CS440: The Maze is on Fire

Andy Rivera, John Juarez

February 19, 2021

1 Maze Generation and Project Setup

To generate a maze we used the function **generateMaze(int dim,double p)** it takes in the parameters of *dim* to construct a dimxdim 2D array and p to determine the probability of a block being filled(1) or not(0).

To set up the project for the path finding algorithms we created an object **Point** with the following attributes:

- Point parent (Previous location of agent)
- X and Y to keep track of location of agent
- stepsTaken (Amount of steps taken to get to the current location.)
- Hueristics (Estimate for A* algorithm)

```
Set dimension 'dim' to create maze
10
Set probability 'p' to create barricades
0.1
0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0
0       0       1       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       1       1       0       0       0
0       0       0       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0       0       1
0       0       0       0       1       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0
0       0       0       1       0       0       1       0       0       0
Set dimension 'dim' to create maze
10
Set probability 'p' to create barricades
0.6
0       0       0       0       0       0       0       1       1       1
0       0       1       1       0       1       1       0       0       1
1       1       0       0       1       1       0       1       1       1
0       1       0       0       0       1       0       1       1       0
1       1       0       1       1       1       1       1       1       1
0       0       0       1       1       0       1       1       0       1
1       1       0       1       1       1       0       0       1       0
1       0       1       0       1       0       1       0       0       0
0       0       1       0       1       0       1       0       1       1
1       0       1       0       1       0       1       0       1       0
```

Figure 1: Maps generated with p = 0.1 and 0.6 respectively

2 DFS Algorithm

We created the method **mazeDFS(int[][] maze, Point start, Point goal).** It takes in 2 points and returns a boolean, True if there exists a path and false if no path exists between the starting point and the goal point.

With two arbituary points in the maze, DFS would be a better option because we are just determining if there is a path between the two points and not necessarily the shortest path and saving space in memory since it adds less points into the fringe.

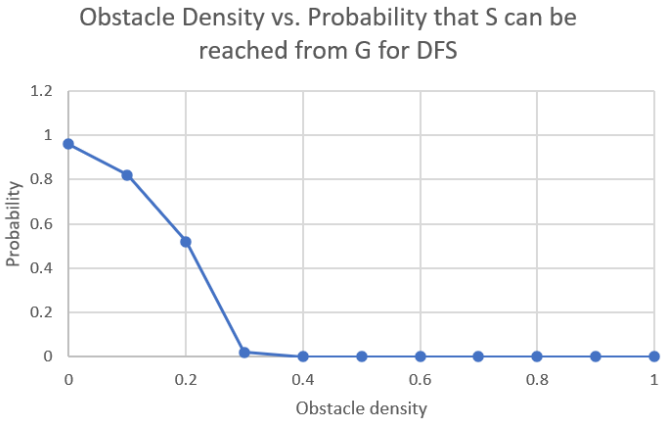

Figure 2: obstacle density p vs probability that S can be reached from G

3 BFS and A* Algorithms

For our BFS algorithm, we created the method **mazeBFS(int[][] maze).** It returns the ArrayList of the points that make up the shortest path.

For our A* algorithm, we created the method **mazeAStar(int[][] maze).** It uses a priority queue that prioritizes points based on an estimation that is determined by adding up the euclidean distance + the de-prioritization of steps that take you further from goal.

```
7 7 0 0 0 0 1 0 0 0 0 0
1 7 0 1 1 0 0 0 1 0 0 1
1 7 0 0 1 1 1 0 0 0 0 0
1 7 7 7 7 7 0 0 0 0 0 0
0 1 0 0 1 7 7 7 0 0 0 0
1 1 0 1 1 0 1 7 7 0 0 0
0 1 1 0 0 1 1 1 7 0 0 0
0 1 0 0 0 1 0 0 7 0 0 1
1 0 1 0 0 0 0 1 7 1 0 0
0 0 0 1 1 0 0 1 7 0 0 0
0 0 0 0 0 0 0 0 7 0 0 0
0 1 0 1 1 0 0 1 7 7 7 7
```

Figure 3: BFS Shortest path represented by 7 dim = 12 p = 0.3

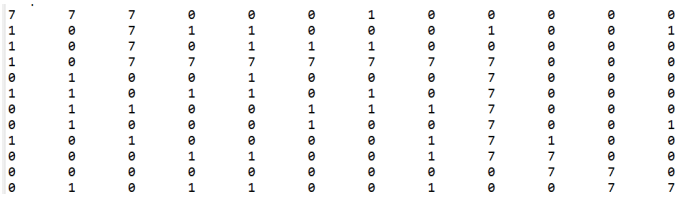

Figure 4: A* Shortest path represented by 7 dim = 12 p = 0.3

4 Algorithms in less than 1 minute

5 Maze is on Fire: Strategy 3

6 Strategies Success Rates

7 What if we had unlimited computational resources?

8 10 Seconds