# Pathfinding artificial intelligence project

Programming AI to find an optimal path is a classic problem in AI. What seems intuitive to humans is less intuitive to an agent that relies on its vision and a rule-base to find the way through the maze.

## Rule base

The agent calculates the distance and finds the shortest route. Whenever we consider a new move, we must calculate which way is the shortest in the short term. We then see what is possible. The rule base consists of the following rules in this order:

1. Move straight towards the goal if it is possible
2. Move the direction that is the shortest of the remaining 2 options towards the goal (obstructed paths are excluded. We choose randomly if the options are equidistant)
3. Move the only direction we can towards the goal

If neither of the 3 vision tiles are open, we must turn find other options:

4. Turn left or right 45 degrees to consider other paths
5. Reverse the agent (if possible, without colliding)

When we consider cells we have already visited, we give them penalty so we will likely choose other paths in the future. This helps us from getting stuck. The rule base can be found in method "decideMove".

## Experiments run

I have experimented throughout the project to make the agent act intelligently. I arrived at the rule base described previously. Afterwards, I tried different levels of obstruction to identify the average, minimum, and maximum moves the robot used per environment.

I also ran experiments with random starting location and goal locations. This did, on average, lead to shorter paths, as it was more likely that the agent and the goal would start closer to each other. It was also harder to generalize and compare the results between each run due to adding randomness to the environment difficulty.
Additionally, I experimented with larger boards, but this did not significantly change the results, since the agent would very rarely move "past" the goal position to reach it.

## Experimental results

Most of the time the robot finds a solution if it exists. Table 1 below sums up the average, minimum, and maximum number of moves the agent uses per environment.

Table 1: Average, minimum, and maximum moves made by the agent for each 4 environments

| % of obstruction tiles | Average moves | Min. moves | Max. moves |
|---|---|---|---|
| **0 %** | 31 | 31 | 31 |
| **10 %** | 31.85 | 31 | 33 |
| **20 %** | 35.20 | 31 | 45 |
| **45 %** | 91.0 | 46 | 189 |

As can be read from table 1 above, there is hardly any difficulty increase for 10% and 20% obstruction tiles. Sometimes the agent gets a long path, but it is efficient on its way to the goal. When 10% of the map was obstacles there was only needed 0.85 more moves on average.

At 45% obstruction it becomes significantly harder for the agent to efficiently find a path, and this is the only environment where the agent could get stuck when there existed at least one possible path. For the sake of comparability, I discarded environments where there existed no path to the goal. There is a greater spread for minimum and maximum number of moves for 45% obstacles.

Out of the 20 attempts for the experiment in the 45% environment, the agent got stuck 3 of 20 times (15%) when there existed at least one possible path. Figure 1 below shows one such example, where the experiment was stopped after 1000 moves. The agent had repeated the same moves for hundreds of actions, but it is unclear if it would eventually reach the goal.

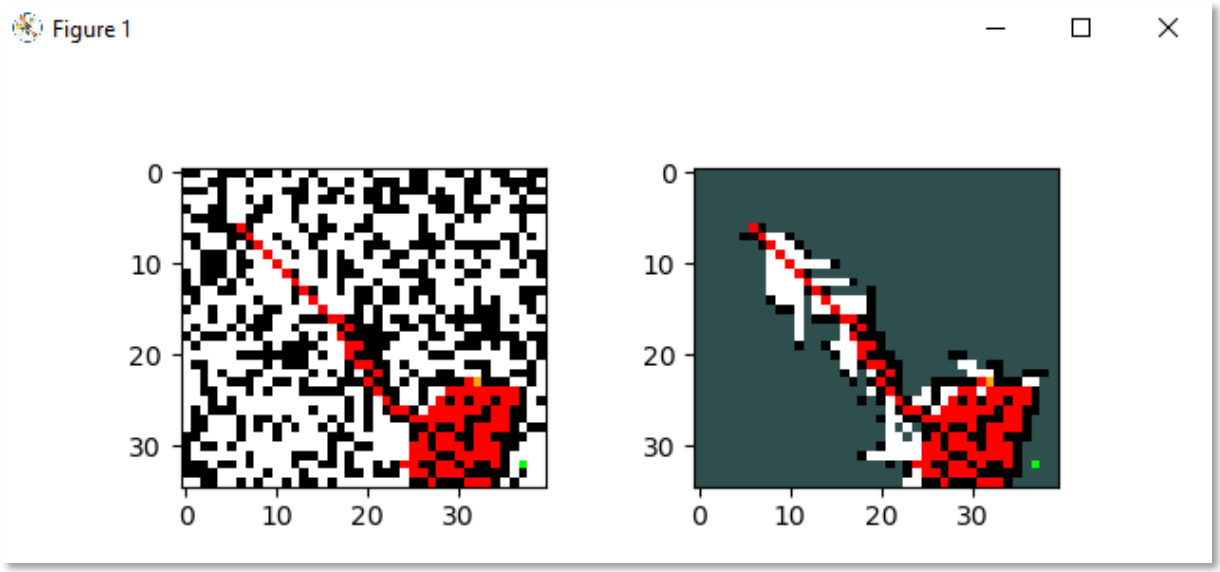
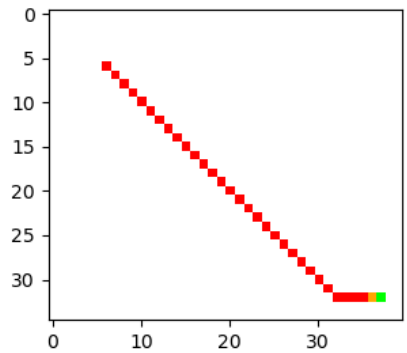
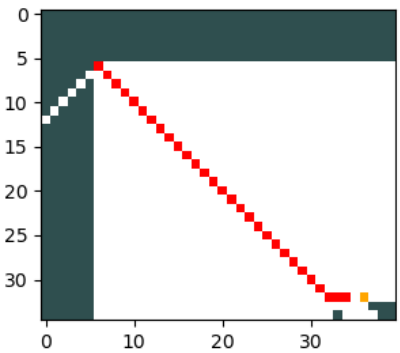
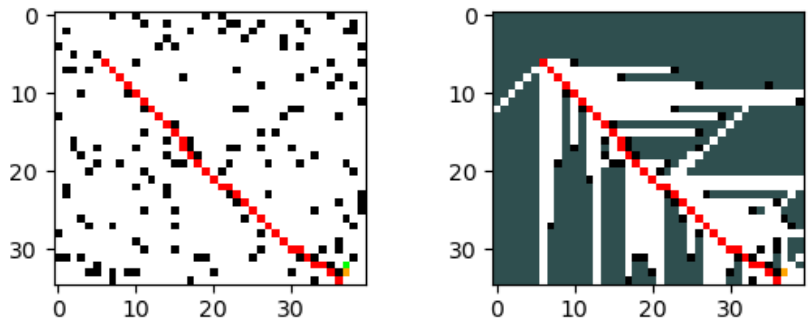
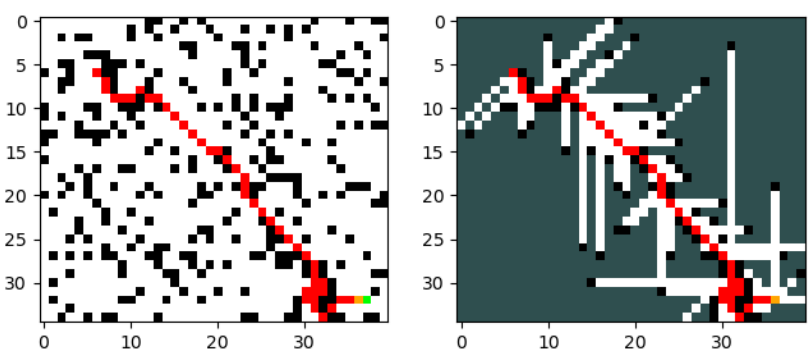
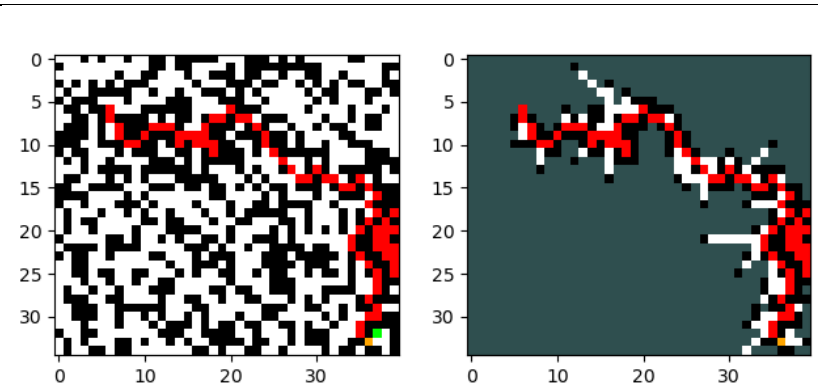



*Figure 1: The agent got stuck and the simulation was ended after 1000 moves.*

Table 2 below displays the run with the most moves for each percentage of obstacles. The path is shown as red, the unsees tiles are grey, the viewed tiles are white, the obstacles are black, the goal is green, and the agent is orange.
Successful runs with the most moves made (most difficult for the agent):

| **Table 2: Simulations with the most moves used** | | |
|---|---|---|
| **% obstacle** | **Actual map** | **What the agent has seen** |
| **0 %** |  |  |

| | |
|---|---|
| **10 %** |  |
| **20 %** |  |
| **45 %** |  |

## Discussion of results

The results indicate that a simple agent can easily find a viable path through a 2-dimensional maze. When the maze is simple (i.e., up to 20% obstacles) it finds the fastest way. As the difficulty of the maze increases, the performance drops drastically. Covering the map with 10%, 20%, and 45% of obstacles, increases the number of moves by 2.7%, 13.5%, and 194%, respectively. For 45%, the fastest route is obviously longer than for 0% obstacles, but the agent also demands more computing and calculations, and found non-optimal paths.

Especially hollow U-shapes proved to be difficult for the agent. Further rules could have been implemented to solve these problems.

## Conclusion

There is only required 5 rules to implement an agent that satisfyingly find a way through a 2-dimensional maze consisting of 0%, 10%, and 20% obstacles. With the current 5 rules, the agent struggled with mazes where 45% of the tiles were obstacles. It successfully found a path 85% of the times, but the number of moves increased significantly compared to 20% obstacles.