Maze Runner: Solving Mazes Using Q-Learning

Submitted by

Elhusseain Aboulfetouh 202202239

Amr Khalid 202201502

Mohamed Hassan 202201257

Seif Mohamed 202201554

Course

CSAI 301 Artificial Intelligence

Table Of Contents

1. Introduction

During the Classical antiquity era, many philosophers, such as Aristotle sought to formulate a set of rules to mechanise the human thought process. That is, given a set of correct initial premises, one can mechanically generate a conclusion. In the 1940s, scientists built on those theories and assumptions to invent the first programmable computer. A few years later, the AI research field was founded. Previously, the maze runner problem (path finding problem) was solved using different AI search techniques. Those techniques included informed, uniformed, and local search algorithms. Now, reinforcement learning will be applied to solve the path finding problem. In particular, the Q-learning algorithm.

2. Problem Modeling

The Maze Runner problem is represented after discretization as a grid of M x N dimensions for the height and the width of the grid, respectively. Each cell in the grid could have a border on any of its 4 dimensions preventing the movement in the direction of the border. The runner (Problem's Agent) could be present in any of the cells in the M x N grid.

2.1 Problem's State Space Representation

Each State that the runner could be present in is represented using its row (X) and its column (Y) in a tuple pair.

$$\text{State} = (X,Y) \tag{1}$$

The Problem's State Space starts from (1,1) for the top left cell in the grid, reaching (M,N) for the bottom right cell.

2.2 Initial State

The initial State represents the starting position of the runner. Both X_O and Y_O have to belong to the State Space

$$\text{Initial State} = (X_O,Y_O) \tag{2}$$

2.3 Goal State

The Goal State represents the target position that the runner should reach by the end of the problem.

$$\text{Goal State} = (X_G, Y_G) \quad (3)$$

2.4 Actions

At each state, an agent can either go one step to east, west, north, or south. If there is a wall in the direction of the agent's movement, then the agent can not take a step towards that direction.

2.5 Problem Size Estimation

The problem size could be determined by the number of cells and the number of borders and their locations.

$$\text{Number of Cells} = M \times N \quad (4)$$

The state space is always smaller than or equal to the number of cells, this is due to the probability of having borders blocking the way to any of the cells in the grid.

$$\text{State Space Size} <= M \times N \quad (5)$$

3. Q-Learning Setup

3.1 Hyperparameters

The Q-learning algorithm contained multiple hyperparameters. First, the reward discount factor, gamma, was set to 0.9. Second, the learning rate, alpha, was set to 0.1. Third, the number of episodes was decided based on the maze dimensions. The greater the dimensions of the maze the more episodes the agent was allowed to go through. Finally, the exploration probability, epsilon, was set to 0.2.

3.2 Rewards

The agent was rewarded with 100 in case of convergence and with -1 for each step taken in the maze on the way to reach the goal. Over the training process, the cumulative reward is sought to be maximized to reach an optimal as shown later in the results' analysis the rewards reach a maximum after a certain amount of episodes.

3.3 Q-Table

The Q-Table consisted of an M x N matrix, where M is the number of rows in the grid, and N is the number of columns. For each entry in the Q-Table, a list of four values each corresponding to the Q(S, A) value of the agent being in state S and having taken action A is found. Initially, all Q(S, A) values were set to zero.

3.4 Training Process

After hyperparameters and Q-Table initialization, and starting from state (1, 1), a random between 0 and 1 was generated. If that number is less than epsilon, the agent takes a random action. Otherwise, the agent takes the action associated with the highest Q(S, A). After that, the agent lands at state S′ and receives a reward R. R is -1 if the state is not the goal state, and 100 if it is. Then Q(S, A) is updated according to the formula:

$$Q(S, A) = Q(S, A) + \text{alpha} * (R + \text{gamma} * (\text{Max}(Q(S', A)) - Q(S, A)))$$

This process is repeated until either the agent reaches the goal state or a prespecified number of steps have passed. The end of this process marks an episode. The agent goes through multiple episodes through which the Q(S, A) values converge towards the optimal values.

3.5 Performance Monitoring

The cumulative rewards were displayed after each interval of 100 episodes to monitor the performance and see if there is a gradual increase showing signs of improvement of the Q-Values for each node in the grid.

The Q-Table is displayed two times, once at the mid episode and once after the training process ends, to track the performance of the Q-Learning model. For each of them, the path reached at this point is displayed visually on the maze to see the visual difference between the path taken after half of the episodes are done and at the end of the training process showing the performance improvement.

Example on training the algorithm on a sample 10 * 10 grid over 500 episodes:

a) Before mid-episode results:

Q-Table at this stage can be accessed through this link

```
Episode 0/500, Total Reward: -664, Epsilon: 0.2
Episode 100/500, Total Reward: -70, Epsilon: 0.2
Episode 200/500, Total Reward: -28, Epsilon: 0.2
```

b) From mid-episode till the end of the training process:

Q-Table at this stage can be accessed through this link

```
Episode 300/500, Total Reward: -26, Epsilon: 0.2
Episode 400/500, Total Reward: -30, Epsilon: 0.2
```

3.6 Performance Evaluation

To assess the learning performance of the agent, The algorithm was applied to three grids with varying sizes, a 30x30 grid, a 45x45 grid, and a 60x60 grid. For each grid, a suitable amount of episodes was set. For the 30x30 grid, the total reward improved the more episodes the agent went through. For example, -1000 in the first episode, -164 at episode 1000, and -82 at episode 2000. For the 45x45 grid, at episode 1 the total reward was -1000, at episode 1000 was -998, at episode 5000 was -218, and at episode 9000 was -122. Finally, for the 60x60 grid, at episode 1 the total reward was -1000, at episode 10000 was -180, and at episode 25000 was -176. Based on these results, it can be concluded that the learning performance is reasonably good. Moreover, based on the visual results, the policy constructed is almost always optimal.