A REPORT

ON

CS F317 Reinforcement Learning: Assignment 1

by

Name of the student	**ID number**
Dev Gala	2021A7PS0182H

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
(Hyderabad Campus)
September, 2024

CONTENTS

1 Introduction

Reinforcement learning is a goal directed learning approach where a learner within an environment decides what actions to take in order to achieve said goal. For each action, the learner receives a reward from the environment. Reinforcement learning is the process of creating a policy i.e deciding a trajectory of actions such that the reward gained by the learner is maximized.

Reinforcement learning problems are modeled using the *Agent Environment Interface.* In this, at each time step, agent(learner) receives a representation of the environment as a state $s_t \in S$, where S is set of all possible states that describe the environment. An agent selects action $A_t \in A(s_t)$, where $A(s_t)$ is set of actions available for that state. In the next time step, the agent receives a reward R_{t+1} as a consequence of A_t

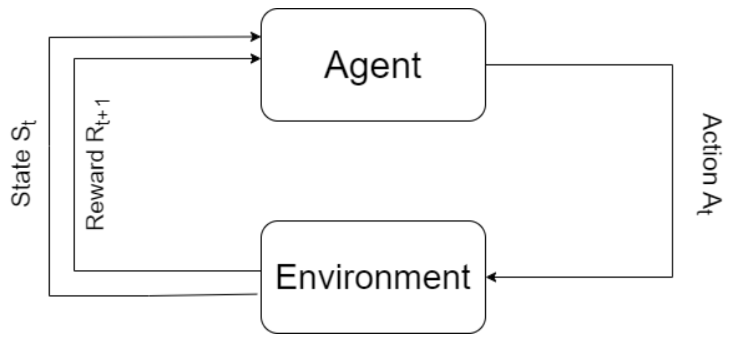


Figure 1: Agent Environment Interface

Many real world problems can be solved to a satisfying degree using reinforcement learning. Some examples include: Chess bots, Robot navigation, creating trading strategies, driver-less automobiles, etc. The code for this entire assignment is available here: https://github.com/devgala/RL-Assignment1

2 Concepts

2.1 Markov Property and Markov Decision Process

A state signal is said to be *Markov* or *to have Markov Property* if it succeeds to retain all relevant information for the agent.

In general, the response from the environment depends on everything that has happened before. Thus it can be defined using the following probability distribution:

$$Pr\left(\frac{R_{t+1} = r, s_{t+1} = s'}{(s_0, A_0, R_1), (s_1, A_1, R_2), ..., (s_{t-1}, A_{t-1}, R_t), s_t, A_t}\right)$$

If state signal was Markov in nature, then environment's response at $t + 1$ can only depend on state and action at t as all the information needed by the agent will be contained in the state signal s_{t+1}. Thus the dynamics can be defined by the probability distribution:

$$p\left(\frac{s', r}{s, a}\right) = Pr\left(\frac{R_{t+1} = r, s_{t+1} = s'}{s_t = s, A_t = a}\right)$$

A *Markov Decision Process* is any RL task that satisfies Markov Property.

2.2 Monte Carlo Methods

To solve RL problems, is to compute an optimal policy that maximizes reward gained. Some methods to solve RL problems require complete knowledge of the environment. This can prove to be disadvantageous in situations where the environment keeps changing or knowledge of the environment is unavailable.

Monte Carlo methods require only experience i.e. actual/simulated interaction with the environment to solve RL problems. It does so by averaging sample returns for state-action pairs or state values. The trade-off here is that Monte Carlo methods can only be used for episodic tasks where each episode has a terminal state followed by a reset.

Thus Monte Carlo methods involve simulating infinite episodes from policy π and averaging the returns for each state-action pair (s, a). The problem arises when π is a deterministic policy, resulting in many state-action pairs to remain un-visited. Thus MC estimates do not improve with experience. To solve this we introduce a small non-determinism in the policy π. For a small probability $\epsilon > 0$ we perform a random action from the action space of s_t. These are called $\epsilon-$soft policies.

2.2.1 On-policy MC methods

Here, we learn the state-action function by sampling episodes from the policy we want to optimize. We start with an $\epsilon-$soft policy and move it towards a deterministic policy.

2.2.2 Off-policy MC methods

Here, we learn state-action function of policy π by sampling episodes from another policy μ. This requires the assumption of coverage that actions in π are occasionally taken in μ. The policy we want to learn is deterministic and is called target policy whereas the policy used for sampling is called behavior policy. To conduct off policy MC control, we use importance sampling i.e estimate expected values of one distribution using another distribution. The importance sampling ratio gives relative probability of trajectories occurring under target and behaviour policies. For trajectory; $s_t, A_t, s_{t+1}, A_{t+1}, ..., A_{T-1}, s_T$, the ISR is given as:

$$\rho_t^T = \prod_{k=t}^{T}\left(\frac{\pi\left(A_k|s_k\right)}{\mu\left(A_k|s_k\right)}\right)$$

3 Grid Problem : Max Treasure Maze

Max Treasure Maze is a grid problem where a maze (represented as an $m \times m$ grid) is filled with n treasures. The player is required to exit the maze with as many treasures as they can find. The grid has treasures placed at location unknown to the player. At each point in the grid the player can make 4 moves : Up, Down, Left, Right. The player's goal is to exit the maze after finding maximum number of treasures. This is an interesting take on the classical maze and finding treasure problem, as the player is required to do both tasks in this which shows an interesting pattern in MC learning methods. As we will see in further sections, it shows the greedy nature of RL learning algorithms. Multiple grids have been designed to show how the algorithm performs in finding the optimal policy. Below are the grids that we're used in this report.

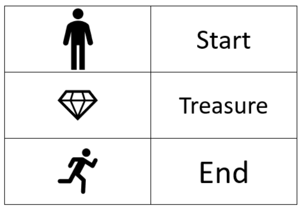


Figure 2: Maze Legend

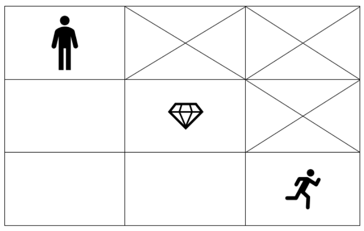


Figure 3: 3×3 maze

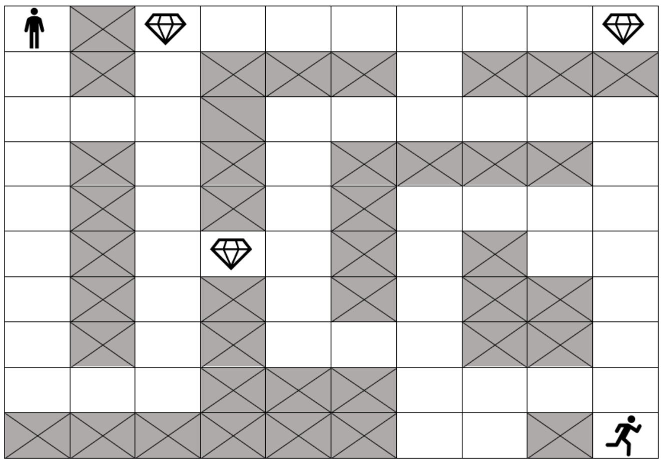


Figure 4: 10×10 maze

4 Modeling to Reinforcement Learning

We use the agent-environment interface to model this problem for Reinforcement Learning. The states for this problem are defined by the 3-tuple $(i, j, k) \in R^3$ where i represents the row number, j represents the column number and k represents the number of treasures found by the agent. The reason for this will be clear when we evaluate the results from our experiment. The action space for each state is the same with 4 actions: *Up, Down, Left, Right*. The design of the reward function is to "motivate" the agent to find all the treasures before exiting the maze. Since the reward at the end of the maze depends on how many treasures the agent has collected, the more treasures it collects, the higher its reward for exiting the maze. Each treasure itself has a high reward to draw the agent toward it. Thus the RL model is given by:

$$S = \{(i, j, k) | (i, j, k) \in R^3 \ \wedge \ 1 \le i, j \le m \ \wedge \ 0 \le k \le n\}$$

$$A(s) = \{Up,\ Down,\ Left,\ Right\} \forall s \in S$$

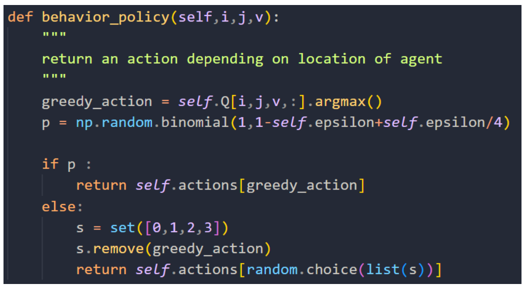
$$R_t(i, j, k) = \begin{cases} 1 & (i, j) \in treasureLocation \\ 2^k & (i, j) = (m, m) \\ 0 & otherwise \end{cases}$$

The reason for selecting 2^k for terminal state reward is to drive the model towards finding more and more treasures. If the reward was ≤ 1, the model would only look for the treasures and not move towards termination.

5 Solving using Monte Carlo Methods

To solve an RL problem using Monte Carlo methods involves generating episodes using a policy, then using the experiences from those episodes to evaluate and improve the policy. This report evaluates the given RL model using on-policy and off-policy Monte Carlo methods. We use the following general algorithm [1] for both on-policy and off-policy with changes in the ISR procedures and behaviour policy.

For on-policy MC control, target and behavior policies are the same, thus $\pi = \mu$ and $W = 1$ as $ISR = 1$. The policy for on-policy MC control is an ϵ-soft policy. For $p = 1 - \epsilon + \frac{\epsilon}{A(s_t)}$, agent takes greedy action, and random action for probability $1 - p$. Below is the code snippet for this type of policy. Here $A(s_t) = 0.25$



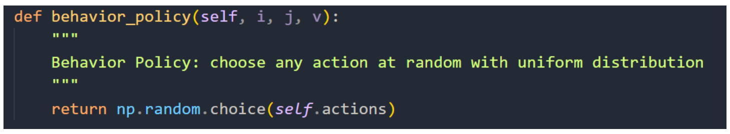
```
def behavior_policy(self,i,j,v):
    """
    return an action depending on location of agent
    """
    greedy_action = self.Q[i,j,v,:].argmax()
    p = np.random.binomial(1,1-self.epsilon+self.epsilon/4)

    if p :
        return self.actions[greedy_action]
    else:
        s = set([0,1,2,3])
        s.remove(greedy_action)
        return self.actions[random.choice(list(s))]
```

Figure 5: On Policy Monte Carlo

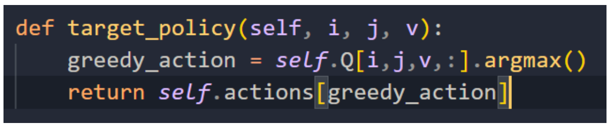
For off-policy MC control, we chose a deterministic target policy π and a more exploratory behavior policy μ. The target policy is greedy and chooses the action with maximum value for a given state in Q table. The behavior policy is more exploratory and chooses any action at random with equal probability. The ISR value for off-policy MC is either 4 when behavior policy and target policy take same action, otherwise 0.

$$\rho_t^t = \begin{cases} 4 & \pi(s_t) = \mu(s_t) \\ 0 & otherwise \end{cases}$$



```python
def behavior_policy(self, i, j, v):
    """
    Behavior Policy: choose any action at random with uniform distribution
    """
    return np.random.choice(self.actions)
```

Figure 6: Off-policy MC control behavior policy



```python
def target_policy(self, i, j, v):
    greedy_action = self.Q[i,j,v,:].argmax()
    return self.actions[greedy_action]
```

Figure 7: Off-policy MC control target policy

Algorithm 1 MC_CONTROL

```
1: $Q(s,a) \leftarrow 0$
2: $C(s,a) \leftarrow 0$
3: $\mu(s,a) \leftarrow$ arbitrary soft-behavior policy
4: $\pi(s,a) \leftarrow$ arbitrary target policy
5: while True do
6:     Generate Episode using $\mu : s_0, A_0, R_1, ..., A_{T-1}, R_T, s_T$
7:     $G \leftarrow 0$
8:     $W \leftarrow 1$
9:     for $t = T-1, T-2, ..., 0$ do
10:        $G \leftarrow G \times \gamma + R_{t+1}$
11:        $C(s_t, A_t) \leftarrow C(s_t, A_t) + W$
12:        $Q(s_t, A_t) \leftarrow Q(s_t, A_t) + \frac{W}{C(s_t,A_t)}(G - Q(s_t, A_t))$
13:        $W \leftarrow W \times \frac{\pi(A_t|s_t)}{\mu(A_t|s_t)}$
14:        if W=0 then
15:            exit for loop
16:        end if
17:    end for
18: end while
```

6 Results

6.1 On Policy Monte Carlo Control

Using on-policy methods for 3x3 maze [3] resulted in the following heat-maps:

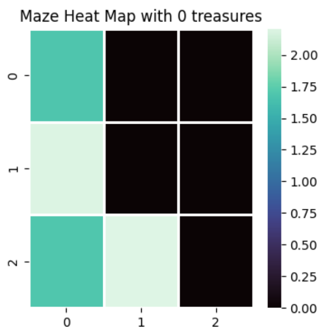


Figure 8: Agent had 0 treasures

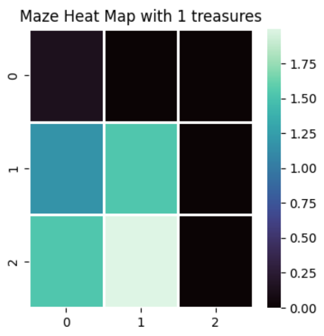


Figure 9: Agent had 1 treasure

Each value of the heat-map shows the maximum value action at that state. Thus by looking at the heat-maps alone do not make much sense as we do not know what action agent has taken to yield the most returns. Overlaying this heat map with an *action-map* i.e a map of actions with max value for that state makes it clear how well the method has worked for solving this grid problem. After overlaying the heat-maps and the action

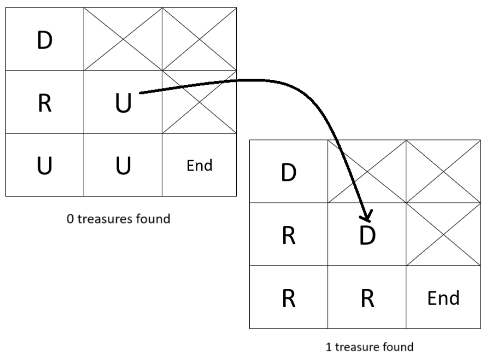


Figure 10: 3x3 maze action map

maps, when agent has 0 treasures, its goal is to look for the 1st treasure. Once its found that, agent moves to the next treasure. After it has collected all treasures, agent moves to the exit of the maze.Similarly we can see this for the 10x10 grid [4] as well. This behavior is only possible when the number of treasures collected is also a part of the state. If this was not a part of the state, the agent will update the older values of Q, resulting in the wrong action being taken. For example, at (0,5) the agent moves right towards the 1^{st} treasure. After reaching (0,6) the next treasure is at (1,5), so the agent will update the action at (0,5) to be *Down* from the original action of *Right*. Thus in the final policy, the agent will never reach the treasure at (0,6).

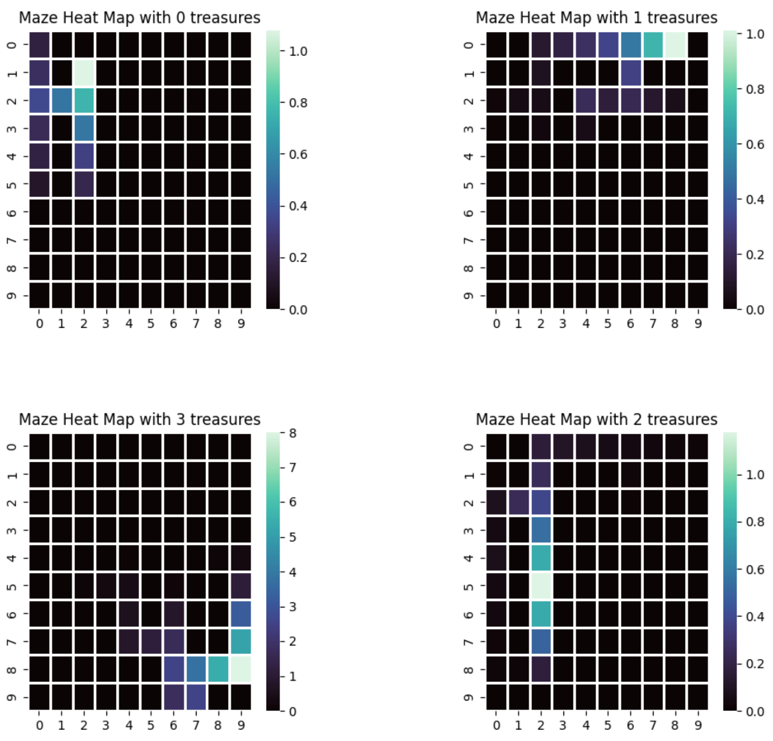


Figure 11: 10x10 maze heat map

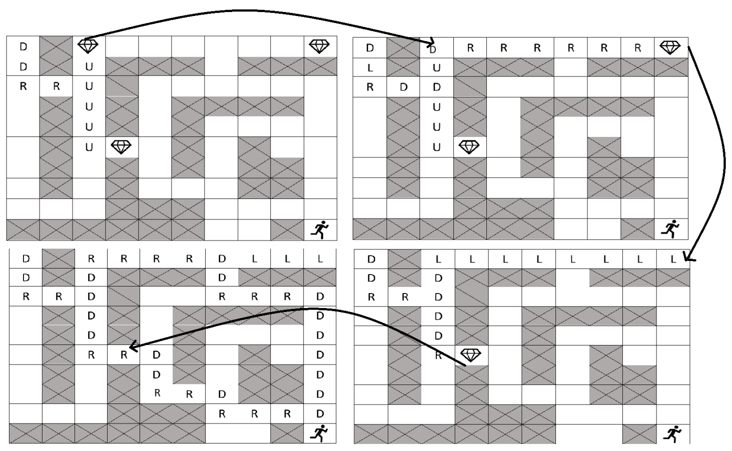


Figure 12: 10x10 Maze action map

The below figure shows the Q-value for each treasure throughout the iterations for 10x10 maze. It reveals how the algorithm moves the policy towards each of the treasures. The blue represents the first treasure, orange for the second and green for the third.

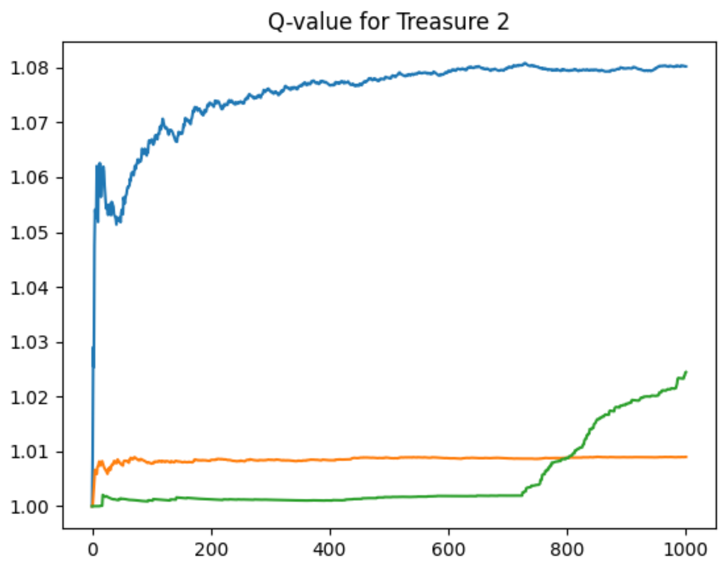
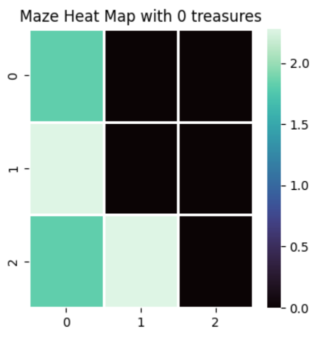


Figure 13: Q values v/s number of iterations

The results we have observed show that on-policy method was quite effective in converging to an optimal policy. It also collected all treasuries before exiting the maze which also shows the algorithm's greedy nature of moving towards the maximum return possible.

6.2 Off-policy Monte Carlo Control

Using off-policy methods for 3x3 maze [3]. The action maps reveal that even after the agent has collected the reward, it moves towards the start due to its behavior policy and makes a way from (0,0) to the exit.



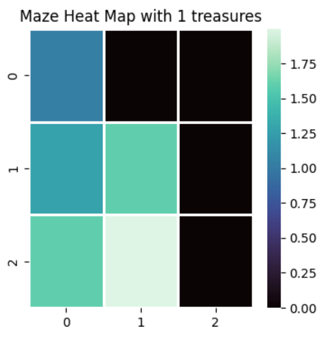


Figure 14: Agent had 0 treasures

Figure 15: Agent had 1 treasure

For 10x10 maze, for each different number of treasures collected, the heat map seems the

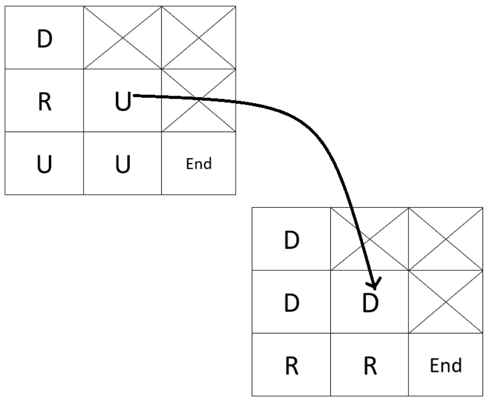


Figure 16: 3x3 Maze action map off-policy

same with higher Q values towards the exit and 0 i.e no change from original array. Even after multiple runs of MC_CONTROL with 10^9 iterations, the results were the same. The action maps, reveal that the algorithm is moving more towards the exit without bothering to collect the treasures. This is attributed to the randomness of the behavior policy. Since behavior policy chooses actions at random, it might reach the end of the maze before it gets a chance to reach any treasure. This in turn contributes to the next episodes following the same path of exiting the maze and ignoring the treasures. Since the goal of the problem is to exit the maze, the algorithm has not given a wrong solution.

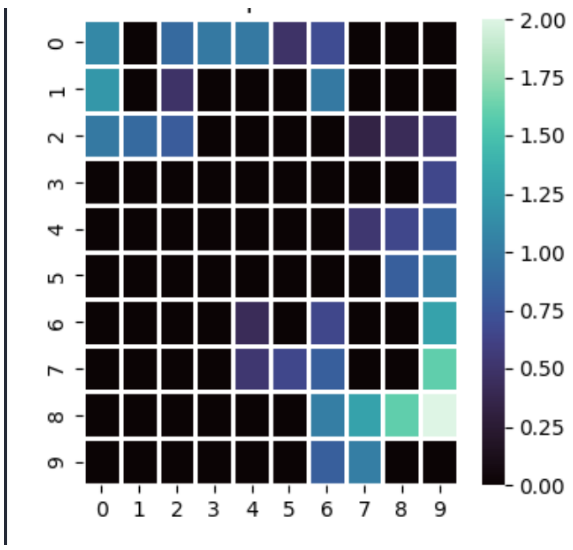


Figure 17: Off policy heatmap 10x10 maze

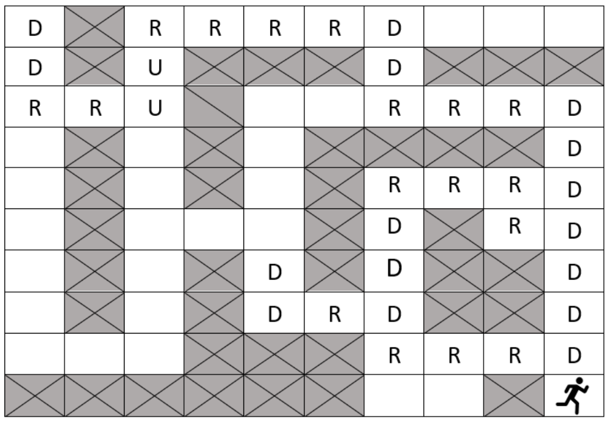


Figure 18: Off policy action map 10x10 maze

7 Comparison of on-policy and off-policy MC control

After going through the results, it seems like the on-policy methods are clearly better as they converge to a policy that collects all treasures and exits the maze. For these experiments, the ϵ value was kept at 0.4 to complete the experiments faster and due to limited compute resources. Even though this affects the experiment, in experiments with higher ϵ values, the final heat-maps showed little change. The more *rigid* nature of on-policy methods makes it ideal for such problems as there is both exploitative and exploratory component to the algorithm. The exploratory nature finds the treasures and the exploitative nature moves the policy towards the path finding the treasures. There is a certain disadvantage to this method as well. Since the exploitative nature is *stronger* than the exploratory nature, the path becomes fixed. Even though the algorithm solves the problem, it gets stuck in the same path without exploring more. This in some cases like when the treasure is farther away than the exit, may lead to the algorithm ignoring that treasure. On policy method is also very compute heavy as it first needs to find an exit which is very tough on a weak exploratory policy leading to longer time for satisfactory results.

Off policy methods did not yield satisfactory results in the experiments. A highly exploratory policy has a chance of not running into a treasure at all. After reaching the end of the maze, the new path was cemented into target policy and thus the algorithm could not change it. This is one of the disadvantages of off policy methods. Since the target policy is completely deterministic, it resists change to itself even with many iterations over the behavior policy. Such grid problems therefore are better left to on-policy methods with both exploration and exploitation.