.

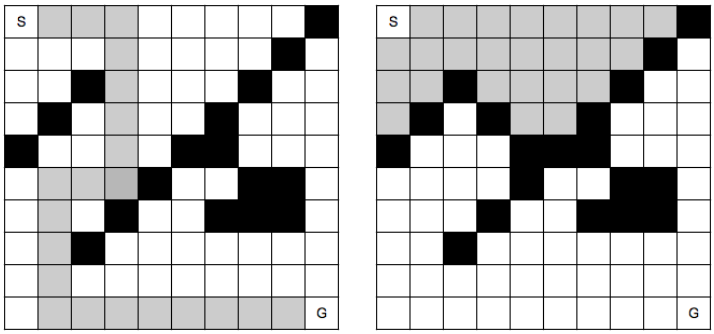

Figure 1: Successful and Unsuccessful Maze Environments.

Mazes may be generated in the following way: for a given dimension **dim** construct a **dim x dim** array; given a probability p of a cell being occupied ($0 < p < 1$), read through each cell in the array and determine at random if it should be filled or empty. When filling cells, exclude the upper left and lower right corners (the start and goal, respectively). It is convenient to define a function to generate these maps for a given **dim** and p.

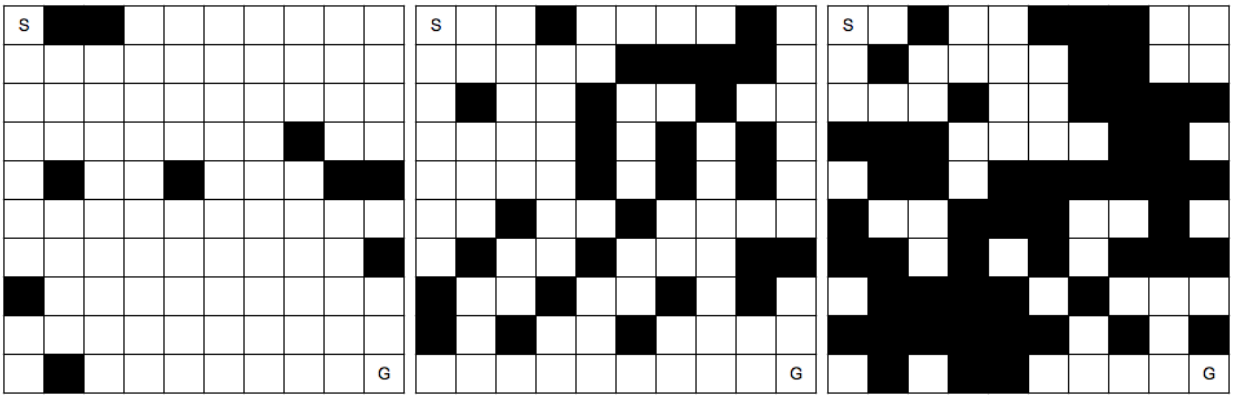

Figure 2: Maps generated with $p = 0.1, 0.3, 0.5$ respectively.

Preliminaries

- **Problem 1**: *Write an algorithm for generating a maze with a given dimension and obstacle density p.*
- **Problem 2**: *Write a DFS algorithm that takes a maze and two locations within it, and determines whether one is reachable from the other. Why is DFS a better choice than BFS here? For as large a dimension as your system can handle, generate a plot of 'obstacle density p' vs 'probability that S can be reached from G'.*
- **Problem 3**: *Write BFS and A^* algorithms (using the euclidean distance metric) that take a maze and determine the shortest path from S to G if one exists. For as large a dimension as your system can handle, generate a plot of the average 'number of nodes explored by BFS - number of nodes explored by A^*' vs 'obstacle density p'. If there is no path from S to G, what should this difference be?*
- **Problem 4**: *What's the largest dimension you can solve using DFS at $p = 0.3$ in less than a minute? What's the largest dimension you can solve using BFS at $p = 0.3$ in less than a minute? What's the largest dimension you can solve using A^* at $p = 0.3$ in less than a minute?*

Consider, as you solve these three problems, simple diagnostic criteria to make sure you are on track. The path returned by DFS should never be shorter than the path returned by BFS. The path returned by A^* should not be shorter than the path returned by BFS. How big can and should your fringe be at any point during these algorithms?

However, we are are not just interested in solving static mazes - these mazes are on *fire*, actively burning down around you. We model this in the following way: for every move the agent makes, the fire then advances (described below), and they alternate back and forth. The agent would like to exit the maze prior to running into the fire. Under this model, while an agent might utilize a search algorithm to generate an optimal path through the the *current* state of the maze (analyzing the current state and planning a future path through it), it will not necessarily be valid for *future* states of the maze. This means that while the agent may plan a short path through the maze, whether or not it will be able to take that path depends on the fire. In this case, the environment is dynamic, changing over time.

We model the advancement of the fire in the following way: any cell in the maze is either 'open', 'blocked', or 'on fire'. Starting out, a randomly selected open cell is 'on fire'. The agent can move between open cells or choose to stay in place, once per time step. The agent cannot move into cells that are on fire, and if the agent's cell catches on fire it dies. But each time-step the agent moves move, the fire may spread, according to the following rules: For some 'flammability rate' $0 \leq q \leq 1$

- If a free cell has no burning neighbors, it will still be free in the next time step.
- If a cell is on fire, it will still be on fire in the next time step.
- If a free cell has k burning neighbors, it will be on fire in the next time step with probability $1 - (1 - q)^k$.

We might express this with the following pseudocode for advancing the fire one step:

```
def advance_fire_one_step(current maze):
    create a copy of the current maze
    for every (x,y) in the current maze:
        if (x,y) is not on fire in current maze and (x,y) is not an obstacle:
            count the number of neighbors of (x,y) in current maze that are on fire
            store that as k
            prob = 1 - (1-q)^k
            if random() <= prob
                mark (x,y) as on fire in maze copy
    return maze copy
```

Note, for $q = 0$, the fire is effectively frozen in place, and for $q = 1$, the fire spreads quite rapidly. **Additionally, blocked cells do not burn, and may serve as a barrier between the agent and the fire.**

How can you solve the problem (to move the agent from upper left to lower right, as before) in this situation?

Consider the following base line strategies:

Strategy 1) At the start of the maze, wherever the fire is, solve for the shortest path from upper left to lower right, and follow it until the agent exits the maze or burns. This strategy does not modify its initial path as the fire changes.

Strategy 2) At every time step, re-compute the shortest path from the agent's current position to the goal position, based on the current state of the maze and the fire. Follow this new path one time step, then re-compute. This strategy constantly re-adjusts its plan based on the evolution of the fire. If the agent gets trapped with no path to the goal, it dies.

For each strategy, for as large a dimension as your system can handle, and obstacle density $p = 0.3$, generate a plot of 'average successes vs flammability q'. Which is the superior strategy? For each test value of q, generate and solve at least 10 mazes, restarting each 10 times with new initial fire locations. **Note:** Please discard any maze where there is no path from the start to the goal node. Additionally, please discard any maze where there is no path from the initial position of the agent to the initial position of the fire - for these mazes, the fire will never catch the agent and the agent is not in danger. *Your prior functions will be useful here.*

Come up with your own strategy to solve this problem (+5 points are available if you come up with a clever acronym for your strategy), and try to beat both the above strategies. How can you formulate the problem in an approachable way? How can you apply the algorithms discussed? Note, Strategy 1 does not account for the changing state of the fire, but Strategy 2 does. But Strategy 2 does not account for how the fire is going to look in the future. How could you include that?

A full credit solution must take into account not only the current state of the fire but potential future states of the fire, and compare the strategy to Strategy 1 and Strategy 2 on a similar average successes vs flammability graph. Additionally, while your strategy may increase survivability, analyze the time costs of your strategy compared to the other two - in a real burning maze, you have limited time to make your decisions about where to move next. Be clear and explicit in your writeup, presenting your reasoning and ideas behind your strategy.

- **Problem 5:** *Describe your improved Strategy 3. How does it account for the unknown future?*
- **Problem 6:** *Plot, for Strategy 1, 2, and 3, a graph of 'average strategy success rate' vs 'flammability q' at $p = 0.3$. Where do the different strategies perform the same? Where do they perform differently? Why?*
- **Problem 7:** *If you had unlimited computational resources at your disposal, how could you improve on Strategy 3?*
- **Problem 8:** *If you could only take ten seconds between moves (rather than doing as much computation as you like), how would that change your strategy? Describe such a potential Strategy 4.*

Your report needs to be clear to the point that the grader can understand and reconstruct your strategy from your description. If they can't, they're justified in giving no credit, and you'll have to argue for it back later.