CS520 ASSIGNMENT 1

Sehej Jain
Rutgers University
sj1030

Gayathri Ravipati
Rutgers University
gr485

Ganesh Rohit Basam
Rutgers University
gb555

Part 0: Environment setup

To generate mazes of size 101×101, we worked on two algorithms. We wrote the first algorithm, which finds a random path from a starting point to a target point and then randomly assigns the remaining spaces in the grid as an obstacle or free space based on an input probability.

Once we generated multiple mazes of varying sizes, we observed that in many cases, the resultant mazes could be solved easily just by observing the maze. This was the case for most grids generated using this algorithm.

We then tried to look for algorithms that would help us generate more complex mazes. To generate the mazes for this assignment, we used a random acyclic maze generator with a given Entry and Exit point [1].

This algorithm ensures no cycles in the path while keeping track of how a node is discovered. We first check if any of the neighbors to the current node is a target. If so, we add it to the top of the stack; otherwise, we just shuffle the neighbors and add them to the stack, ensuring that the generated maze is random.

In the GUI we have prepared for the project, the agent's movement is represented using a blue square. The target is yellow, and the entire travelled path is highlighted in green and is displayed after it is discovered. Red cells are the blockers where the agent cannot go.

Fig. 1 shows the GUI of one of the generated mazes after completing the Repeated A star algorithm. More figures for the UI are added in the Appendix.

Part 1 - Understanding the methods

Explain in your report why the first move of the agent for the example search problem from Figure 8 is to the east rather than the north given that the agent does not know initially which cells are blocked.

It chooses to move east as we get the distance between $[5, 2]$ and $[5, 5]$ as $h = 3$, and the cost $g = 3$. It also does not have any blockers in the cell on its right.
If it chooses to move north, the g and h values will be higher, implying a higher f value. So the preference is for the lowest f-value path.
Moving towards the east has a lesser h value when compared to the north. Hence it moves towards the east.

This project argues that the agent is guaranteed to reach the target if it is not separated from it by blocked cells. Give a convincing argument that the agent in finite grid worlds indeed either reaches the target or discovers that this is impossible in finite time. Prove that the number of moves of the agent until it reaches the target or discovers that this is impossible is bounded from above by the number of unblocked cells squared.

For any $n \times n$ maze, let b be the number of unblocked cells.
The worst-case scenario for Repeated A* would have the following:

- The agent will not be able to move more than 1 step after getting the path from A*. It will encounter an obstacle every time it takes one step further.

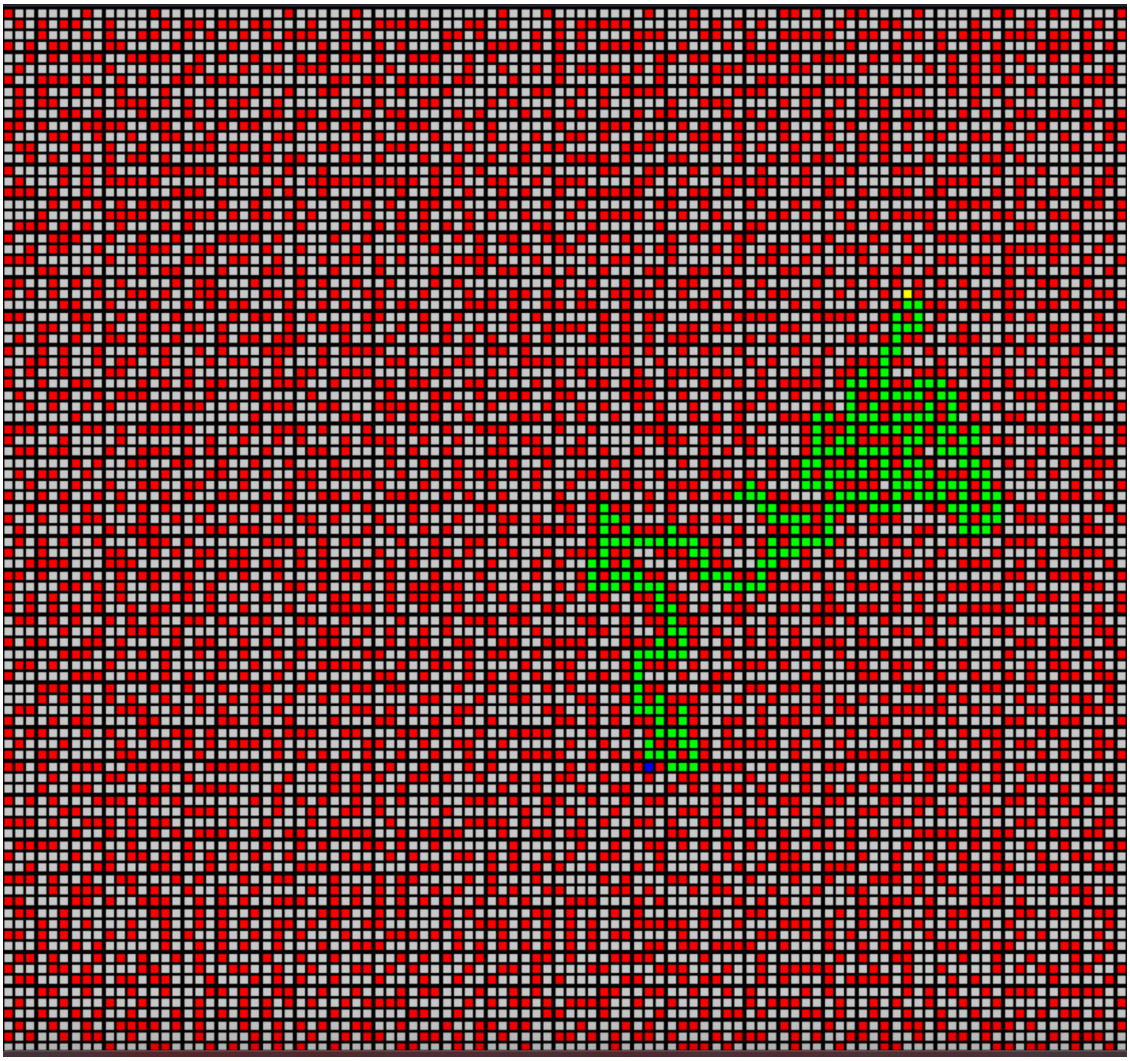

Figure 1: Output of the algorithm once Repeated A* is finished.

- It will not be able to reach the goal. However, it will not find this until the agent reaches closer to the goal. This means the Repeated A* runs for the maximum number of iterations but finds that the goal is unreachable just before it would have reached the goal state.

In any iteration of A*, the open list would have at most b nodes/blocks.
That means that the maximum length of the path returned by A* is b. For the worst-case scenario, the agent would call A* after each step, and the longest possible path, in this case, would be if it needs to go through each unblocked cell.
Also, in the worst-case scenario, repeated A* would have to backtrack to previous points/blocks. The algorithm would backtrack its movement until and unless A* found no path. In the worst case, it would have to go to each unblocked position b times.
Therefore the maximum number of moves for the agent is always less than or equal to b^2, which is the number of unblocked cells.

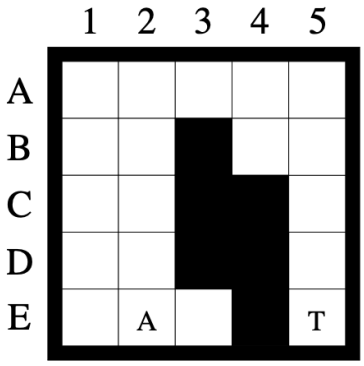

Figure 8: Second Example Search Problem

Figure 2: Figure 8 of the Assignment

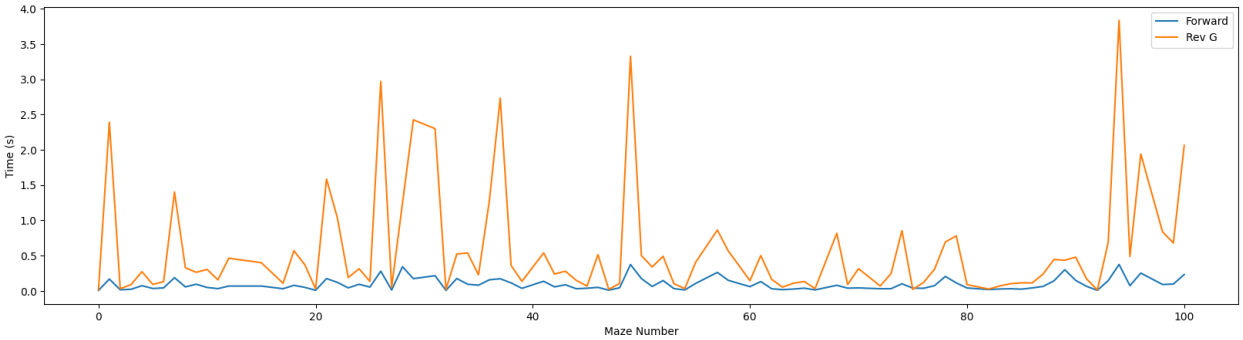

Figure 3: Plot of running time in seconds for the increasing and decreasing g values

Part 2 - The Effects of Ties

Repeated Forward A* needs to break ties to decide which cell to expand next if several cells have the same smallest f-value. It can either break ties in favor of cells with smaller g-values or in favor of cells with larger g-values. Implement and compare both versions of Repeated Forward A* with respect to their runtime or, equivalently, number of expanded cells. Explain your observations in detail, that is, explain what you observed and give a reason for the observation.

We observed that in the second case (lower g has higher priority), the algorithm took longer time than the first case in all the grids. Figure 3 represents our findings.

Setting higher g values as high priority for Repeated A* means that in the case two nodes have equal f values, then repeated A* expands the one farther away from the start node. This encourages A* to find the path leading away from the start point for A*.
If Repeated A* has low g values set as higher priority, then it would expand nodes closer to the start state first. So, this would take longer time to reach the goal.

Part 3 - Forward vs. Backward

Implement and compare Repeated Forward A* and Repeated Backward A* with respect to their runtime or, equivalently, number of expanded cells. Explain your observations in detail, that is, explain what you observed and give a reason for the observation. Both versions of Repeated A* should break ties among cells with the same f-value in favor of cells with larger g-values and remaining ties in an identical way, for example randomly.

Figures 4, 5 and 6 represent the plots of the two algorithms when compared on running time, number of expansions and number of A* calls.

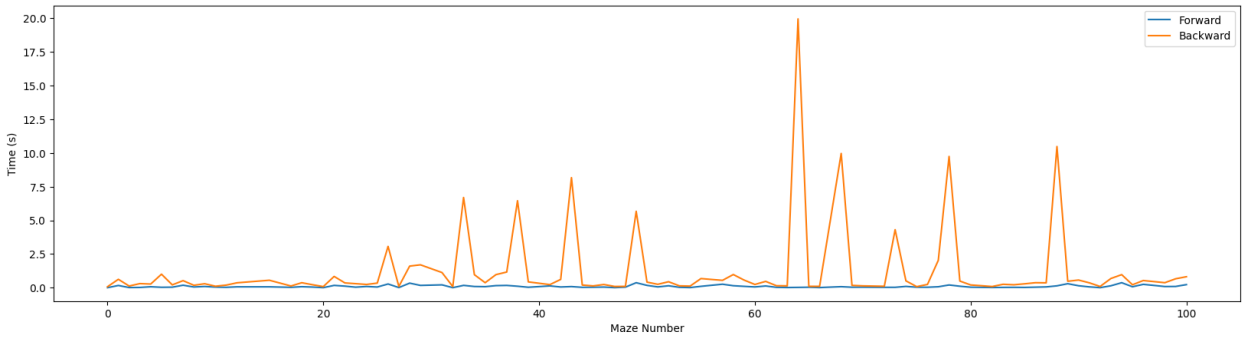

Figure 4: Plot of running time to compare forward and backward A*

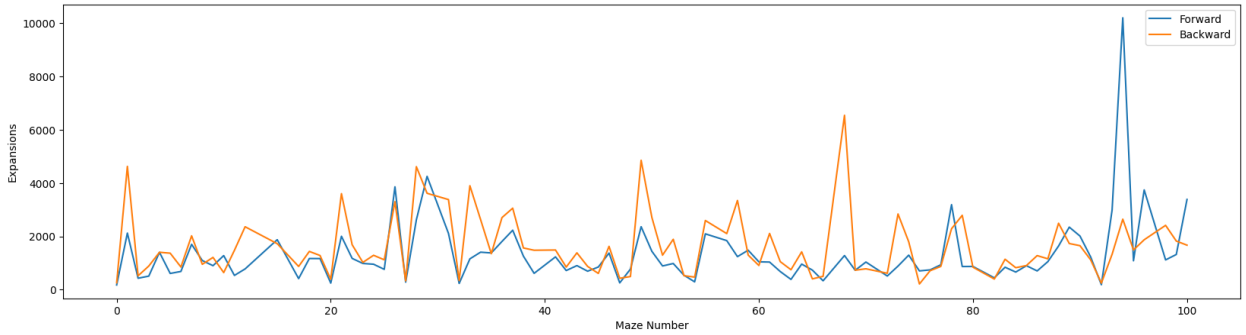

Figure 5: Plot of number of expansions to compare forward and backward A*

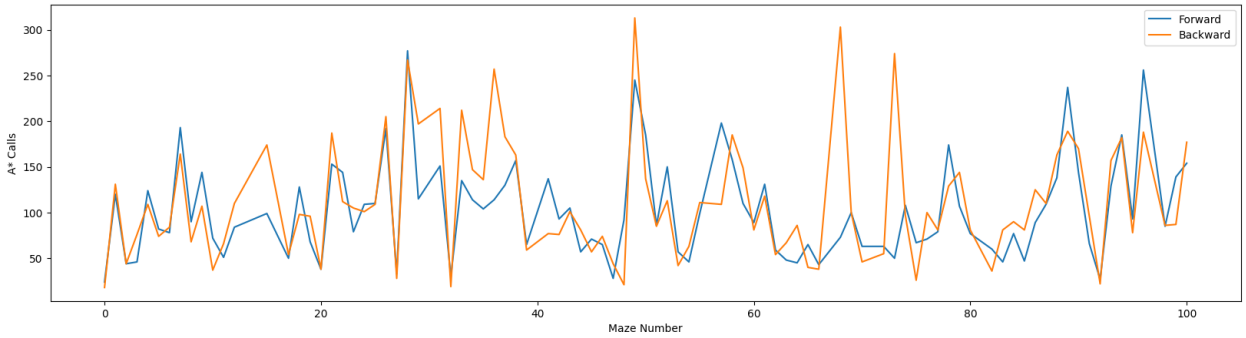

Figure 6: Plot of number of A* calls to compare forward and backward A*

We noticed that forward A* takes lesser time in all the cases. Backward A * is close to A* except in some cases when it takes drastically more time.

We believe that this is because of the maze structures and the relative density of the blockers near the start point and along the path.

In the case of backward A*, the algorithm gets us a path from the target to the start, which means it is acting blindly to where the blockers might be.

Once again, the performance of the two algorithms is highly dependent on the distribution of the blockers in the gridworld.

Part 4 - Heuristics in the Adaptive A*

The project argues that "the Manhattan distances are consistent in gridworlds in which the agent can move only in the four main compass directions." Prove that this is indeed the case.

A heuristic function h is said to be consistent if

$$\forall(n, a, n_0) : h(n) \leq c(n, a, n_0) + h(n_0) \tag{1}$$

We need to prove that if we consider Manhattan distance as a heuristic is consistent.
Let us assume that Source is at (i, j) and the target is at (x, y).

The Manhattan distance between these two points is $|x - i| + |y - j|$ which is the actual heuristic value.

Let us have the cost value after applying an action to move from n to n_0 is always 1. With that we have $c(n, a, n_0)$ as 1 for any a.
Therefore, we need to prove that $h(n) \leq c(n, a, n_0) + h(n_0)$.

$h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$
There are four possibilities of $h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$ from the initial step. We calculate the updated heuristic w.r.t the standard manhattan distance $|x - i| + |y - j|$

Case 1: The agent moves one step above the current position, where (i, j) is changed to $(i, j + 1)$. Replacing it on the equation we get:

$h(n) \leq c(n, a, n_0) + h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$

$$|x - i| + |y - j| \leq 1 + |x - i| + |y - (j + 1)|$$

$$|x - i| + |y - j| \leq 1 + |x - i| + |y - j - 1|$$

This holds the condition as $c(n, a, n_0) + h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$ will be equal to h(n).

Case 2:
The agent moves one step below the current position, where (i, j) is changed to $(i, j - 1)$. Replacing it on the equation we get:
$h(n) \leq c(n, a, n_0) + h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$

$$|x - i| + |y - j| \leq 1 + |x - i| + |y - (j - 1)|$$

$$|x - i| + |y - j| \leq 1 + |x - i| + |y - j + 1|$$

This holds the condition as $c(n, a, n_0) + h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$ has value higher by 2 when compared to the $h(n)$

Case 3:
The agent moves one step right to the current position, where (i,j) is changed to (i+1,j). Replacing it on the equation we get:

$h(n) \leq c(n, a, n_0) + h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$

$|x - i| + |y - j| \leq 1 + |x - (i + 1)| + |y - j|$

$|x - i| + |y - j| \leq 1 + |x - i - 1| + |y - j|$

This holds the condition as $c(n, a, n_0) + h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$ will be equal to $h(n)$.

Case 4:

The agent moves one step left to the current position, where (i,j) is changed to (i-1,j). Replacing it on the equation we get:

$h(n) \leq c(n, a, n_0) + h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$

$|x - i| + |y - j| \leq 1 + |x - (i - 1)| + |y - j|$

$|x - i| + |y - j| \leq 1 + |x - i + 1| + |y - j|$

This holds the condition as $c(n, a, n_0) + h(n_0)$ - is the heuristic of the new position we get when we do an action from $h(n)$ has value higher by 2 when compared to the h(n).

Furthermore, it is argued that "The h-values hnew(s) ... are not only admissible but also consistent." Prove that Adaptive A* leaves initially consistent h-values consistent even if action costs can increase.

Since the h_{new} values of Adaptive A star are consistent, then this eqn

$$h(s) \leq c(s, a) + h(succ(s, a))$$

for a state and all actions a that can be executed in from that state is satisfied. So the new h values in adaptive a star are consistent when the action costs are constant.

$$\begin{aligned} f(s) &\leq gd\,(s_{\text{start}}\,) \\ g(s) + h(s) &\leq g\,(s_{\text{goal}}\,) \\ h(s) &\leq g\,(s_{\text{goal}}\,) - g(s) \\ h(s) &\leq h_{\text{new}}\,(s) \end{aligned} \tag{2}$$

Moreover, the new h values of all expanded states s are not smaller than the immediately preceding h-values $h(s)$ and thus, by induction, also all previous h-values, including the user-supplied h-values. So the new h values are consistent for any iteration in the Adaptive A* algorithm.

So, in the case when the action costs are increasing, the above equation for consistency will still be satisfied even if the action costs increase because the h_{new} and $h(succ(s, a))$ are constant. So, the right-hand side of the equation is always greater than the left-hand side.

Part 5 - Heuristics in the Adaptive A*

Implement and compare Repeated Forward A* and Adaptive A* with respect to their runtime. Explain your observations in detail, that is, explain what you observed and give a reason for the observation. Both search algorithms should break ties among cells with the same f-value in favor of cells with larger g-values and remaining ties in an identical way, for example randomly.

Fig. 7 presents the plot of runtimes for Adaptive and Forward A* used in Repetitive A*.

In some cases, Adaptive A* is faster, as expected. However, this is not true for all cases. We believe that there might be some reasons behind this:

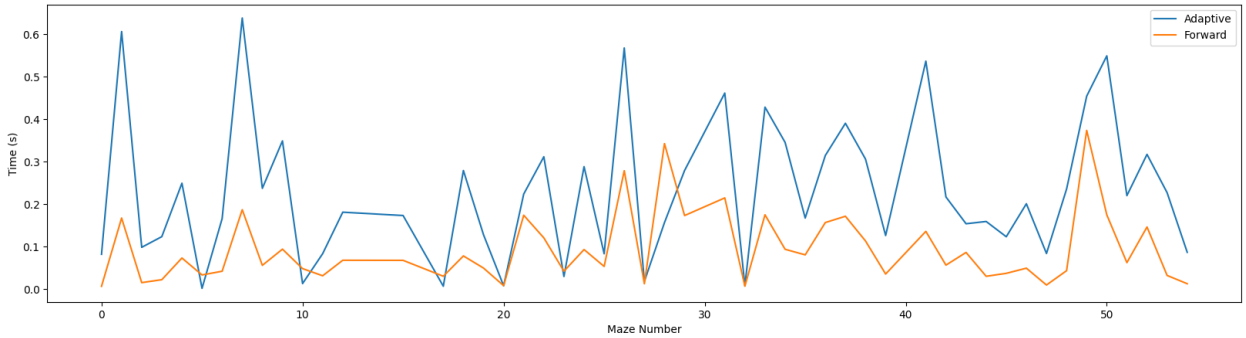

Figure 7: Plot of runtime in seconds for Adaptive and Forward A*

- We need to update the h values in each iteration and adds to the runtime significantly since we iterate to calculate all the new values.
- One reason could be the type of maze. Just as it was observed by Xiaoxun et al. [2], it depends on the situation, number of blockers and a lot of factors related to the grid.
- One last reason could be the choice of data structure.
- One more reason can be the sampling noise. We are working with a small number of mazes. With the large dimensions of the maze and, we cannot derive valid conclusions from just 50 sample mazes with no known patterns of obstacle sparsity or density.

Part 6 - Statistical Significance

Performance differences between two search algorithms can be systematic in nature or only due to sampling noise (= bias exhibited by the selected test cases since the number of test cases is always limited). One can use statistical hypothesis tests to determine whether they are systematic in nature. Read up on statistical hypothesis tests (for example, in Cohen, Empirical Methods for Artificial Intelligence, MIT Press, 1995) and then describe for one of the experimental questions above exactly how a statistical hypothesis test could be performed. You do not need to implement anything for this question, you should only precisely describe how such a test could be performed.

Considering the question in Part 2, where we compare the two algorithms which break ties in favor of cells with smaller g-values or in favor of cells with larger g-values.

We have two algorithms. The hypotheses in this case would be:

- H_0(Null Hypothesis) : The difference in the mean of the runtime speeds for the two algorithms is zero.
- H_a(Alternate Hypothesis) : The difference of the means is not 0.

The possible outcomes in this case are:

- The null hypothesis is refuted, or
- The Alternate hypothesis is refuted.

To perform a statistical test, we get sample data from the algorithms and use a test statistic to derive the possible outcomes. In this case, we need to generate a large number of random mazes with varying probability for blockers in the grids.
We need to derive our outcomes based on statistically significant results.
Once we have this, we choose the type of test. Some of the more popular tests are T-Tests and Z-Tests. Others include ANOVA Test and Chi Square test.

We chose the 2-tailed T-Test for testing the mean in our case. It is used to measure the significant difference between the means of 2 groups. For T-Test, we need the mean, standard deviation and the number of data samples for the two algorithms.

Then we set the confidence level. This is used to accept or reject our null hypothesis. Usually people take this as $p = 0.05$.

If our T-Test statistic $< p$, we reject the null hypothesis and this implies that our results are significant.

References

[1] Random Acyclic Maze Generator with given Entry and Exit point, https://www.geeksforgeeks.org/random-acyclic-maze-generator-with-given-entry-and-exit-point/, Date accessed: 2022-10-17

[2] Sun, Xiaoxun and Koenig, Sven and Yeoh, William. (2008). Generalized Adaptive A*.. 1. 469-476. 10.1145/1402383.1402451.

Appendix

Table 1: Results

Algorithm	Mean Running Time	Mean A* Calls
Forward A*	0.093	103.28
Backward A*	0.987	110.66
Forward A* favoring small g values	0.6435	98.52
Adaptive A*	0.23	116.04

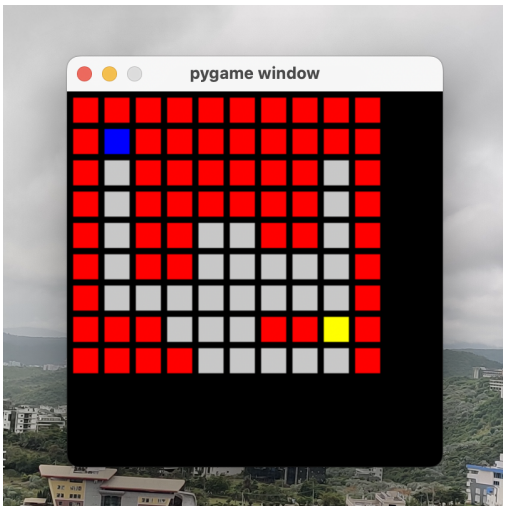

Figure 8: UI for a small grid before running the algorithm

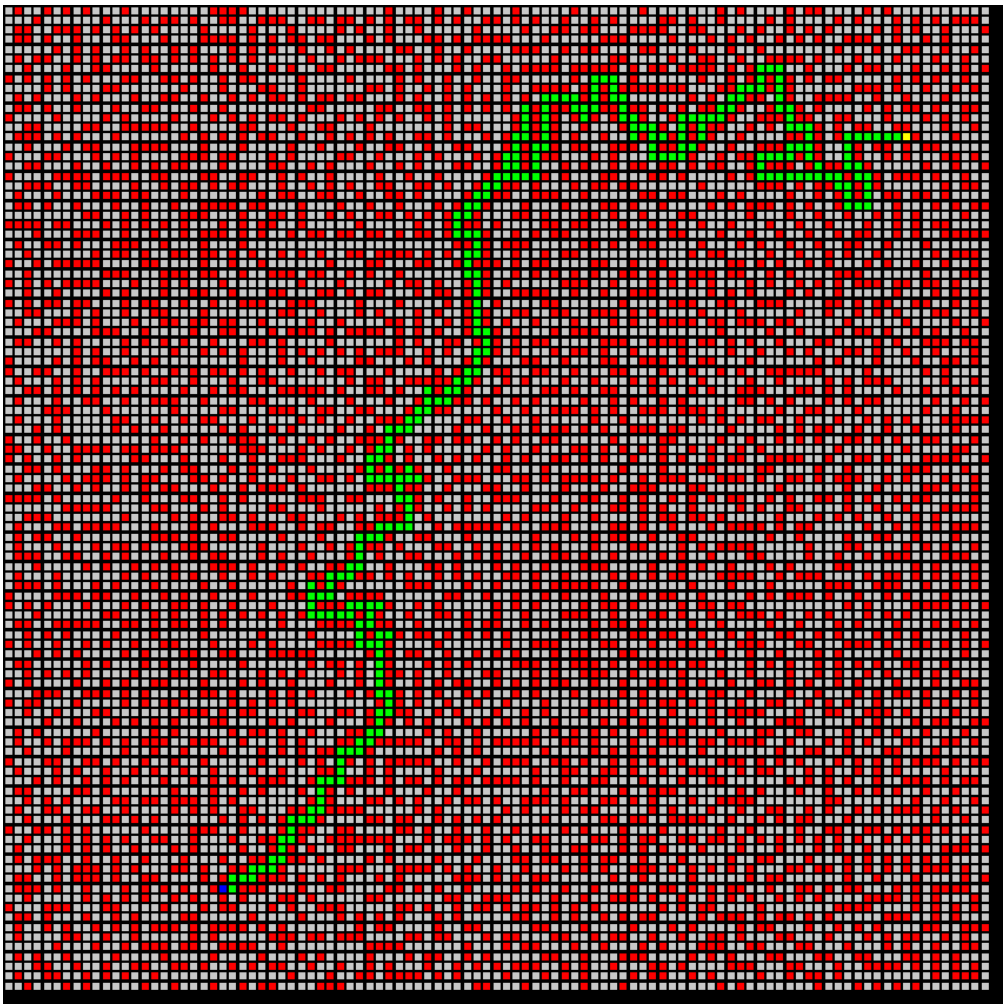

Figure 9: Found path in one maze

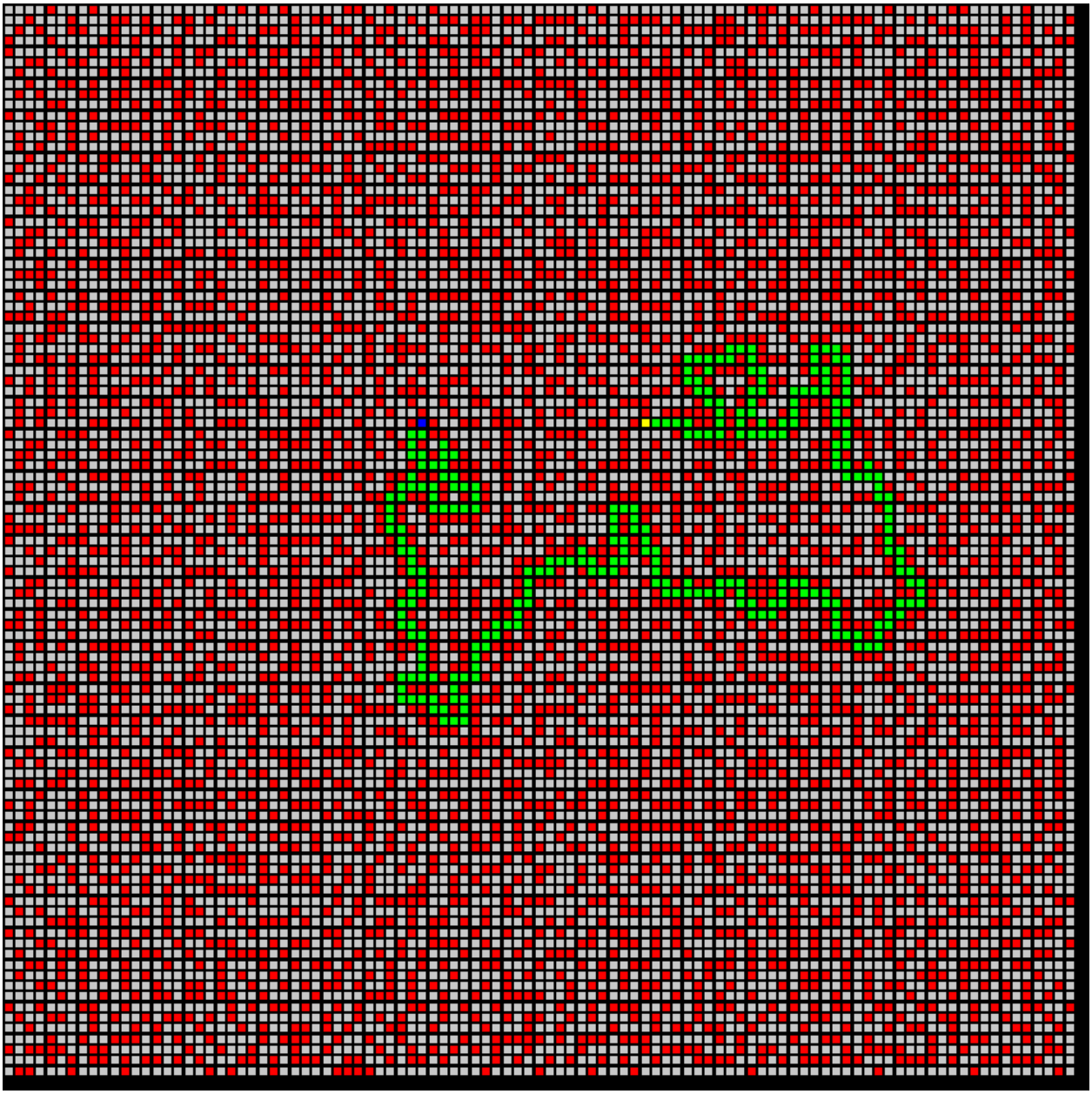

Figure 10: Found path in another maze