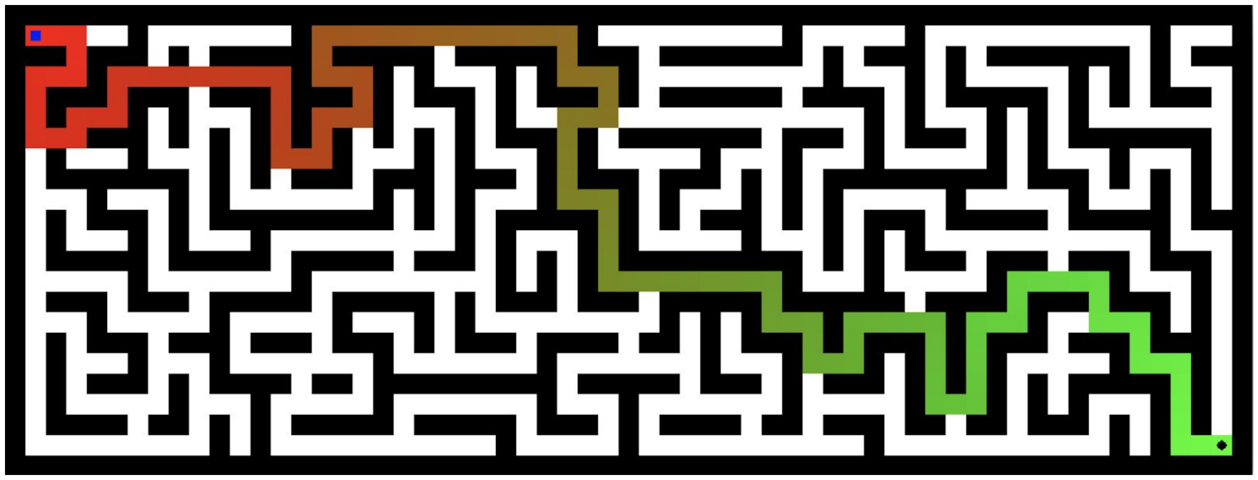
Garrett O’Grady, Rahul Kunji
Gogrady2, rahulsk2

Part 1: Basic Pathfinding

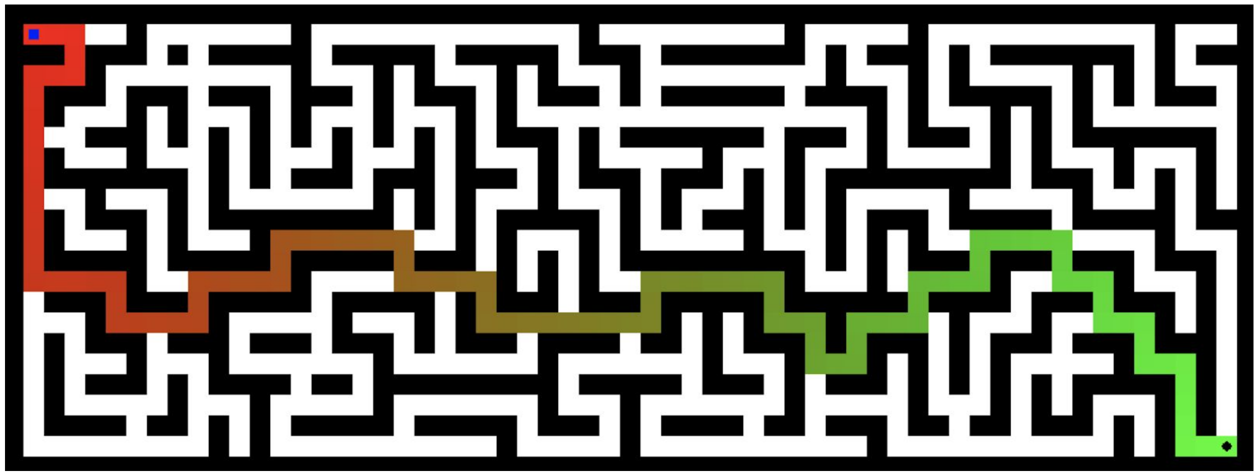
DFS, MediumMaze.txt



Path Length: 125
States Explored: 464

As expected the DFS works by first exploring a path on the frontier completely before choosing any other pathways

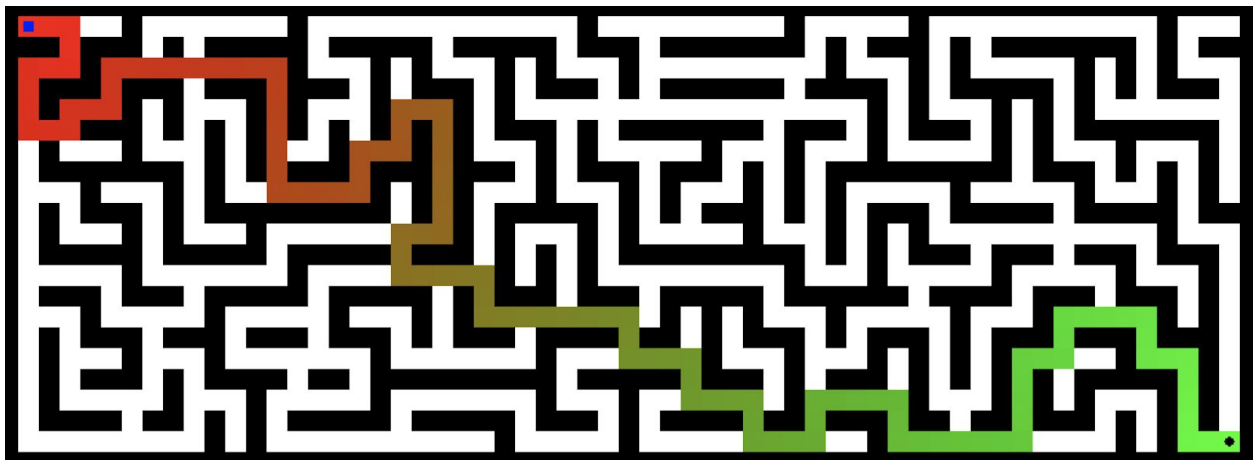
BFS, MediumMaze.txt



Path Length: 107
States Explored: 817

The Breadth First Search returns the most optimal solution

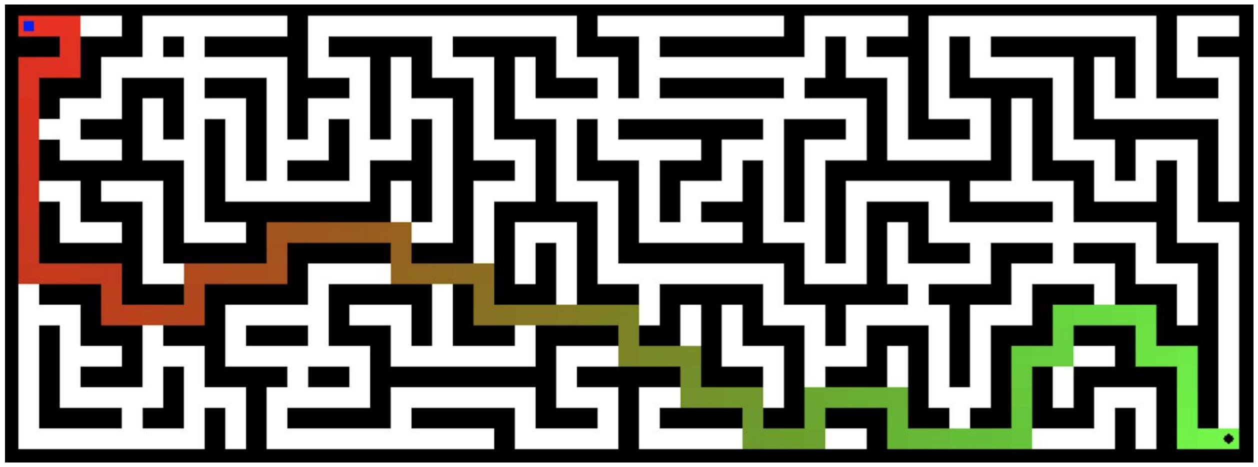
Greedy, MediumMaze.txt



Path Length: 117
States Explored: 144

The Greedy search albeit better than DFS still returns a very suboptimal path

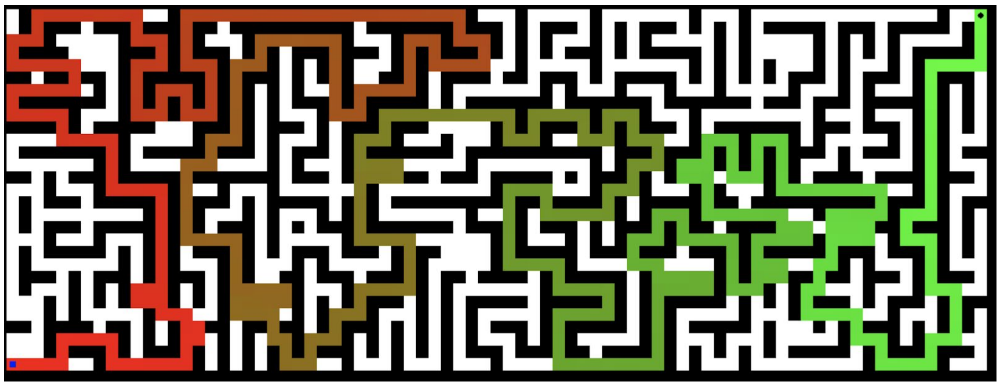
A*, MediumMaze.txt



Path Length: 107
States Explored: 542

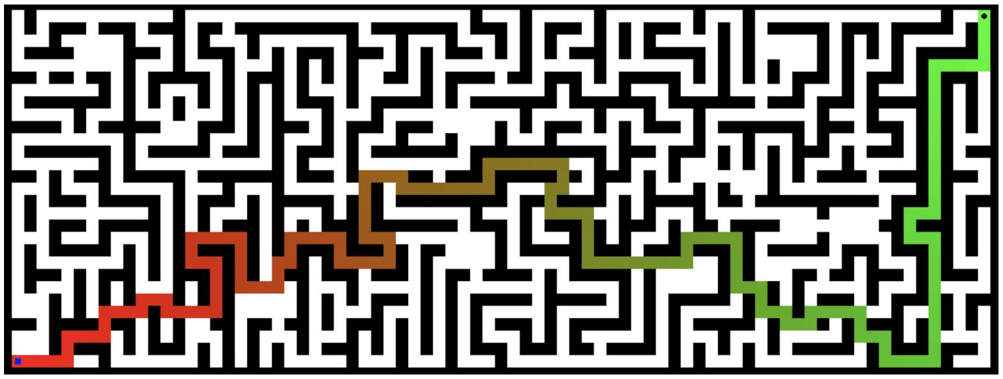
Astar achieves the optimal path length, and it does so while exploring much less states compared to BFS

DFS, bigMaze.txt



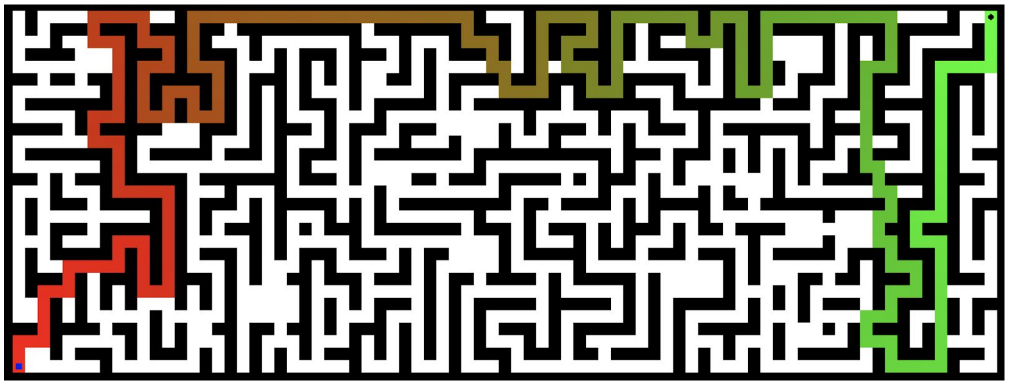
Path Length: 525
States Explored: 899

BFS, bigMaze.txt



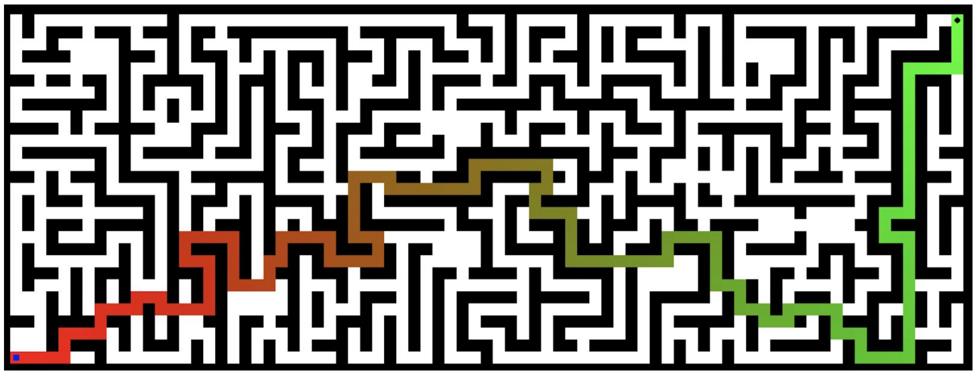
Path Length: 175
States Explored: 45420

Greedy, bigMaze.txt



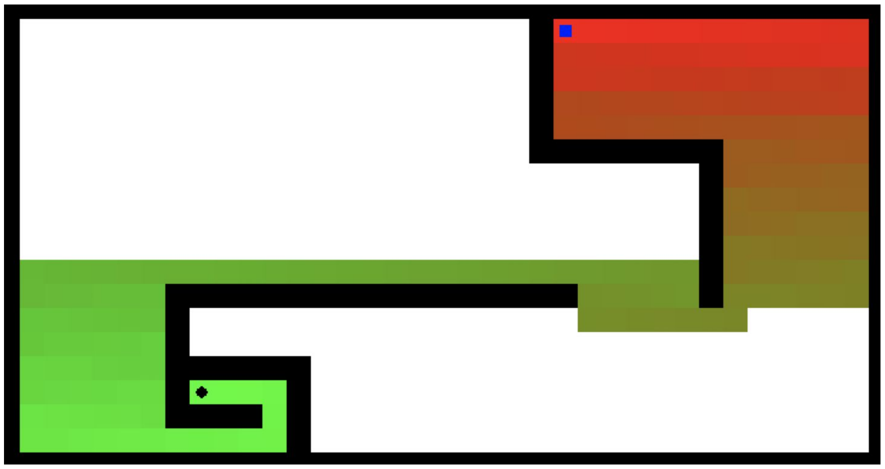
Path Length: 277
States Explored: 691

A*, bigMaze.txt



Path Length: 175
States Explored: 1498

DFS, openMaze.txt

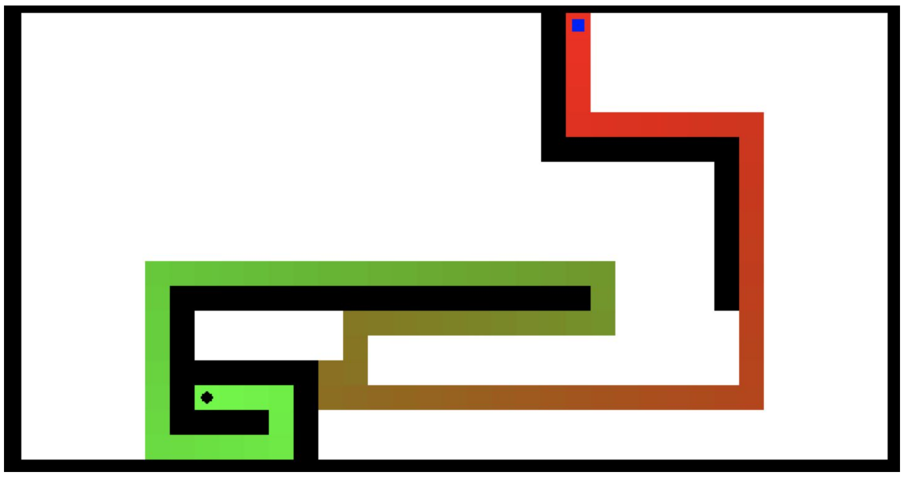


Path Length: 199
States Explored: 886

BFS, openMaze.txt

Did not return anything after running for a long time

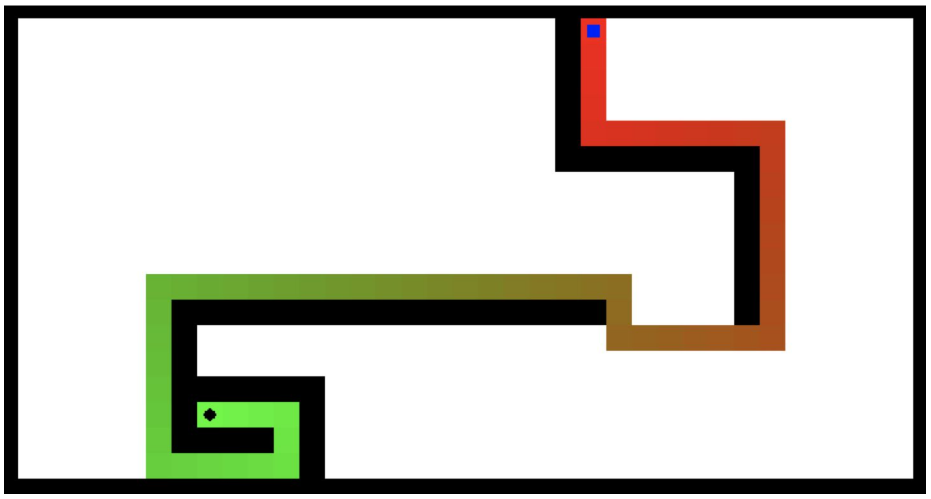
Greedy, openMaze.txt



Path Length: 91
States Explored: 3853

This is typical greedy behaviour, it tries to run towards the goal whilst ignoring the maze walls and then backtracks to another solution

Astar, openMaze.txt

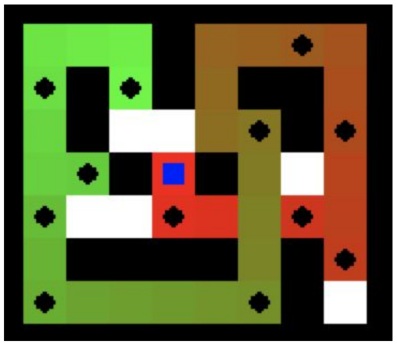


Path Length: 63
States Explored: 1863

Part 2

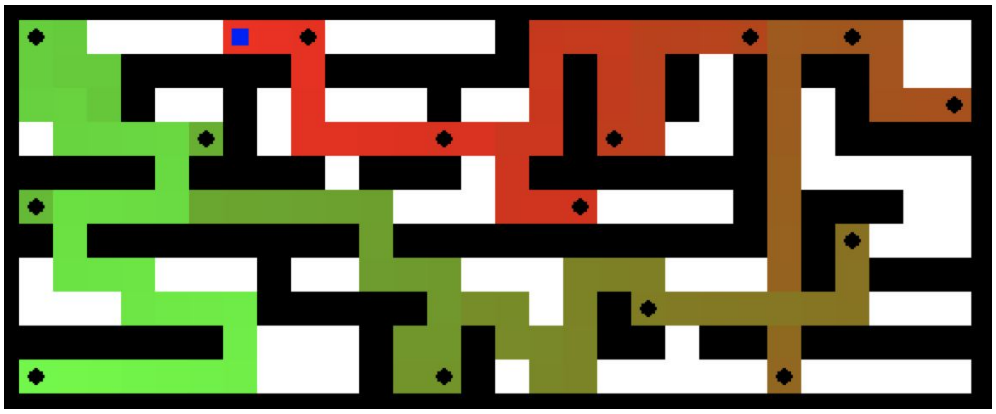
We implemented multi-goal versions of all search algorithms (including BFS, DFS and Greedy)
We tried out a bunch of heuristics with A* in this part, some fairly straightforward but some were a bit convoluted. We found out the weighted minimum manhattan distance to work best with a weight factor D of 2.
We know that this Heuristic is admissible and consistent as it never overestimates the path to the goal. It takes the minimum path and multiplies by D when there are more than D nodes, otherwise it just returns the minimum manhattan distance. This way we make sure to never overestimate the complete path cost.

A*, tinySearch.txt



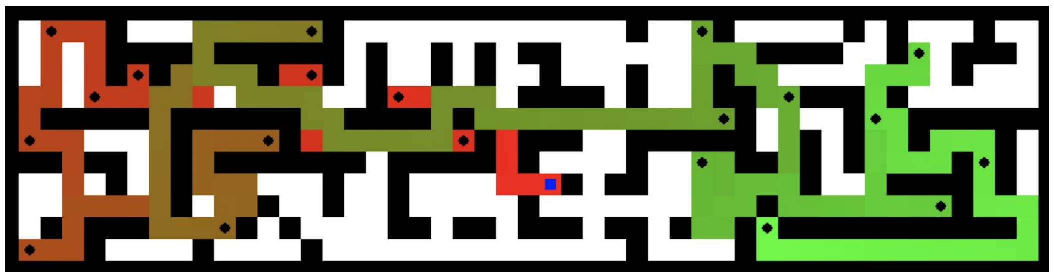
Path Length: 49
States Explored: 52

A*, smallSearch.txt



Path Length: 169
States Explored: 331

A*, mediumSearch.txt



Path Length: 250
States Explored: 497

Heuristic Description

Heuristic Used

Name: getWeightedAstarHeuristicMinDistanceToAnyObjective
Function: This is a weighted Manhattan Distance heuristic. We return the minimum manhattan distance to the nearest goal and multiply it by a constant factor. Another implementation could have been where we vary the constant weight to see what gives the best result

Other heuristics tried

Name: getAstarHeuristicSumOfAllGoals
Function: This is the sum of all manhattan distance from current to all goal state. This would not be admissible as it overestimates the cost to traverse all goals

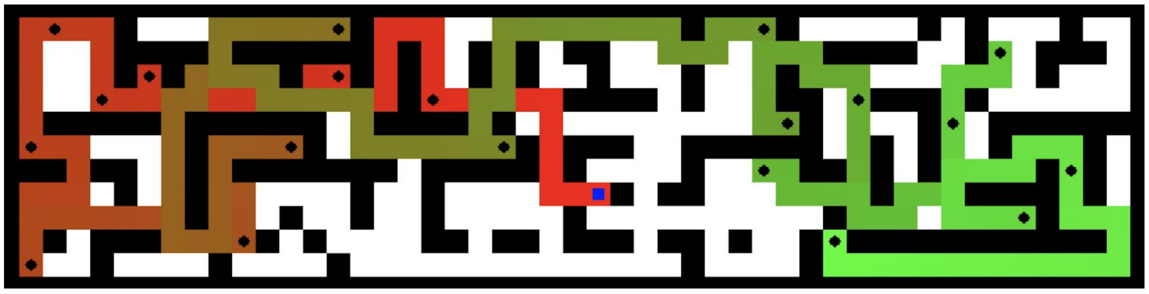
Name: getAstarHeuristicSumOfMinimumConnectedGoals
Function: Here we find the manhattan distance to the nearest goal, and then we sum it recursively with the path costs from that goal to the next nearest goal

Name: getAstarHeuristicSumOfMinimumConnectedGoalsPreComputed
Function: This is used in where we calculates the Dijkstra single source all paths distance between each goal state. The logic is that we precompute the manhattan distance from a each objective to every other objective. Then we calculate the heuristic value as h(x) = cost from current to nearest goal, dijkstra cost from that to all other goals. This was our most promising effort, but sadly we are getting a suboptimal result with this heuristic

Name: getAstarHeuristicBiConnectedComponents

Function: attempted to break the graph at cut vertices to turn the problem into a solvable traveling salesman problem. I (garrett) took a graph theory class in undergrad where we did this to solve a maze problem, however I could not replicate it for our maze. Essentially it would solve each connected component split at cut vertex, solve the left part of the graph and then solve the right part of the graph.

Name: getMaxDistanceHeuristic
Function: this heuristic calculated and stored the distance from each objective using bfs in a hash table, and found the max distance between two objectives. After that it calculates the distance of the neighbor to whichever one of the furthest objectives is closer and then adds that to the max distance of the two objects. It is admissible because you have to travel at least the distance at some point in the search between the max distance between the objectives and it is more efficient to collect the objectives nearby instead of returning to them.



Path Length: 248
States Explored: 3885