**Assignment 1: Search**

University of Illinois at Urbana-Champaign

Spring 2019 CS 440/ECE 448

Heting Fu                        hfu6                       Section Q

Yuhao Liu                 yliu167                  Section Q

Yuhao Min                    ymin6                    Section Q

Submitted on: February 4th, 2019

**Section I: Algorithms (Search)**

In all the algorithms, the node is simply the coordinate of the agent.

1. BFS: In BFS, the coordinates of the agent were used as the state representation. A node is a state in BFS. A queue (First-in-first-out, FIFO) was used to store all the state representations. In this case, the frontier will be all the neighbors. To detect repeat states, we used a (2x2 matrix of Booleans) to store if a point has been visited.

2. DFS: In DFS, the algorithm is the same as that in BFS except of the use of a stack (first-in-last-out, FILO) as frontier list. Specifically, the coordinates of the agent were used as the state representation. A stack (FILO) was used to store all the state representations. In this case, the frontier will be all the neighbors. To detect repeat states, we used a (2x2 matrix of Booleans) to store if a point has been visited.

3. Greedy: In Greedy search algorithm, the coordinates of the agent were used as the state representation. In addition, the Manhattan distance of the agent to the goal, defined by



was evaluated. A priority queue based on the Manhattan distance was used to store all the state representations, with the minimum distance at the top of the heap. In this case, the frontier will be all the end nodes of paths from the start node. To detect repeat states, we used a (2x2 matrix of Booleans) to store if a state has been visited.

4. A\*: In A\* search algorithm, the coordinates of the agent, the Manhattan distance between the current node and the goal, the current cost from the start to the current node and the number of remaining goals were used as the state representation. The frontier is a list that contains a set of end states from the start state. To detect repeat states, we used a (2x2 matrix of Booleans) to store if a state has been visited.

**Section II: Algorithms**

The heuristic used for single dot A\* and greedy BFS is Manhattan distance. In the maze, a optimal path from an agent to a goal will be a clear path, unobstructed by any wall, between them. Since in the maze, the agent can only move unit distance every time and in at only the x or y direction. The optimal path must have the distance equal to the sum of the absolute difference between the x coordinates and the absolute difference between the y coordinates, which is the Manhattan distance between the agent and the goal. Any path with a wall will increase the path length. That is, the Manhattan distance will never overestimate the cost of reaching the goal.

For multiple dot A\*, our heuristic is based on the Manhattan distance. Initially, a priority queue is used to store the Manhattan distance from the start to all the goals. The agent will find the goal with the smallest Manhattan distance. At every step, we will update the list with the Manhattan distance from the current node to all the goals. If at the current node, another goal becomes the closet goal, the agent will switch from the current goal to the new goal. Since Manhattan distance is admissible and we update Manhattan at every step, our heuristic will never overestimate the cost of going to the next goal. That is, our heuristic is admissible.

**Section III: Results (Basic Pathfinding)**

All screenshots can be found in the appendix. The solution costs and the number of expanded nodes are shown in Table 1.

**Table 1. Results of all algorithms on all mazes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Maze** | **Solution costs** | **Expanded states** |
| BFS | mediumMaze.txt | 111 | 646 |
|  | bigMaze.txt | 183 | 1286 |
|  | openMaze.txt | 52 | 559 |
| DFS | mediumMaze.txt | 175 | 374 |
|  | bigMaze.txt | 537 | 987 |
|  | openMaze.txt | 252 | 278 |
| Greedy | mediumMaze.txt | 147 | 348 |
|  | bigMaze.txt | 277 | 457 |
|  | openMaze.txt | 70 | 96 |
| A\* | mediumMaze.txt | 111 | 377 |
|  | bigMaze.txt | 183 | 1316 |
|  | openMaze.txt | 52 | 613 |

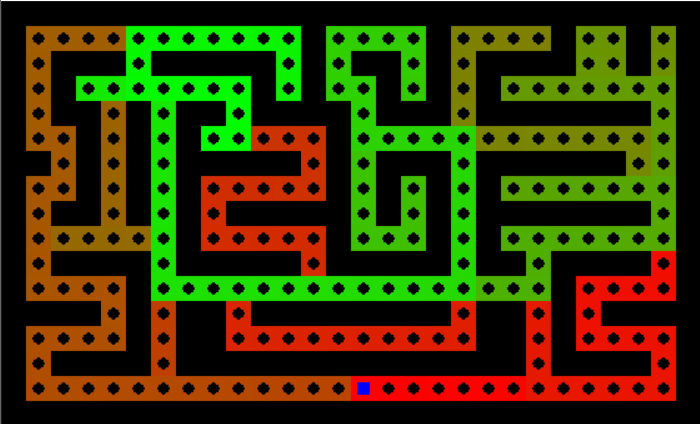
**Section IV: Results (Search with multiple dots)**

All screenshots are shown in the appendix. The solution costs and the numbers of states expanded are shown in Table 2. Although our heuristic is admissible. Our code fails to yield the optimal path. Possible sources of the error are potential edge cases which we have not considered, wrong calculation of the current cost and wrong path retrieval during back tracing.

**Table 2. Results of A\* algorithm for multiple dots**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Maze** | **Solution costs** | **Expanded states** |
| A\* | tinySearch.txt | 126 | 34 |
|  | smallSearch.txt | 327 | 196 |
|  | mediumSearch.txt | 556 | 363 |

**Extra Credit:**

We used a BFS based search strategy that does not guarantee an optimal solution but will always find a solution. The algorithm runs BFS from the start of the maze. Once it reaches an objective in the objective array, the objective is popped from the array, and another round of BFS starts at the objective location. The search ends when the frontier list (here we used a queue, FIFO) is empty or the objective array is empty. With this BFS based approach we were able to achieve a path length of 363 and states explored of 1041 on the bigDot maze.

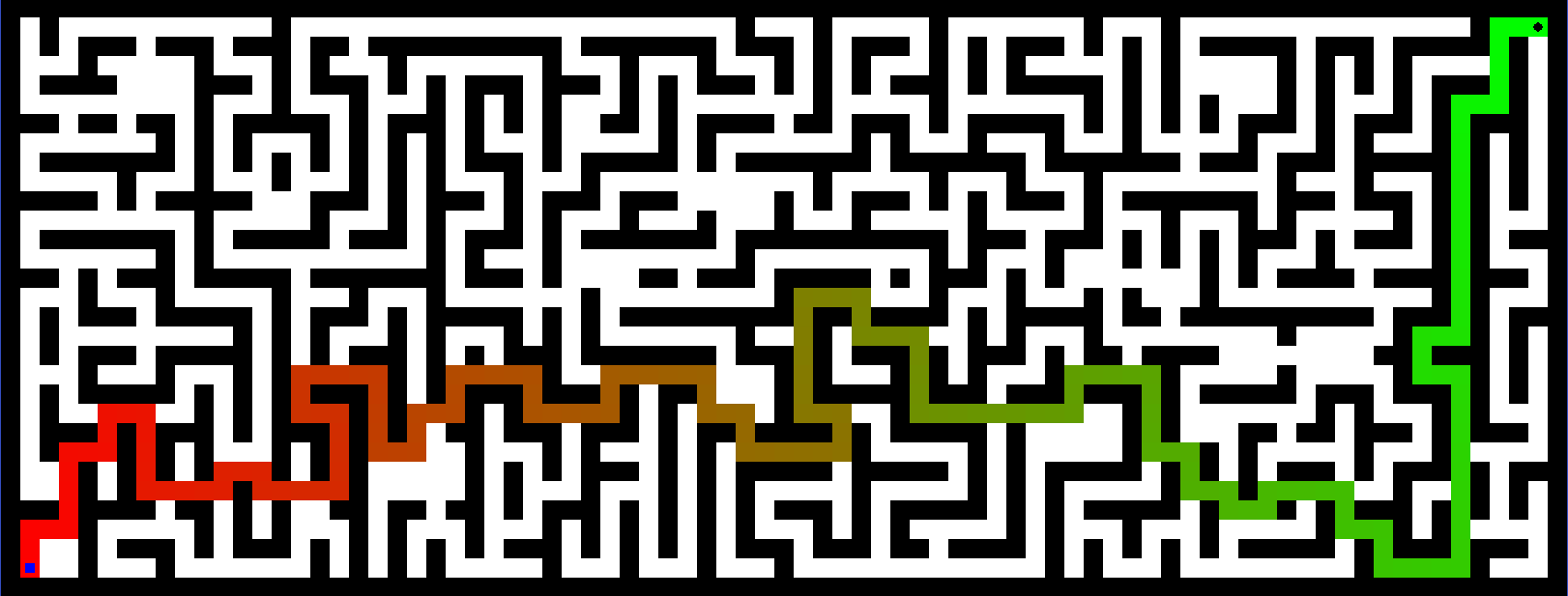
**Statement of Contribution:**

All members mostly did all of the algorithm on our own different versions and compared the result to ensure the result is as required. But for A\* algorithm for single and multiple dots we worked collectively and ended up using Heting Fu’s version. Since for the Section I, all of our version of codes can run successfully with the predicted outcome, for BFS we also used his version. And for DFS and Greedy, we used Yuhao Ming and Yuhao Liu’s final version of codes respectively.

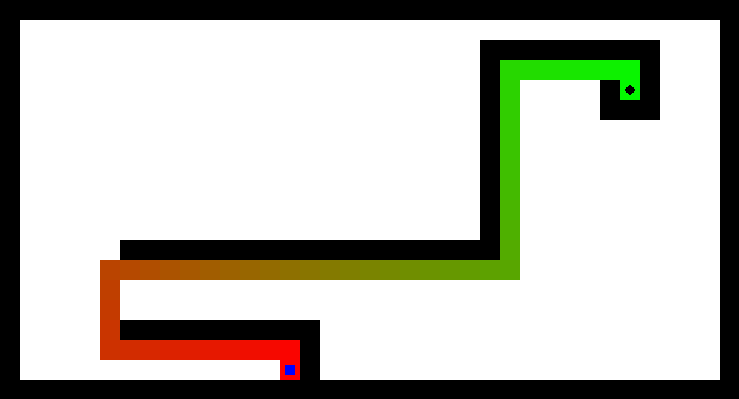
**Appendix**



**Figure 1. BFS on medium maze**

****

**Figure 2. BFS on big maze**

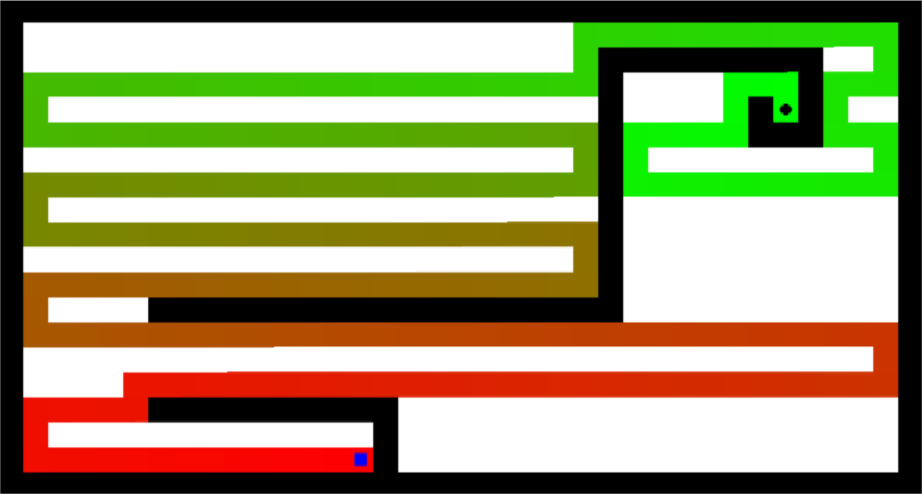
****

**Figure 3. BFS on open maze**

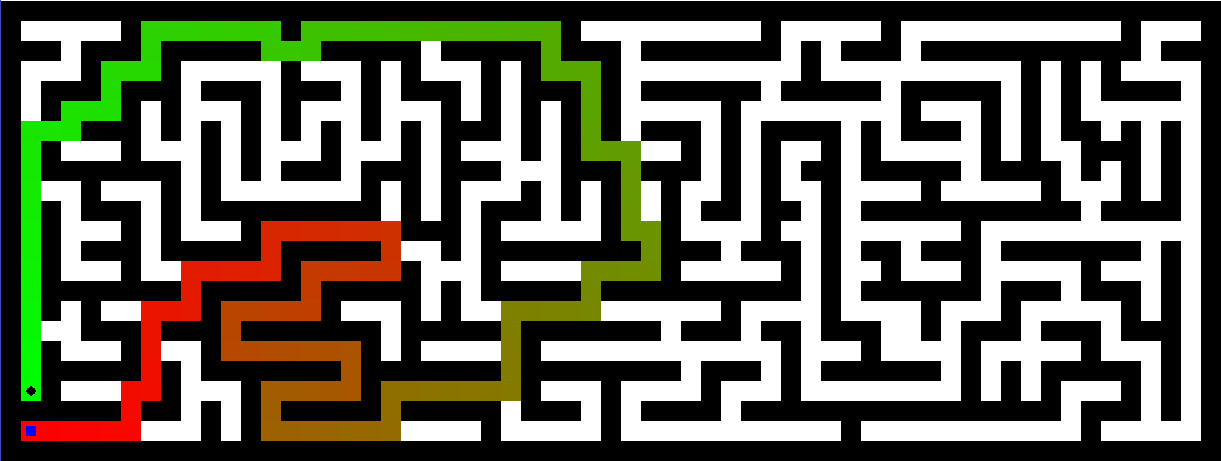
****

**Figure 4. DFS on medium maze**



**Figure 5. DFS on big maze**

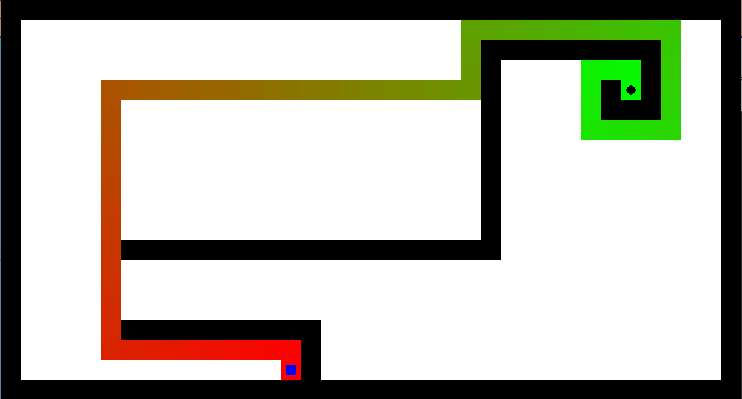
**Figure 6. DFS on open maze**

****

**Figure 7. Greedy on medium maze**

****

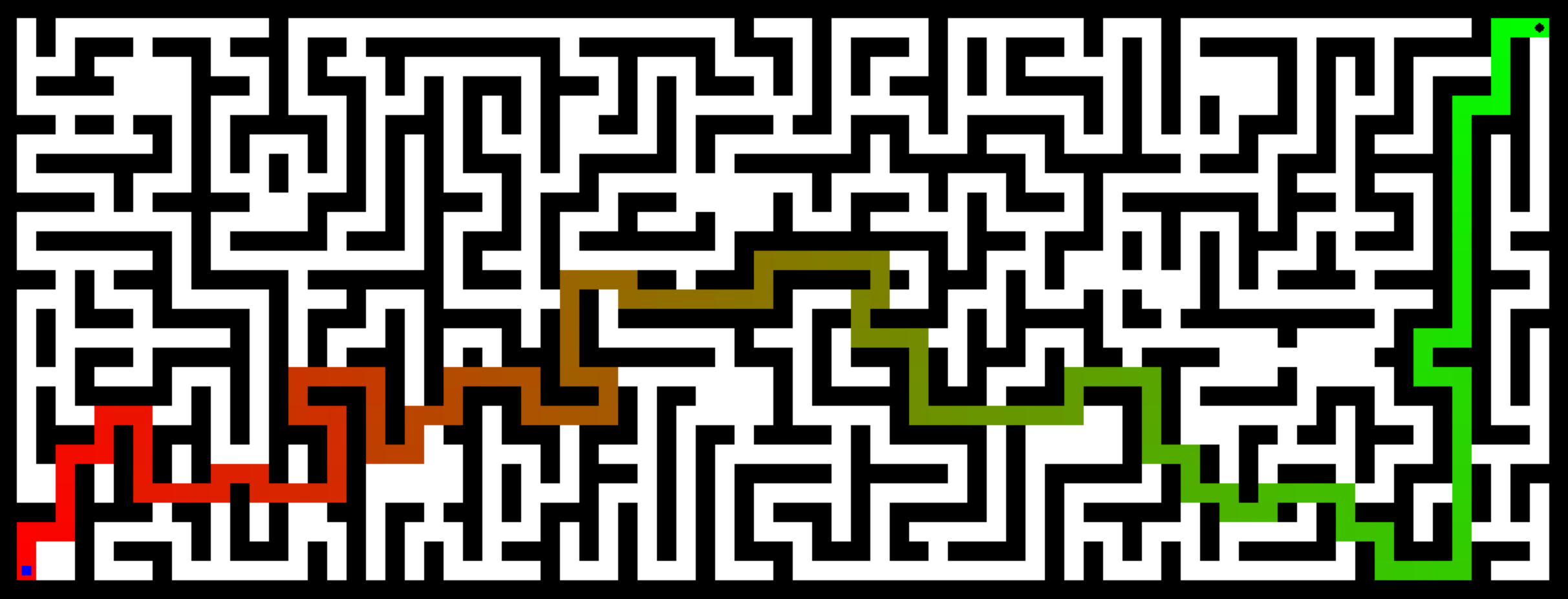
**Figure 8. Greedy on big maze**

****

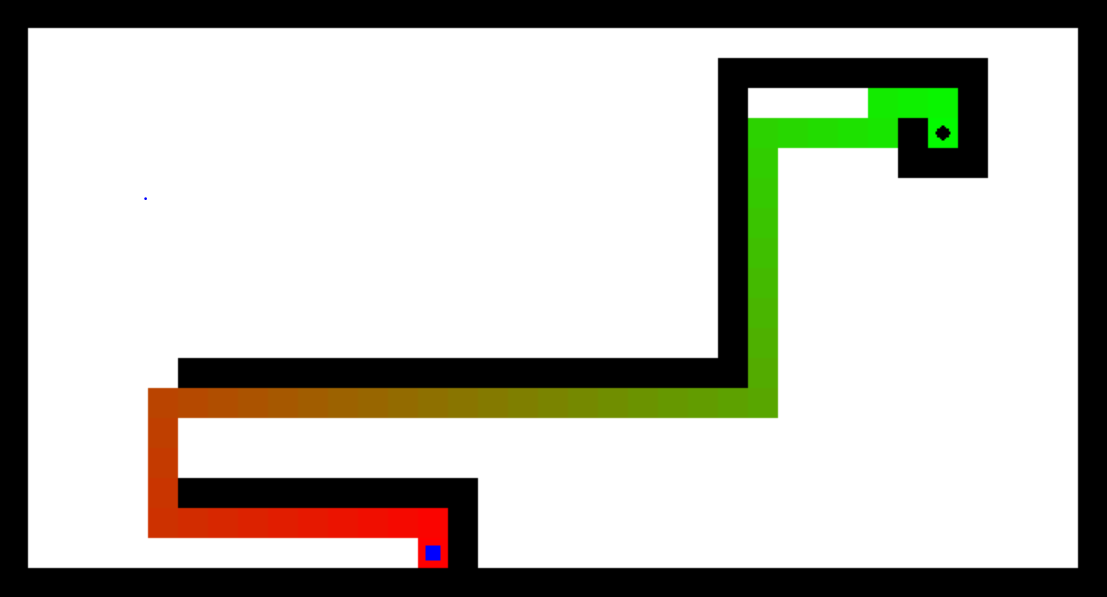
**Figure 9. Greedy on open maze**

****

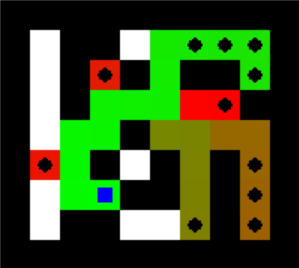
**Figure 10. A\* on medium maze**



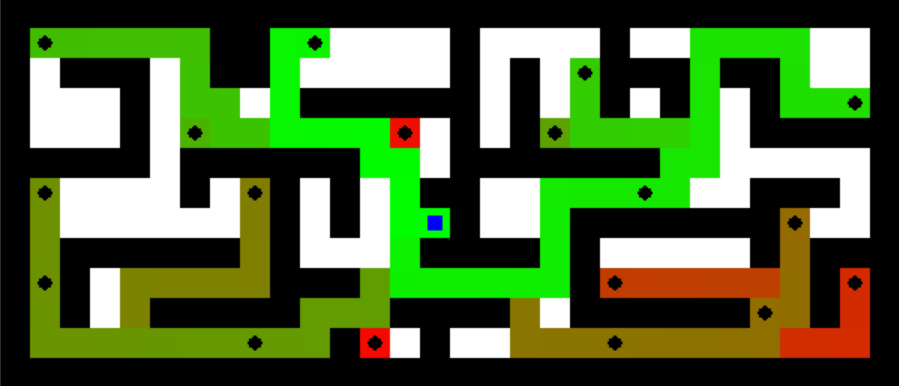
**Figure 11. A\* on big maze**



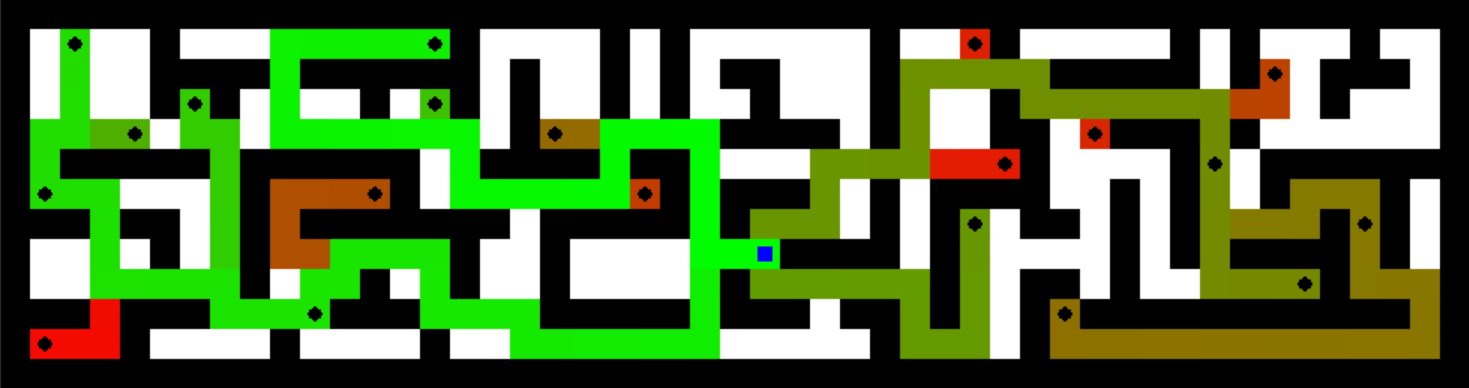
**Figure 12. A\* on open maze**

****

**Figure 13. A\* on tiny search**

****

**Figure 14. A\* on small search**

****

**Figure 15. A\* on medium search**