Queen's UNIVERSITY

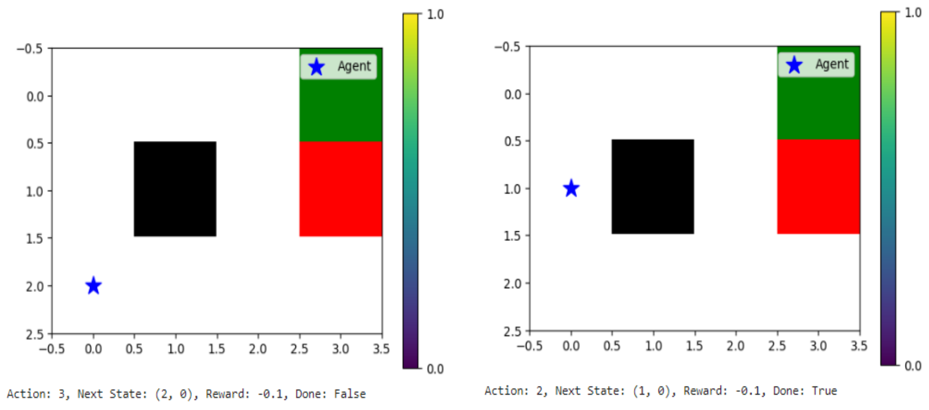
Comprehensive Analysis of Grid world Environment Using Monte Carlo methods

Presented to professor: **Sidney Givigi**

By: **Amr Mostafa Ibrahim**

I. Introduction

This report presents a comparative analysis of four Monte Carlo methods applied to a Grid world environment. The agent has four possible actions: Up, Right, Left, and Down. Each step the agent takes, it receives a small negative reward of -0.1. If the agent reaches the goal state, it receives a positive reward of +1. However, there is also a danger state that the agent must avoid, as entering this state results in a negative reward of -1.



```
Action: 3, Next State: (2, 0), Reward: -0.1, Done: False
```

```
Action: 2, Next State: (1, 0), Reward: -0.1, Done: True
```

(A) (B)

Fig.1 The grid world environment. **A**) Initial state for agent (2, 0) ". **B**) After 30 interactions, agent reached state (1, 0), received reward (-0.1) and episode terminated.

The goal of this report is to compare the performance of four Monte Carlo algorithms in this Grid world environment. The algorithms will be evaluated based on the following criteria:

- Cumulative Rewards
- Total Samples to Reach Target Reward
- Training Time
- State Values (V(s))
- Policy (π(s))

Additionally, the algorithms will be tested in both deterministic and stochastic environments (Noise= 0.4) , and the impact of varying exploration parameter (epsilon) will be analyzed.

By conducting a thorough comparison of these Monte Carlo algorithms, the report aims to provide insights into their strengths, weaknesses, and suitability for solving the Grid world task.

The remainder of this report is structured as follows: **Section II** discussed algorithms that we implemented Section **III** illustrate criteria that we used in our comparison between different algorithms **Section IV** mention results and discussed it with stochastic and deterministic environment and show the effect of changing parameters as gamma and epsilon Finally, **Section V** concludes the report with a discussion on findings.

II. Algorithms

The following four Monte Carlo algorithms were selected for comparison in the Grid world environment:

Monte Carlo With Exploring Starts (**On-policy**):

It is an algorithm in reinforcement learning where the agent's policy is continuously improved based on actual returns observed after executing episodes. Exploring starts ensuring that every state-action pair is visited enough times by starting episodes in every possible state-action pair at least once. This guarantees that the policy improves towards the optimal policy because all possibilities are explored. The policy is updated after each episode using the average return of the observed returns, ensuring convergence to the optimal policy under the assumption that all state-action pairs are explored sufficiently.

Monte Carlo Without Exploring Starts (**On-policy**):

It differs from the exploring starts approach by relying on a method such as ε-greedy to ensure that all state-action pairs are sufficiently explored. In this approach, the agent follows a behavior policy that is typically an ε-greedy version of the current policy, which occasionally takes random actions to ensure exploration. This method incrementally improves the policy by using observed returns to update the action-value function, which is then used to refine the policy. Despite the lack of explicit exploring starts, the ε-greedy strategy ensures that the algorithm converges to the optimal policy by balancing exploration and exploitation.

Monte Carlo Prediction (Off-policy):

It is used to estimate the value function of a target policy different from the behavior policy used to generate the episodes. This method relies on important sampling to correct for the difference between the target and behavior policies. The algorithm observes episodes generated by the behavior policy, then reweights the returns using the importance sampling ratio to estimate the expected return as if the target policy had been followed. Off-policy MC prediction is particularly useful in situations where the target policy cannot be directly executed but can still be evaluated based on data collected from a different policy.

Monte Carlo Control (Off-policy):

It extends the concept of off-policy prediction to learning optimal policies. It involves learning the optimal policy by observing the returns of episodes generated by a behavior policy, which is different from the target policy being optimized. This method uses importance sampling to reweight returns to accurately reflect what they would be under the target policy. Off-policy MC control allows for more flexible data collection since the behavior policy can be exploratory and may differ significantly from the optimal policy being learned. This approach is powerful in scenarios where the optimal policy cannot be explored directly, allowing for convergence to the optimal policy through indirect experience.

All four algorithms share some common components, such as the use of Monte Carlo sampling to estimate the value functions and the updates to the policy based on the value function estimates.

III. Evaluation Criteria

To effectively compare the performance and efficiency of the different Monte Carlo algorithms, we employ several evaluation criteria. These criteria provide a comprehensive assessment of each algorithm's strengths and weaknesses, helping us determine their overall effectiveness and practical applicability. The key evaluation metrics are as follows:

Cumulative Rewards

Cumulative rewards represent the total rewards accumulated over episodes, indicating the overall performance of an algorithm. Higher cumulative rewards suggest better long-term effectiveness. This criterion helps compare which algorithm consistently achieves better outcomes over time, highlighting overall policy success.

Total Samples to Reach Target Reward

This measures the sample efficiency of an algorithm, indicating how quickly it learns to achieve a target reward. Fewer samples mean higher efficiency. Comparing this criterion shows which algorithm learns faster and is more resource-efficient, important in costly or time-consuming environments.

Training Time

Training time refers to the computational time required for an algorithm to converge. Shorter training times indicate better practical feasibility and faster deployment. This criterion helps identify which algorithms are computationally efficient and scalable, important for real-time applications.

State Values (V(s))

State values estimate the expected return from a given state. Accurate state values improve decision-making. Comparing state value estimates reveals which algorithms provide reliable predictions, crucial for understanding future rewards and improving policies.

Policy (π(s))

The policy defines the agent's actions in different states. A good policy maximizes cumulative rewards. Comparing policies shows which algorithms learn effective strategies, indicating their ability to guide the agent's actions reliably and robustly.

IV. Results and Discussion

1. Comparison in Deterministic Environment (Noise =0)

We evaluated four Monte Carlo algorithms using the specific criteria that we mentioned in previous section and below are the results and comparisons for each algorithm and we used fixed number of “episodes = **1000”** through the whole implementation of four algorithms.

Cumulative Rewards

On-Policy MC with Exploring Starts: **0.4801**

On-Policy MC Control without Exploring Starts: **0.3261**

Off-Policy MC Prediction: **-2.9746**

Off-Policy MC Control: **0.5208**

Higher cumulative rewards indicate better long-term performance. Off-Policy MC Control achieved the highest cumulative rewards, suggesting it was the most effective in consistently finding high-reward paths. Off-Policy MC Prediction had significantly negative cumulative rewards, indicating poor performance.

Total Samples to Reach Target Reward

On-Policy MC with Exploring Starts: **147**

On-Policy MC Control without Exploring Starts: **1658**

Off-Policy MC Prediction: **114**

Off-Policy MC Control: **122**

Fewer samples required to reach the target reward indicate higher efficiency. Off-Policy MC Prediction required the fewest samples, demonstrating its efficiency in learning the optimal policy quickly, despite its poor cumulative rewards. On-Policy MC Control without Exploring Starts required the most samples, showing inefficiency in learning.

Training Time

On-Policy MC with Exploring Starts: **0.36 seconds**

On-Policy MC Control without Exploring Starts**: 0.26 seconds**

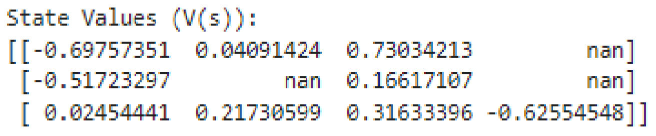
Off-Policy MC Prediction: **0.16 seconds**

Off-Policy MC Control: **0.05 seconds**

Shorter training times are preferable for practical applications. Off-Policy MC Control had the shortest training time, making it the most computationally efficient and suitable for real-time implementations.

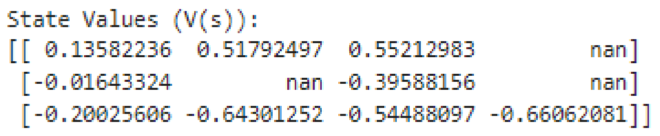
State Values (V(s))

Accurate state value estimates improve decision-making. Off-Policy MC Control provided the most balanced state values, reflecting its strength in evaluating future rewards accurately. Off-Policy MC Prediction had extremely large or erratic values, indicating instability.



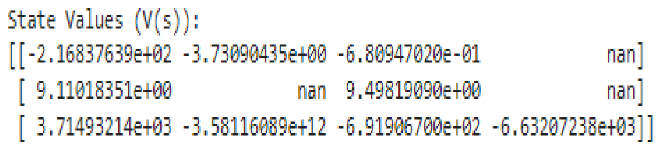
```
State Values (V(s)):
[[-0.69757351  0.04091424  0.73034213         nan]
 [-0.51723297         nan  0.16617107         nan]
 [ 0.02454441  0.21730599  0.31633396 -0.62554548]]
```

On-policy MC with exploring



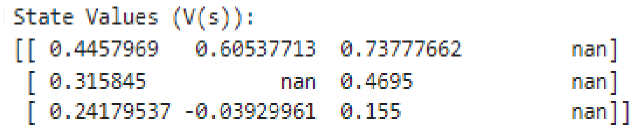
```
State Values (V(s)):
[[ 0.13582236  0.51792497  0.55212983         nan]
 [-0.01643324         nan -0.39588156         nan]
 [-0.20025606 -0.64301252 -0.54488097 -0.66062081]]
```

On-policy MC without exploring



```
State Values (V(s)):
[[-2.16837639e+02 -3.73090435e+00 -6.80947020e-01             nan]
 [ 9.11018351e+00             nan  9.49819090e+00             nan]
 [ 3.71493214e+03 -3.58116089e+12 -6.91906700e+02 -6.63207238e+03]]
```

Off-policy MC prediction

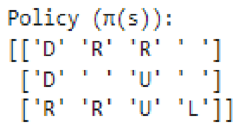


```
State Values (V(s)):
[[ 0.4457969   0.60537713  0.73777662         nan]
 [ 0.315845           nan  0.4695             nan]
 [ 0.24179537 -0.03929961  0.155             nan]]
```

Off-policy MC control

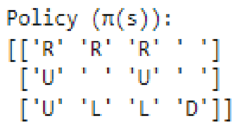
Policy (π(s))

A robust policy maximizes cumulative rewards. Off-Policy MC Control yielded a consistent and effective policy, effectively guiding the agent's actions across various states. The policies from On-Policy MC with Exploring Starts and On-Policy MC Control without Exploring Starts were also reliable but less effective in maximizing cumulative rewards compared to Off-Policy MC Control.



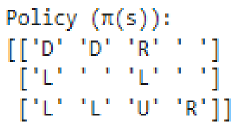
```
Policy (π(s)):
[['D' 'R' 'R' ' ']
 ['D' ' ' 'U' ' ']
 ['R' 'R' 'U' 'L']]
```

On-policy MC with exploring



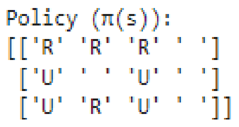
```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'L' 'L' 'D']]
```

On-policy MC without



```
Policy (π(s)):
[['D' 'D' 'R' ' ']
 ['L' ' ' 'L' ' ']
 ['L' 'L' 'U' 'R']]
```

Off-policy MC prediction

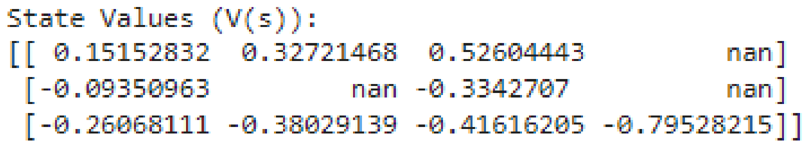


```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'R' 'U' ' ']]
```

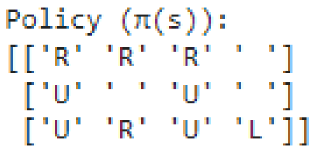
Off-policy MC control

2. Comparison in Stochastic Environment (Noise =0.4)

i. On-policy MC with Exploring Starts



```
State Values (V(s)):
[[ 0.15152832  0.32721468  0.52604443         nan]
 [-0.09350963         nan -0.3342707          nan]
 [-0.26068111 -0.38029139 -0.41616205 -0.79528215]]
```



```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'R' 'U' 'L']]
```

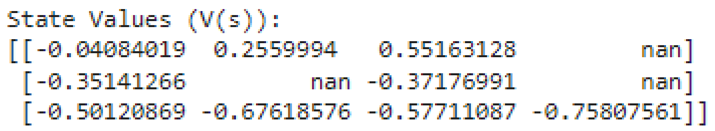
Cumulative Rewards: **0.048**

Total Samples to Reach Target Reward: **331**

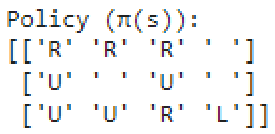
Training Time: **0.38 seconds**

In a stochastic environment, the state values show more variability, reflecting the added uncertainty in transitions. The cumulative reward decreased, and more samples were needed to reach the target reward compared to the deterministic environment, indicating the increased difficulty in learning.

ii. On-policy MC control without exploring starts



```
State Values (V(s)):
[[-0.04084019  0.2559994   0.55163128         nan]
 [-0.35141266         nan -0.37176991         nan]
 [-0.50120869 -0.67618576 -0.57711087 -0.75807561]]
```



```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'U' 'R' 'L']]
```

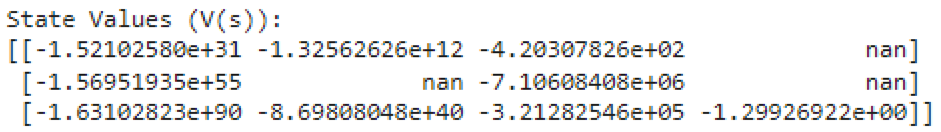
Cumulative Rewards: **-0.35650**

Total Samples to Reach Target Reward: **3092**

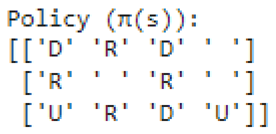
Training Time: **0.45 seconds**

The cumulative rewards turned negative, and the algorithm required significantly more samples to reach the target reward. This indicates that without exploring starts, the algorithm struggled more in the stochastic environment, reflecting less efficient learning and policy optimization.

iii. Off-policy MC prediction



```
State Values (V(s)):
[[-1.52102580e+31 -1.32562626e+12 -4.20307826e+02             nan]
 [-1.56951935e+55             nan -7.10608408e+06             nan]
 [-1.63102823e+90 -8.69808048e+40 -3.21282546e+05 -1.29926922e+00]]
```



```
Policy (π(s)):
[['D' 'R' 'D' ' ']
 ['R' ' ' 'R' ' ']
 ['U' 'R' 'D' 'U']]
```

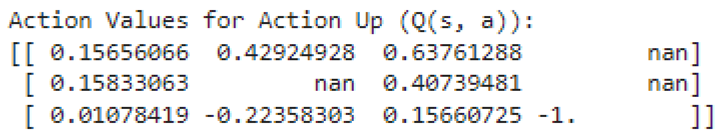
Cumulative Rewards: **-2.42770**

Total Samples to Reach Target Reward: **7213**

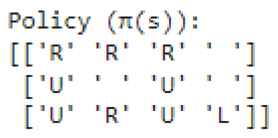
Training Time: 0.47 seconds

The state values became extremely large, indicating instability in the algorithm's predictions. The cumulative rewards were highly negative, and the number of samples needed was significantly higher, showing that off-policy MC prediction struggled with the added noise, leading to poor performance and inefficiency.

iv. Off-policy MC control



```
Action Values for Action Up (Q(s, a)):
[[ 0.15656066  0.42924928  0.63761288         nan]
 [ 0.15833063         nan  0.40739481         nan]
 [ 0.01078419 -0.22358303  0.15660725 -1.        ]]
```



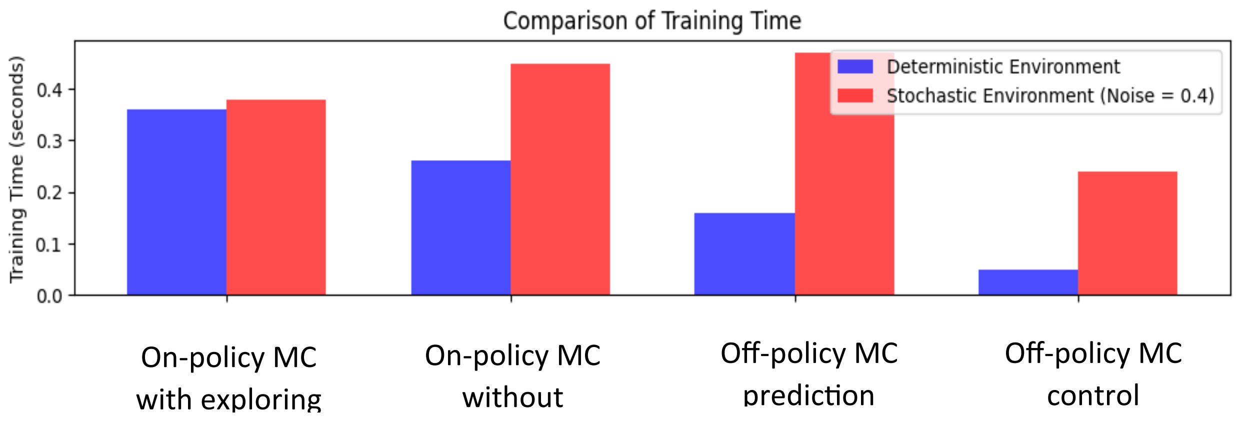
```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'R' 'U' 'L']]
```

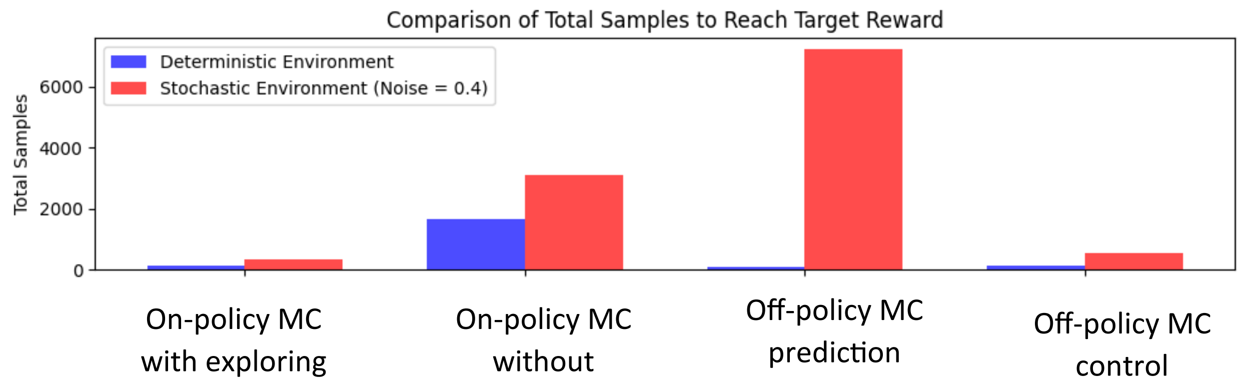
Cumulative Rewards: **0.0569**

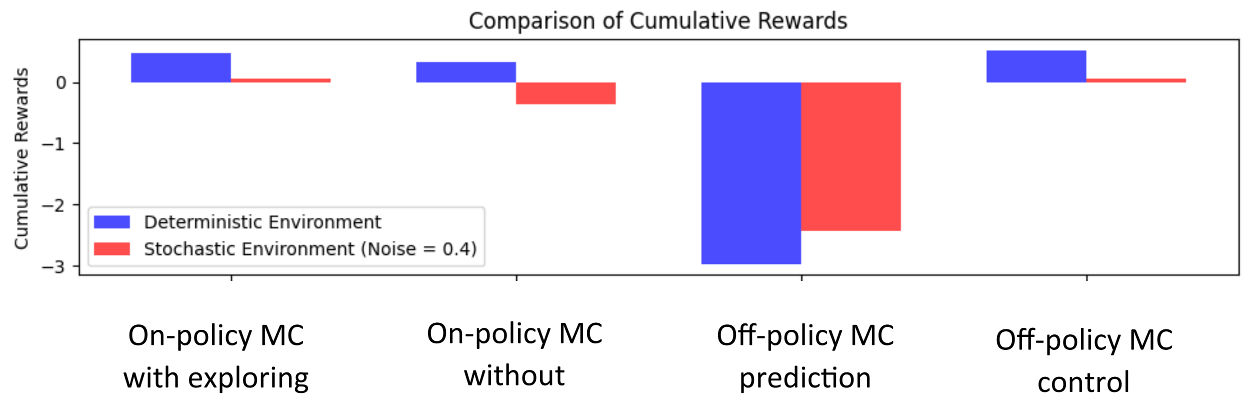
Total Samples to Reach Target Reward: **540**

Training Time: **0.24 seconds**

Off-policy MC control maintained relatively stable state values and achieved a small positive cumulative reward. The total samples to reach the target reward increased compared to the deterministic environment, but the algorithm still performed reasonably well under noise, showing robustness in policy control.







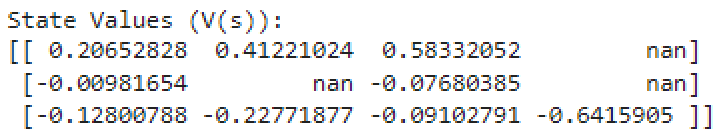
Effect of Noise on the Environment

Introducing noise to the environment significantly impacted the performance of all algorithms. The stochastic transitions increased the uncertainty in learning, making it harder for the algorithms to accurately estimate state and action values. This led to increased sample requirements and, in some cases, negative cumulative rewards. On-policy algorithms, especially without exploring starts, struggled more with the added complexity, while off-policy control maintained better stability and performance. Overall, noise added a layer of difficulty, highlighting the importance of robust learning methods in uncertain environments.

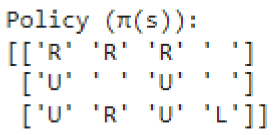
3. Study the effect of exploration strategy (epsilon) at Deterministic Environment

3.1 Epsilon =0.4

i. On-policy MC with Exploring Starts



```
State Values (V(s)):
[[ 0.20652828  0.41221024  0.58332052         nan]
 [-0.00981654         nan -0.07680385         nan]
 [-0.12800788 -0.22771877 -0.09102791 -0.6415905 ]]
```



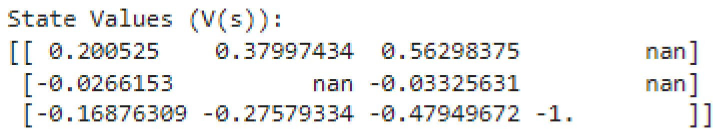
```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'R' 'U' 'L']]
```

Cumulative Rewards: **0.21950**

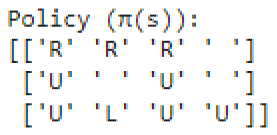
Total Samples to Reach Target Reward: **217**

Training Time: **0.34 seconds**

ii. On-policy MC control without exploring starts



```
State Values (V(s)):
[[ 0.200525    0.37997434  0.56298375         nan]
 [-0.0266153          nan -0.03325631         nan]
 [-0.16876309 -0.27579334 -0.47949672 -1.        ]]
```



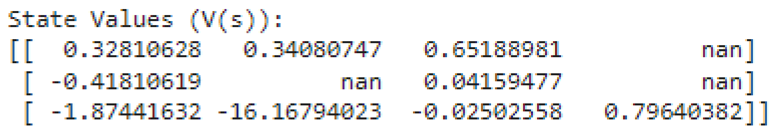
```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'L' 'U' 'U']]
```

Cumulative Rewards: **0.18730**

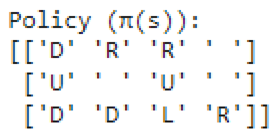
Total Samples to Reach Target Reward: **245**

Training Time: **0.48 seconds**

iii. Off-policy MC prediction



```
State Values (V(s)):
[[  0.32810628   0.34080747   0.65188981          nan]
 [ -0.41810619          nan   0.04159477          nan]
 [ -1.87441632 -16.16794023  -0.02502558   0.79640382]]
```



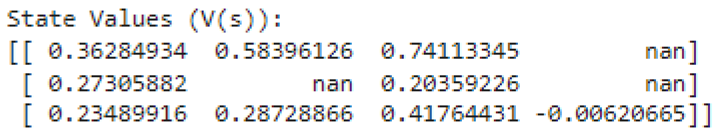
```
Policy (π(s)):
[['D' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['D' 'D' 'L' 'R']]
```

Cumulative Rewards: **-1.64870**

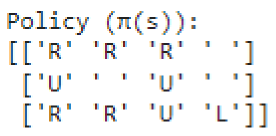
Total Samples to Reach Target Reward: **109**

Training Time: **0.09 seconds**

iv. Off-policy MC control



```
State Values (V(s)):
[[ 0.36284934  0.58396126  0.74113345         nan]
 [ 0.27305882         nan  0.20359226         nan]
 [ 0.23489916  0.28728866  0.41764431 -0.00620665]]
```



```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['R' 'R' 'U' 'L']]
```

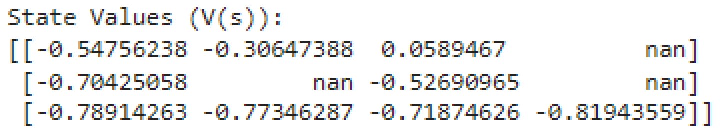
Cumulative Rewards: **0.01360**

Total Samples to Reach Target Reward: **251**

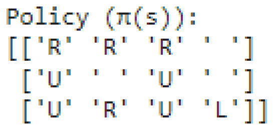
Training Time: **0.05 seconds**

3.1 Epsilon =0.9

i. On-policy MC with Exploring Starts



```
State Values (V(s)):
[[-0.54756238 -0.30647388  0.0589467          nan]
 [-0.70425058         nan -0.52690965         nan]
 [-0.78914263 -0.77346287 -0.71874626 -0.81943559]]
```



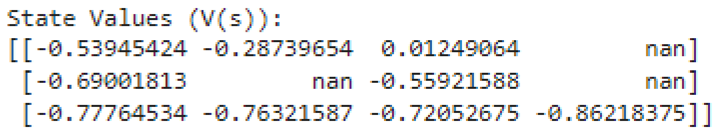
```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'R' 'U' 'L']]
```

Cumulative Rewards**: -1.83540**

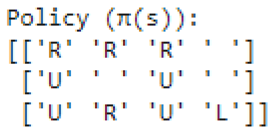
Total Samples to Reach Target Reward: **313**

Training Time: **0.44 seconds**

ii. On-policy MC control without exploring starts



```
State Values (V(s)):
[[-0.53945424 -0.28739654  0.01249064         nan]
 [-0.69001813         nan -0.55921588         nan]
 [-0.77764534 -0.76321587 -0.72052675 -0.86218375]]
```



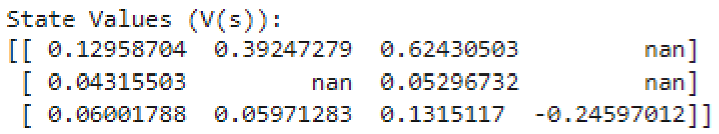
```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'R' 'U' 'L']]
```

Cumulative Rewards: **-1.82390**

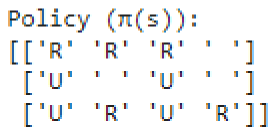
Total Samples to Reach Target Reward: **3632**

Training Time: **0.45 seconds**

iii. Off-policy MC prediction



```
State Values (V(s)):
[[ 0.12958704  0.39247279  0.62430503         nan]
 [ 0.04315503         nan  0.05296732         nan]
 [ 0.06001788  0.05971283  0.1315117  -0.24597012]]
```



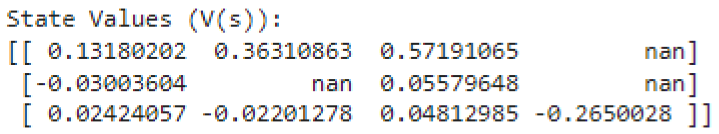
```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'R' 'U' 'R']]
```

Cumulative Rewards: **-1.78630**

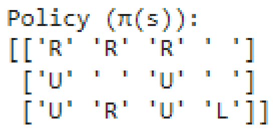
Total Samples to Reach Target Reward: **1856**

Training Time: **0.09 seconds**

iv. Off-policy MC control



```
State Values (V(s)):
[[ 0.13180202  0.36310863  0.57191065         nan]
 [-0.03003604         nan  0.05579648         nan]
 [ 0.02424057 -0.02201278  0.04812985 -0.2650028 ]]
```

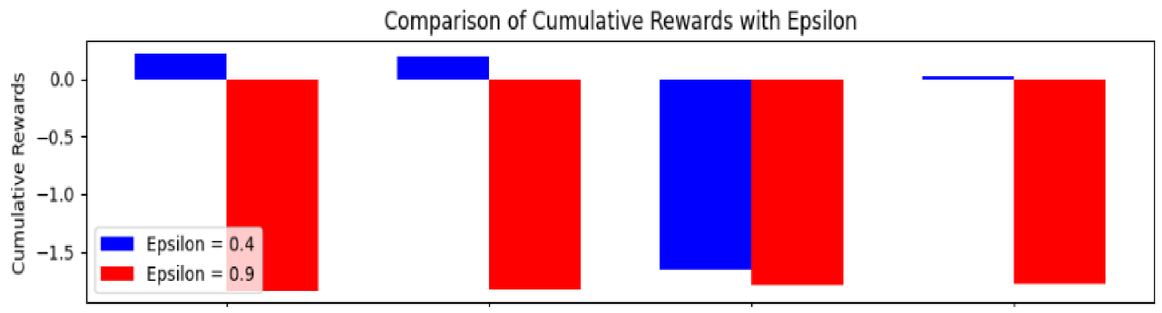


```
Policy (π(s)):
[['R' 'R' 'R' ' ']
 ['U' ' ' 'U' ' ']
 ['U' 'R' 'U' 'L']]
```

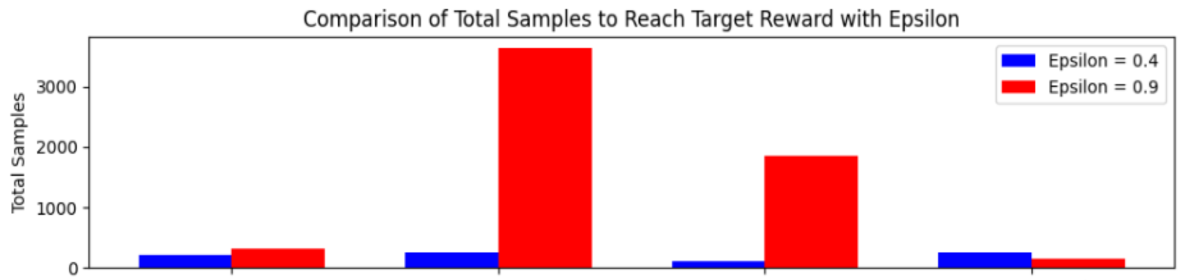
Cumulative Rewards**: -1.77760**

Total Samples to Reach Target Reward: **157**

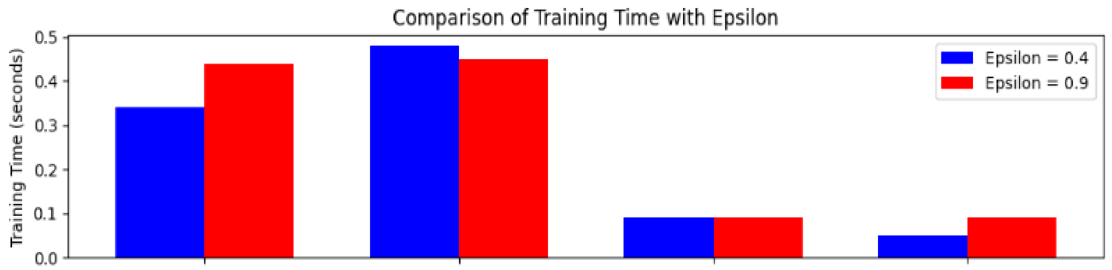
Training Time: **0.09 seconds**



On-policy MC with exploring | On-policy MC without | Off-policy MC prediction | Off-policy MC control



On-policy MC with exploring | On-policy MC without | Off-policy MC prediction | Off-policy MC control



On-policy MC with exploring | On-policy MC without | Off-policy MC prediction | Off-policy MC control

Effect of exploration strategy (epsilon Value) through algorithms

The effect of changing epsilon from **0.4** to **0.9** is notable in the results. With a higher epsilon of 0.9, the exploration rate is increased, leading to more random actions taken during training. This randomness can cause the agent to explore a wider range of states and actions, which may result in a lower cumulative reward and longer training time compared to when epsilon is lower (0.4). However, the impact of epsilon on the training time may vary depending on the specific algorithm and environment dynamics.

V. Conclusion

In summary, we studied four Monte Carlo algorithms in different environments and explored how changing the exploration strategy affects their performance. Here's what we found:

1. Deterministic Environment: Off-Policy MC Control performed the best. It had higher rewards, used samples efficiently, trained quickly, and made good decisions.
2. Stochastic Environment (Noise = 0.4): Off-Policy MC Control still did well, but noise made learning harder for all algorithms. On-Policy algorithms struggled more with the randomness.
3. Effect of Exploration (Epsilon): Higher epsilon values led to more exploration but also more randomness. This affected rewards and training times differently for each algorithm.

In conclusion, Off-Policy MC Control stood out as a reliable choice, especially in uncertain situations. Balancing exploration and exploitation remain important for getting the best results in reinforcement learning.