Reinforcement Learning Assignment 1

Karthik Prakash 2022A7PS0022H

Kishan Abijay P 2022A7PS0037H

1 Introduction

In this report, we explore an interesting reinforcement learning problem set in a 3D grid game environment. The game involves a series of stacked grids, where the player begins at the bottom-left corner of the lowest grid and aims to reach the upper-right corner of any grid. The challenge lies in the player deciding between moving up to a higher grid, which provides a greater reward but also involves more disturbance, or moving down to a lower grid with fewer disturbances but a smaller reward.

We solve this problem using both on-policy and off-policy Monte Carlo methods, comparing both the approaches in terms of performance and efficiency. On-policy Monte Carlo methods will ask the player to use the same strategy it is learning, refining the policy based on its experiences. Off-policy Monte Carlo methods will allow the player to learn from different strategies, exploring a wider range of actions and outcomes. Using the results, we illustrate the benefits and limitations of each approach.

2 Game Description

We have h square grids with dimensions $n \times n$ that are stacked on top of each other. Each square can be represented as (x, y, z) where $1 \leq z \leq h$ represents the grid number and $1 \leq x, y, \leq n$ represents the cell's position on that grid.

The player starts at $(1, 1, 1)$, which is the bottom-left corner on the lowermost grid. The goal of the game is to reach (n, n, k) for some $1 \leq k \leq h$, which is the upper-right corner of any grid. Reaching the upper-right corner of the k-th grid gives you a score of k points and ends the game.

The player can move up, left, down or right which moves them in 2D within the current grid. They can also go to the next higher grid or next lower grid from their current position. This means that all moves can be represented as $(x, y, z) \rightarrow (x \pm 1, y \pm 1, z \pm 1)$, provided that the final state is within bounds.

However, after a player has made a move, there is a chance of them being randomly pushed away from the direction of the goal. The strength and direction of this push are random but higher grids have a higher probability of being pushed away further in comparison to lower grids.

After $n^2 \cdot h$ moves, the game ends automatically (even if the player did not reach any of the goal states). Fundamentally, the game involves deciding between choosing an "easy" grid with less "turbulence" to get a lower reward and choosing a "hard" grid with more "turbulence" to get a higher reward.

Internally, the turbulence for each level is modelled with the help of a radially symmetric 2-D Gaussian distribution centered at the origin:

$$f(x, y) = \frac{1}{2\pi s^2} \cdot exp(-\frac{x^2 + y^2}{2s^2})$$

where $s = \sigma_x = \sigma_y = \frac{i}{h}$ for the i-th grid.

After each of the player's moves, we sample a point (h, k) from this distribution and modify the signs of both coordinates to make them both negative ($x \to -|x|$). We then truncate the coordinates to get integers (h', k') and add this to the player's position:

$$(x, y, z) \to (x + h', y + k', z)$$

If this results in an invalid state (negative coordinates), we change the position to the nearest valid state. Note that the turbulence depends only on the grid number (but not the player's position in a specific grid) and always pushes them away from the goal (due to the sign modifications).

3 Approach

Since the player only gets a fixed amount of moves, the reward function should have a penalty for each move made. However, we also want to encourage the agent to explore the higher states (due to the higher rewards lying at the goal). With this in mind, we define:

$$R_{move} = -\frac{1}{z}$$

$$R_{goal} = n^2 \cdot z$$

where z is the grid number of the agent's current position.

Every time the agent makes a move, it gets a reward of R_{move}. If the agent reaches one of the goal states at the upper-right of any grid, it gets a reward of R_{goal}. Note that this means the player gets punished less for moving within higher grids compared to moving within lower grids.

3.1 On-Policy Monte Carlo

On-policy methods focus on improving the same policy the agent uses to interact with the environment. The agent uses the current policy π to decide actions during each episode, and updates this policy based on the experiences gained from the episodes. We use an on-policy Monte Carlo control algorithm that uses generalised policy iteration (GPI) along with policy updating using an ϵ-greedy strategy

As the agent progresses, it keeps track of the total rewards it collects. This helps the agent understand how good or bad its actions were. After finishing an episode, the agent uses the rewards it collected to update its Q-values. If an action led to high rewards, its Q-value increases. Once the Q-values are updated, the policy is adjusted.

The agent favors actions that have the highest Q-values, meaning it will choose actions that are most likely to lead to the best outcomes. The policy is updated using an ϵ-greedy strategy to balance exploration and exploitation during learning:

$$\pi = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & a = a^* \\ \frac{\epsilon}{|A|} & a \neq a^* \end{cases}$$

Here, first-visit Monte Carlo is being used, which means for each state-action pair, the policy is updated only if the pair hasn't been seen before in that episode. After simulating for a given number of episodes, the learned Q-values and the improved policy is returned.

3.2 Off-Policy Monte Carlo

Off-policy methods generally use two policies: a behaviour policy that is generally more exploratory and a target policy, which is the optimal policy being learned. While they are generally slower to converge and have higher variance when compared to on-policy methods, they are more powerful and can be modified in many ways.

We use a off-policy Monte Carlo control algorithm that uses generalised policy iteration (GPI) and weighted importance sampling. For the behaviour policy, we decided to use an ϵ-greedy policy, which allows us to control its exploratory nature.

Weighted importance sampling is used to estimate values from the target policy given various samples (episodes) generated from the behaviour policy. After generating an episode using the behaviour policy, we iterate over it in reverse and update the reward and action values before using those to improve the target policy.

4 Results

For the game parameters, we used $n = 10$ and $h = 10$. We used a discount factor of $\gamma = 1$ and $\epsilon = 0.1$.

For on-policy Monte Carlo, we evaluated the policy after every 100 training episodes by running it for 30 iterations and taking the mean total reward. For doing this for 2000 training episodes, we plotted the mean reward:

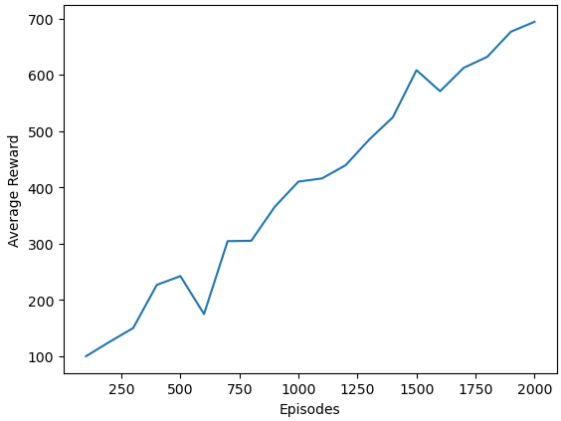

Figure 1: Optimal policy from on-policy Monte Carlo

We then evaluated the final optimal policies produced by both methods after 10000 episodes. On-policy gave an average reward of 973.078 with a standard deviation of 18.501 while off-policy gave an average reward of 972.564 with a standard deviation of 21.313.

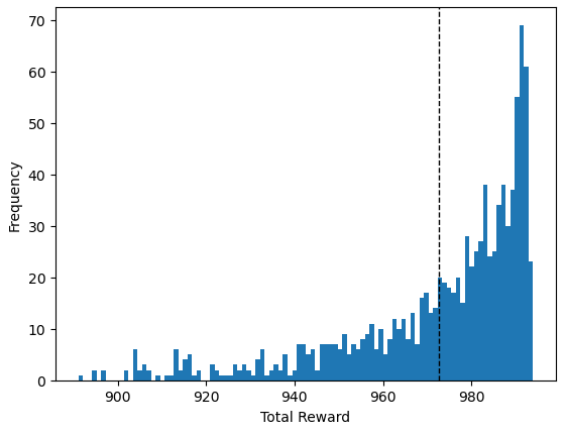

Figure 2: Histogram of the total reward accumulated in off-policy Monte Carlo

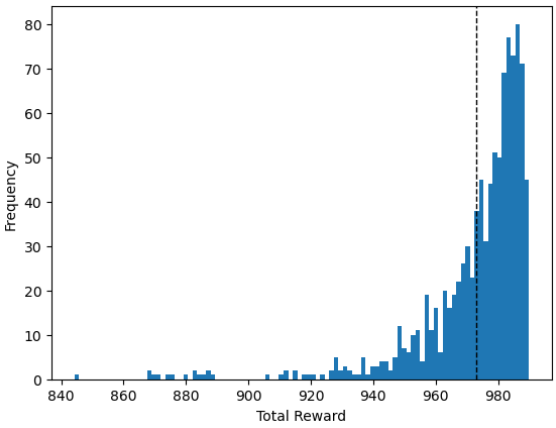

Figure 3: Histogram of the total reward accumulated in on-policy Monte Carlo

The graphs illustrate that even though the mean reward from both methods is quite similar, off-policy gives a noticeably higher variance in final reward, which is to be expected.

For reference, here is the optimal policy produced by off-policy Monte Carlo (U represents going up to the next higher grid while D represents going down to the next lower grid).

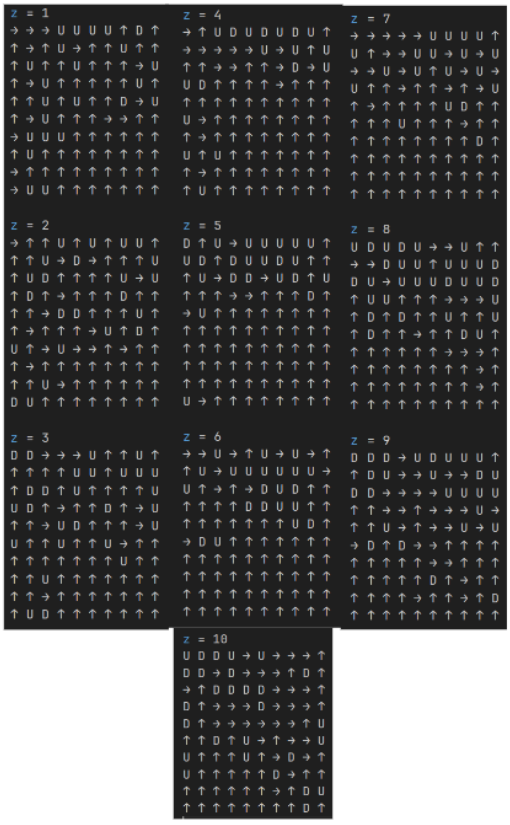

Figure 4: Optimal policy from off-policy Monte Carlo

5 Conclusion

In this report, we analyzed the performance of on-policy and off-policy Monte Carlo methods for solving a complex grid navigation problem with turbulent conditions. Our findings reveal differences in the efficiency and stability of these approaches.

On-policy Monte Carlo methods had a faster convergence to an optimal policy and had quicker execution times. Due to a more stable learning process, a lower standard deviation of the rewards and policies learned is observed. This makes on-policy methods advantageous in environments where rapid adaptation and execution are crucial. The disadvantages are that it does limited exploration and there is a danger of potential inefficient use of samples.

Off-policy Monte Carlo methods, with more exploration of different strategies, showed slower convergence and higher variance in the learned policies. This approach allowed the agent to learn from a wider range of experiences, but the increased variance and longer learning times indicate it is relatively complex to integrate and optimise. But it also provides more flexibility in learning from different policies and exploration of diverse strategies which is helpful in complex environments.

Overall, the choice between on-policy and off-policy methods depends on the specific requirements of the problem. For scenarios demanding quick convergence and stable performance, on-policy methods are preferable. However, when exploring a diverse range of strategies is essential, possibly due to the presence of a complex environment, and there is more tolerance for higher variance, off-policy methods offer more valuable insights.

To improve our results further, we could experiment more with different ϵ and γ values and look at more sophisticated methods like discount-aware importance sampling. We could also try out other reward models (for example, incorporating the Manhattan distance to the goal to encourage moving towards it) or even more sophisticated reinforcement learning algorithms like Q-learning and SARSA.