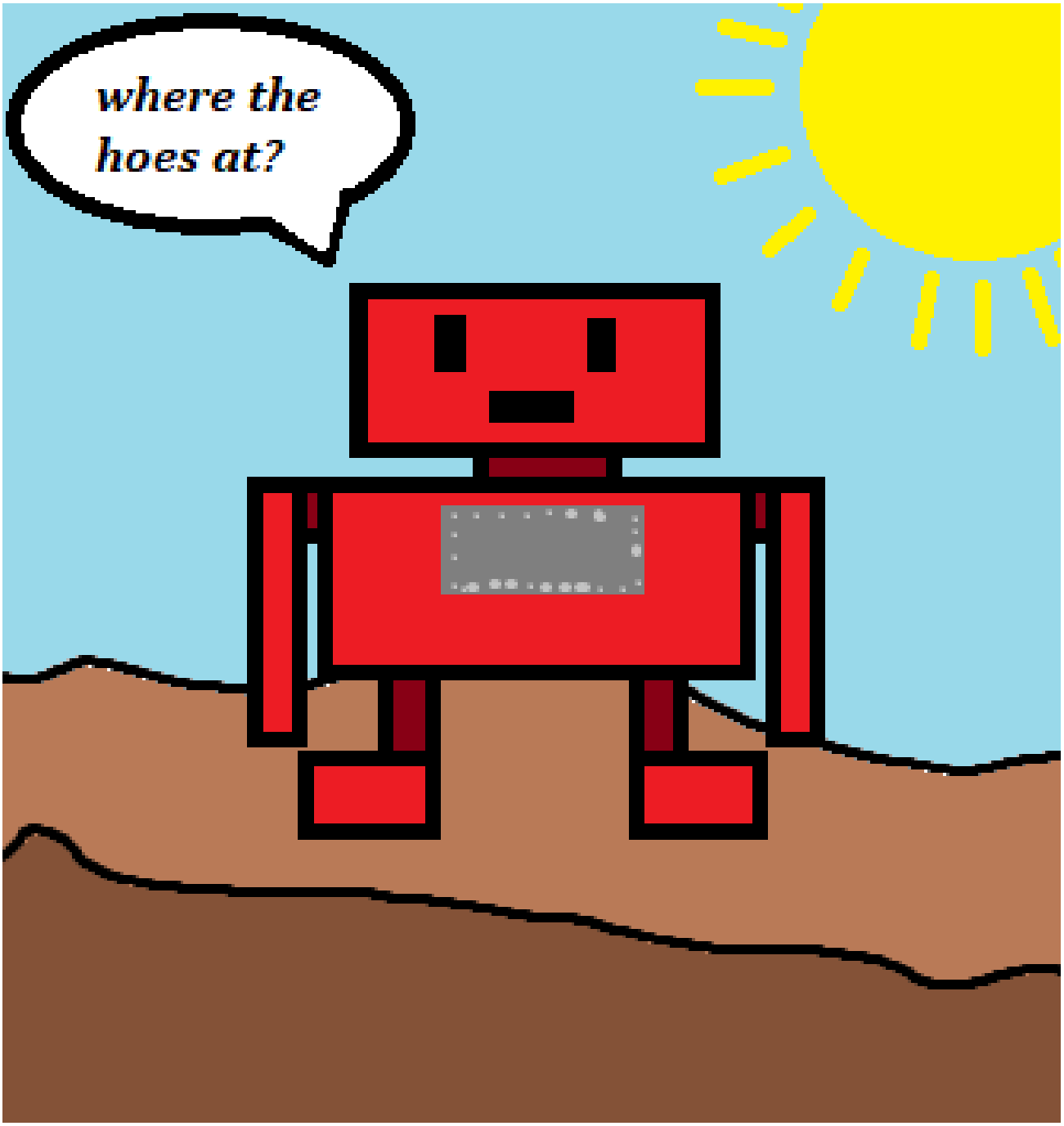



# Terrain navigation with Ai

author: Illya Reseland

date: 2024.10.1

# Intro

The task at hand was to create an Ai agent (thematically styled as a robot) that would navigate a maze (thematically styled as one of Finland's more populated areas). The map consisted of a grid of squares of varying terrain types. The Ai agents, or the robots, goal being to find the optimal path from the starting square to the goal square.

# Method and Theory

Firstly, to specify the task. An Ai agent is tasked with navigating a maze. The Ai agent will be assigned a start square and must reach the goal square. The squares differ in the cost of going through them and the optimal path is not the shortest one; it is the one that has the lowest cost. The maze is static and does not change. The Ai gets a specified number of runs through the maze so that it can learn the maze and figure out the optimal path through it. The Ai can see the square to the left, right, below, and above it. The Ai also does not initially know where it is or where the goal state is or anything about the map.

The task of having an Ai agent navigate a maze is one of the most common and widely explored ones. This means that there is a solution created by someone for damn near any variation of the problem. You have algorithms like a star and m star or breath-first search and uniform cost search making up just a few of the existing solutions.

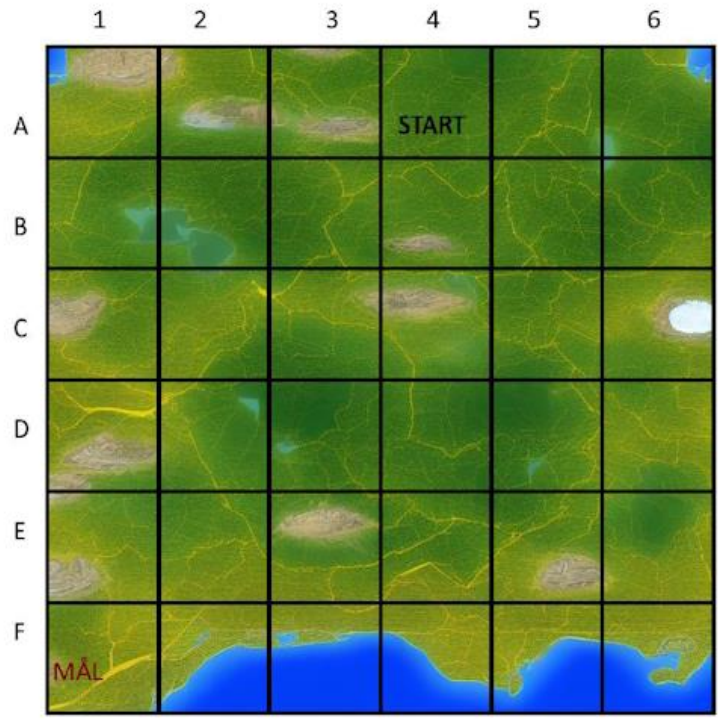
The method of choice this time around was q-learn. It is simple to implement and has the upside of letting the Ai teach itself. The downside is that the Ai is stumbling around blindly until it performs the task enough times to gather enough information about the map. Unsurprisingly the more times it is given the opportunity to run through the map the better it gets.

# Solution

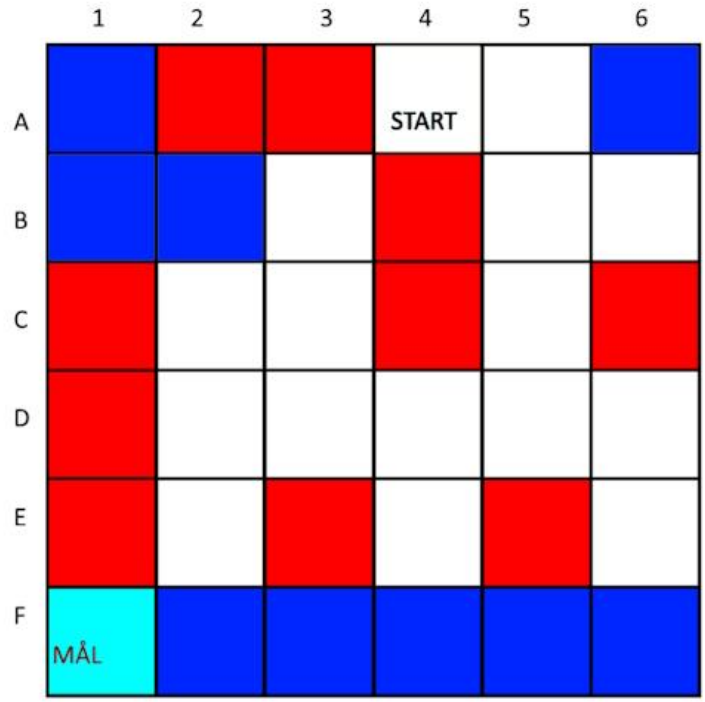
The code for the solution can be found on my github.

q-learn was used for the Ai for the reasons above.

The map used was a 6 by 6 grid:




The map can be simplified and divided into several types of terrain; water, mountains, and grass. Each one was assigned a cost with grass being the lowest followed by mountains and then by water as the most expensive:



Since there was no specified cost for the terrain types it led to the challenge of having to figure out what cost the various terrain types should have. As expected, giving different ratios led to different answers to the question of what the optimal path was. After trying a few different ones, it was settled on having grass be 1, mountains be 2 and water be 4 as that felt realistic and gave interesting answers.

To make the Ai a little more efficient it was made so that it could not revisit squares. As the task is to move from start to goal there would be no need to move over the same square twice. This stops the Ai from going back and forth and helps push it in the right direction.

It was also attempted to approximate the answer with Monte Carlo. Because of the small size of the map, it was found that given enough random attempts it would find the optimal solution. This scales quite poorly though. If the map were 100 by 100 squares or 1000 by 1000 then Monte Carlo would be of much less use and have a much harder time getting anything halfway decent.

# Results

The results were an Ai that could consistently find the optimal path given enough attempts. Given too few, the answer was less likely to be correct. It would still get a good answer; just not the best one. Because of the small size of the map, it was likely to guess what the optimal path was or at least come remarkably close to it. On top of that the ai was likely to guess the optimal path within the first few episodes. This means the results do not show the upside of using more episodes, but that would be more apparent with a larger and more complex map.

running the Ai with 10 episodes the output was:

[3, 1], [3, 2], [3, 3], [2, 3], [2, 4], [1, 4], [1, 5], [0, 5]

score: 14

running with a 100 episodes:

[3, 1], [2, 1], [2, 2], [2, 3], [1, 3], [1, 4], [0, 4], [0, 5]

score: 10

running with a 1000 episodes:

[3, 1], [2, 1], [2, 2], [2, 3], [1, 3], [1, 4], [0, 4], [0, 5]

score: 10

This represents the path the Ai took. [0, 0] being the top left and [5, 5] the bottom right. Note that it does not count the start square as the first move. The score is the sum of the cost of all the moves.

# Discussion/Conclusion

I would say the results are satisfactory and that this project nicely showcasses the capabilities of q-learn even if in a suboptimal manner. If it were to be done again a larger map would yield better results as it would present a more demanding challenge and give a better opportunity to the created Ai to prove itself. Other than that, it does a fine job demonstrating q-learn both in terms of the way it functions and the results it can produce given a task it is suited for.