Project -2

Himani Thakkar

Parinaz Shiri

Reinforcement Learning
(DSCI 6650)

Table of Contents

PART - 1

EXPERIMENT DESCRIPTION

The GridWorld environment is a 5x5 grid representing a reinforcement learning problem. Each cell in the grid represents a possible state. The agent can move in four directions: up, down, left, and right. If the agent attempts to move off the grid, it remains in its current position and receives a penalty.

SPECIAL STATES:

- Blue Square: Any action taken in this state yields reward of 5 and causes the agent to jump to the Red Square.
- Green Square: Any action taken in this state yields a reward of 2.5 and causes the agent to jump to either the Yellow Square or the Red Square with equal probability (0.5 each).
- Red Square and Yellow Square: Serve as potential jump destinations from the Green Square, and Blue Square

REWARD STRUCTURE:

- Blue Square: +5 reward and jump to Red Square.
- Green Square: +2.5 reward and jump to either Yellow or Red Square with 0.5 probability.
- Attempt to Step Off the Grid: -0.5 reward.
- Move from a White Square to Another White Square: 0 reward.

EXPERIMENT SETUP:

- Initialization: The agent starts at a random white square.
- Actions: The agent selects an action (up, down, left, or right) at each time step.
- State Transition: The agent transitions to a new state based on the selected action and receives a corresponding reward.

TASK:

Use reinforcement learning algorithms to learn an optimal policy and calculate states with highest values.

QUESTION 1 - Consider a reward discount of $\gamma = 0.95$ and a policy which simply moves to one of the four directions with equal probability of 0.25. Estimate the value function for each of the states using (1) solving the system of Bellman equations explicitly (2) iterative policy evaluation (3) value iteration. Which states have the highest value?

SOLUTION - We created a grid world of size 5x5, with the expected positions of the blue, green, yellow, and red states, along with an option to choose the method to estimate the value function and begin the evaluation. The output will produce the results with the highest state the and the value.

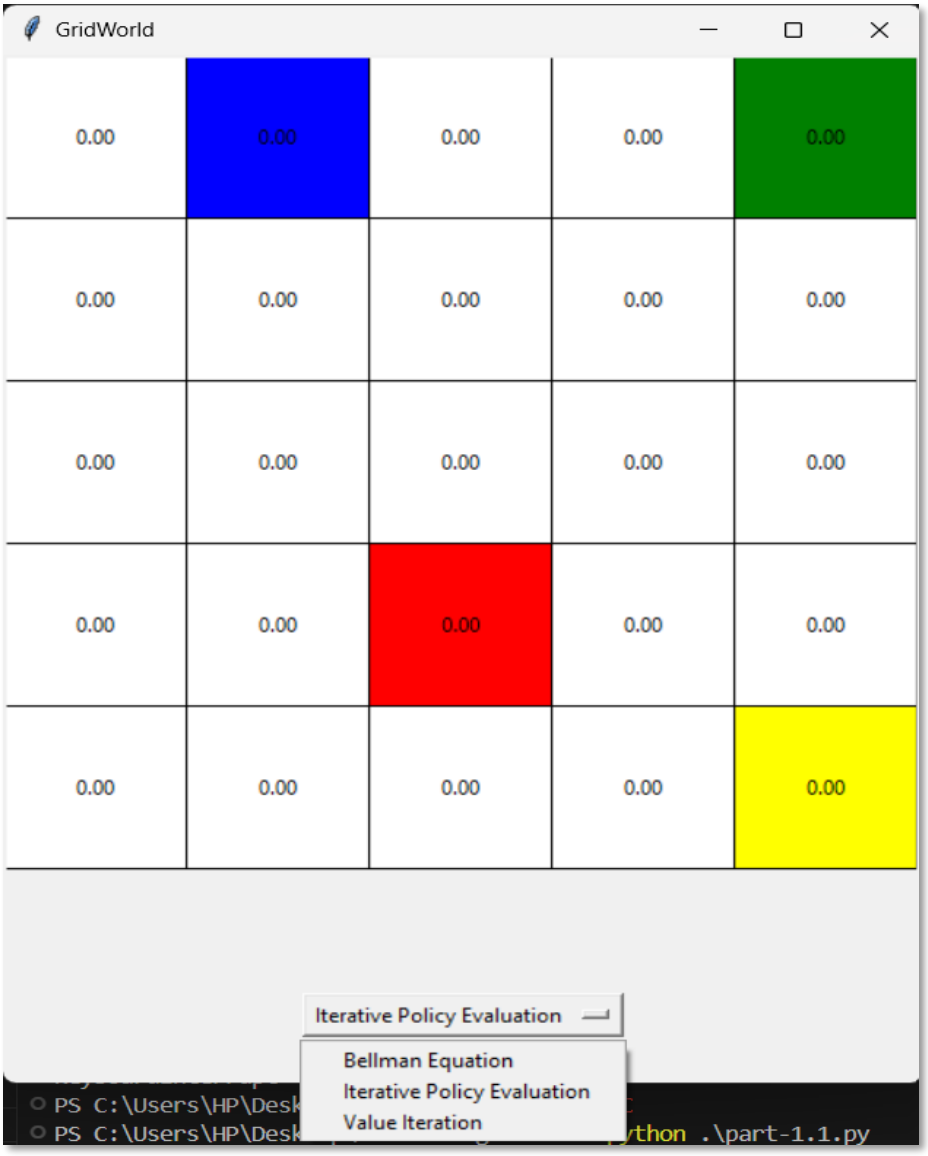

Figure 1: Grid World

1.1) SOLVING THE BELLMAN EQUATIONS EXPLICITLY

The Bellman equation averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the discounted value of the expected next state, and reward expected along the way.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right], \quad \text{for all } s \in \mathcal{S},$$

The value function was computed by solving the linear equation. The blue state is the best state under the policy, which gave a reward of +5 for any action and jumped to state red. The states on the border have the least (negative) value, as the chances of running into the edge and getting a negative reward of -0.5 is the highest.

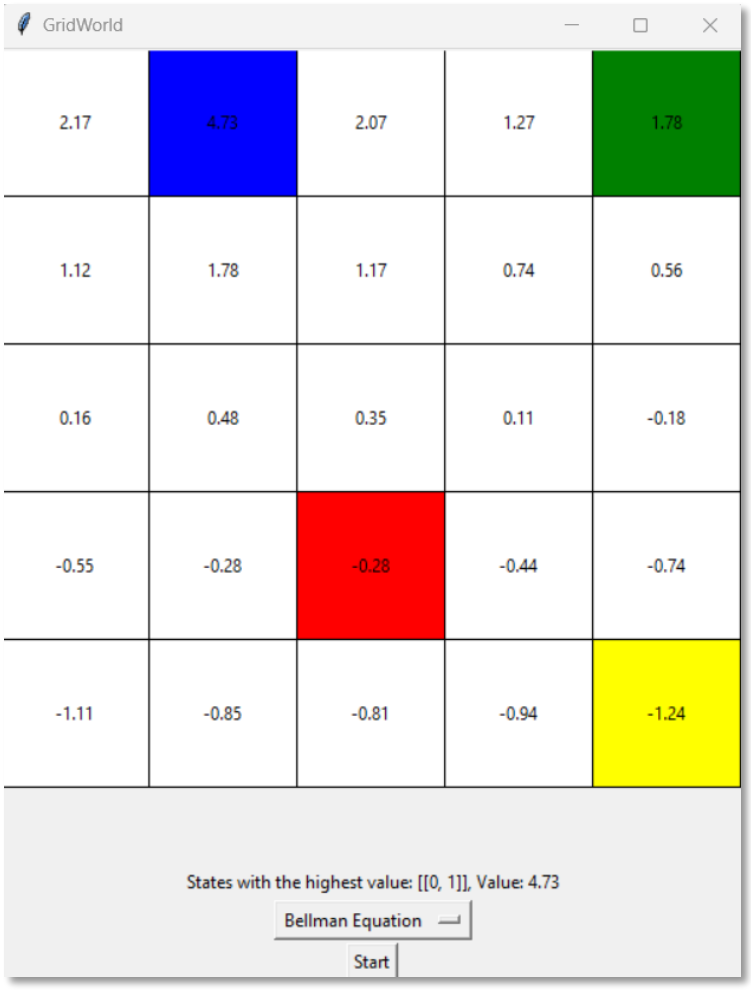

Figure 2: Solving Bellman Equation Explicitly

2.1) ITERATIVE POLICY EVALUATION

The bellman equation can be modified to create an update rule for the algorithm.

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

It updates the current value of state *s* with a new value calculated from the existing values of the successor states of *s* and the expected immediate rewards for all possible one-step transitions under the policy being evaluated. Each iteration of iterative policy evaluation revises the value of every state once, resulting in a new approximate value function vk+1. We took an epsilon value of 0.01 and if the value of delta (change) is less than that, the algorithm would terminate. It involves repeatedly updating the value function until it converges to the true value function for that policy.

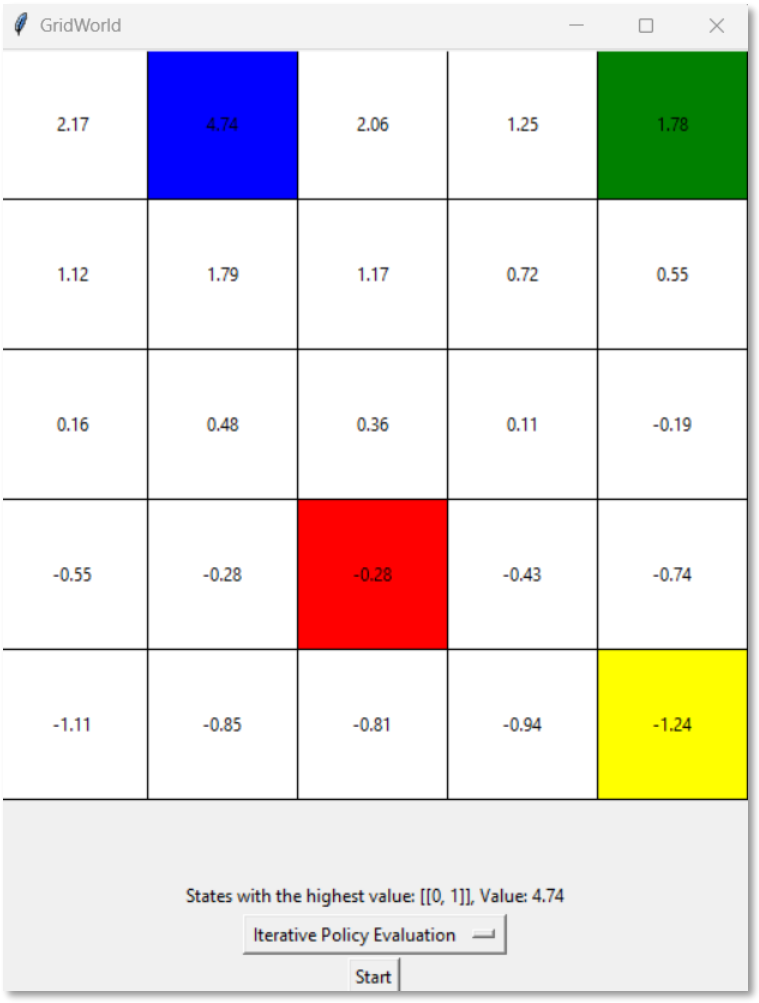

Figure 3: Iterative Policy Evaluation

Similarly, the output of the Iterative Policy Evaluation also produces a similar output with the highest state being (0,1) – The blue state.

2.2) Value Iteration

The previous method requires multiple sweeps through the state set, and we need to wait until convergence. Value iteration stops after just one update of each state. It combines the policy improvement and truncated policy evaluation. It takes a maximum over all actions. Its termination is similar to policy evaluation, which stops when the function changes by a smallest amount.

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

We took an epsilon value of 0.01 and if the value of delta (change) is less than that, the algorithm would terminate.

Figure 4: Value Iteration

As expected, the highest state is (0,1). The value iteration converges faster than any other method used before, while producing the same result.

QUESTION 2 - Determine the optimal policy for the grid world problem by (1) explicitly solving the Bellman optimality equation (2) using policy iteration with iterative policy evaluation (3) policy improvement with value iteration.

SOLUTION - A policy π^* is said to be better than all other policies π' for all states *s* such that $v\ \pi^*(s) >= v\ \pi(s)$, which is known as the optimal policy.

In this experiment, we explored three methods to determine the optimal policy for a 5x5 GridWorld problem, keeping the dynamics the same as part 1.1. We solved the Bellman optimality equation, policy iteration with iterative policy evaluation, and policy improvement with value iteration. Additionally, we added a reset button that would bring the environment back to it's original to select another option again.

2.1) SOLVING THE BELLMAN EQUATION EXPLICITLY TO GET THE OPTIMAL POLICY

The Bellman optimality equation is solved iteratively. For each state, the action that maximizes the expected value is selected.
The value function is updated until convergence, ensuring minimal change between iterations.

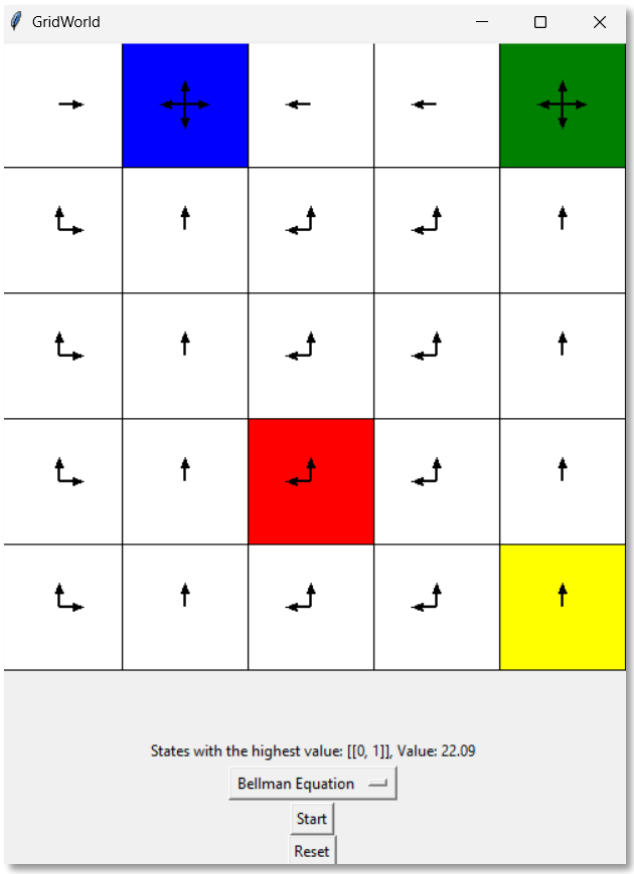

Figure 5: Bellman Equation Optimal Policy

The value function converges to the optimal value for each state and the optimal policy directs the agent to states with the highest rewards.
Special states like the green state have dynamic optimal actions depending on pre-defined conditions. The optimal policy stays the same for different runs of the algorithm however, the green state changes. This result can be linked with equal of probability of moving between red and yellow state.
The optimal policy is all states moving towards the blue state, which has the highest reward. We moved the location of the policy state to confirm our results, to a different position (1,1) and the optimal policy changed to all state moving and pointing towards (1,1) indicating the move to the highest reward state.

2.2) ITERATIVE POLICY EVALUATION

A policy is evaluated iteratively by calculating the value function for the current policy until convergence.
The policy is then improved by selecting actions that maximize the expected value based on the evaluated value function. The process repeats until the policy stabilizes.

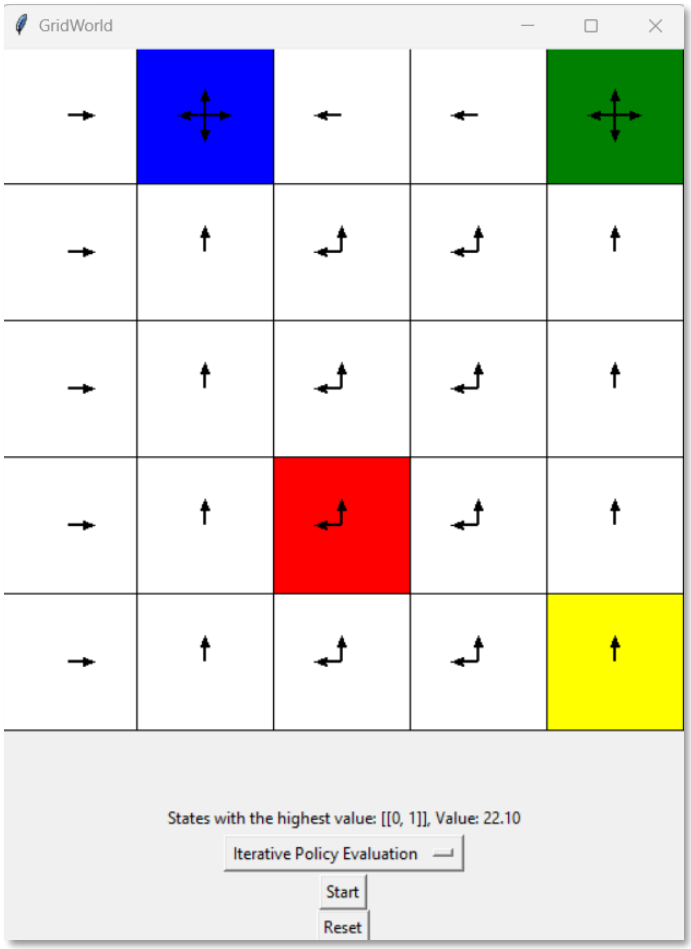

Figure 5: Policy Evaluation Optimal Policy

The iterative process effectively finds the optimal policy through repeated evaluation and improvement.
The resulting policy is optimal and consistent with the Bellman equation method.

2.3) VALUE ITERATION

The main difference between value iteration and iterative policy evaluation is that value iteration directly finds the optimal value function by considering the best action at each step and the other evaluates the value function for a specific policy and then improves the policy based on the evaluation.

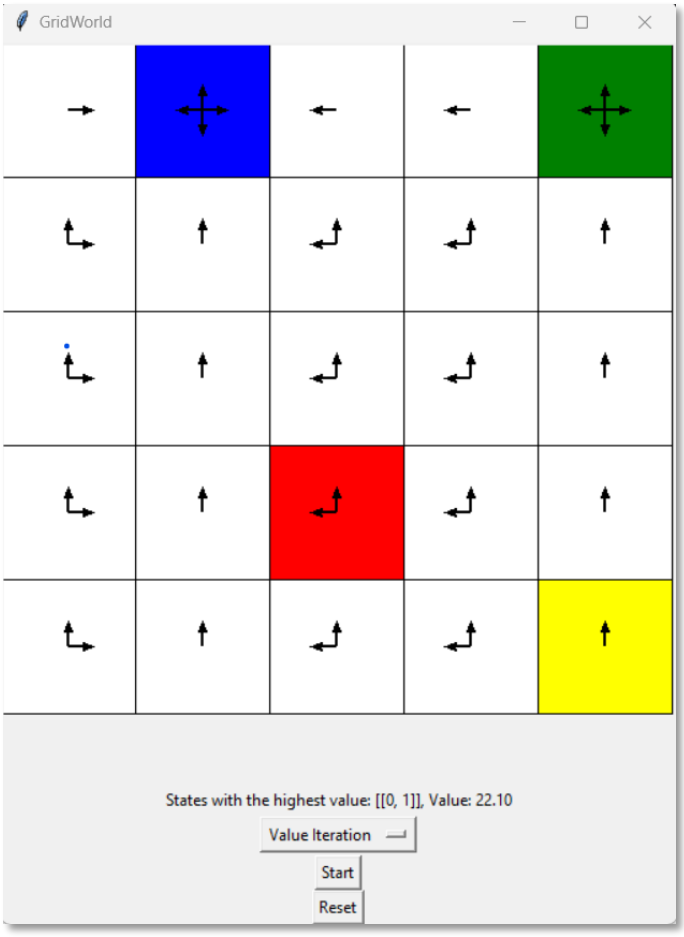

Figure 6: Value Iteration Optimal Policy

Like the other methods, value iteration shows the same optimal policy with all states pointing towards the state with the highest reward (blue). As expected, the behavior of the green state changes every run, due to the probability of agent jumping to either red or yellow state. We confirmed this, by fixing the jump to just yellow and putting the condition where it will only jump to red if the value of np.random.rand() >1, which will never occur. In that scenario, it always moves to yellow regardless of the action taken and the policy stayed the same except the direction of the green being the same with every new run.

PART 1 CONCLUSION

The highest state in this environment is the blue state, having the highest reward and the optimal policy is all states moving towards the blue state as well. This was consistent across all three methods. The key difference between the three methods is the performance and the convergence. Value iteration is the quickest to find the optimal policy followed by iterative policy evaluation.

PART - 2

EXPERIMENT DESCRIPTION

The GridWorld environment is the same as before. Each cell in the grid represents a state. The agent can move in four directions: up, down, left, and right. If the agent attempts to move off the grid, it remains in its current position and receives a penalty.

SPECIAL STATES:

- Blue Square: Any action taken in this state yields reward of 5 and causes the agent to jump to the Red Square.
- Green Square: Any action taken in this state yields a reward of 2.5 and causes the agent to jump to either the Yellow or the Red Square with equal probability (0.5 each).
- Red and Yellow: Serve as potential jump destinations from the Green, and Blue Square
- Black Squares: Serves as terminal states in the grid and termination occurs once the agent hits one of the black squares.

REWARD STRUCTURE:

- Blue Square: +5 reward and jump to Red Square.
- Green Square: +2.5 reward and jump to either Yellow or Red Square with 0.5 probability.
- Attempt to Step Off the Grid: -0.5 reward.
- Move from a White Square to Another White Square: -0.2 reward.
- Black Squares: terminal states' rewards are not affecting the algorithm. In here we assume that their reward is 0.

EXPERIMENT SETUP:

Episodic Instruction: When the agent hits any of the terminal states that episode terminates and then a new episode will be generated.

QUESTION 1 - Use the Monte Carlo method with (1) exploring starts and (2) without exploring starts but the ε-soft approach to learn an optimal policy for this modified gridworld problem. Use the same discount factor of $\gamma = 0.95$ as you have in the Part 1 above. You can start with a policy with equiprobable moves.

SOLUTION - We created a grid world of size 5x5, with the expected positions of the blue, green, yellow, red, and black states, along with an option to choose the exploring starts or fixed start Monte Carlo method to learn the optimal policy for this modified grid. The output will produce the results with the highest state the and the value.

1.1) MONTE CARLO WITH EXPLORING STARTS

Episode Generation: algorithm generates episodes starting from random states except the terminal states, ensuring diverse exploration of the state space.
For each episode, the agent takes actions according to the current policy and transitions to new states, accumulating rewards until reaching a terminal state.
Return Calculation: For each episode, the algorithm calculates the return (G) by summing the discounted rewards from the end of the episode to the beginning.
The discount factor (gamma) is applied to future rewards to reduce their present value.

$$G = \gamma G + R_{t+1}$$

where γ is the discount factor, G is the cumulative return, and Rt+1 is the reward received after taking the action.

Policy and Value Update:

The algorithm maintains a record of returns for each state-action pair and their counts.

For each state-action pair encountered in an episode, the average return is updated using the new return (G).

The value function for each state is updated based on the average return. The policy for each state is updated to the action with the highest return. An epsilon-greedy approach is used to maintain exploration. The best action is assigned a probability of $1-\epsilon$, while the rest of the actions in that state are each assigned a probability of $\frac{\epsilon}{number\ of\ actions}$.

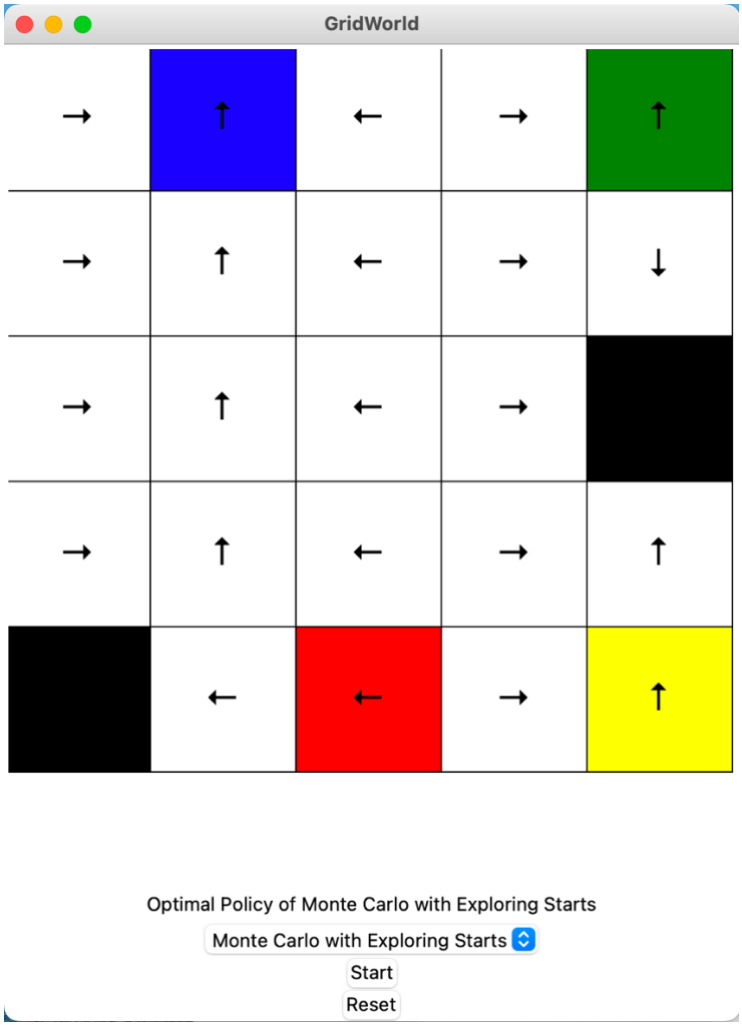

Figure 7: Monte Carlo with Exploring Starts Optimal Policy

1.2) MONTE CARLO WITHOUT EXPLORING STARTS

Monte Carlo methods without exploring starts learns an optimal policy by generating episodes from arbitrary starting states in the environment rather than ensuring coverage of every state-action pair from the outset. The process begins with an initialization of the state-action values and the policy, which starts as equiprobable across actions for each state. Episodes are generated by following the current policy from randomly chosen starting states, with the sequence of states, actions, and rewards recorded.

After each episode, the value estimates for state-action pairs are updated based on the returns observed, and the policy is refined to be ϵ-soft, meaning it combines exploitation (choosing the best action with high probability) with (choosing random actions with a small probability).

This approach relies on the policy's inherent exploration mechanism to eventually visit all state-action pairs, iteratively improving the policy until it converges to an optimal solution.

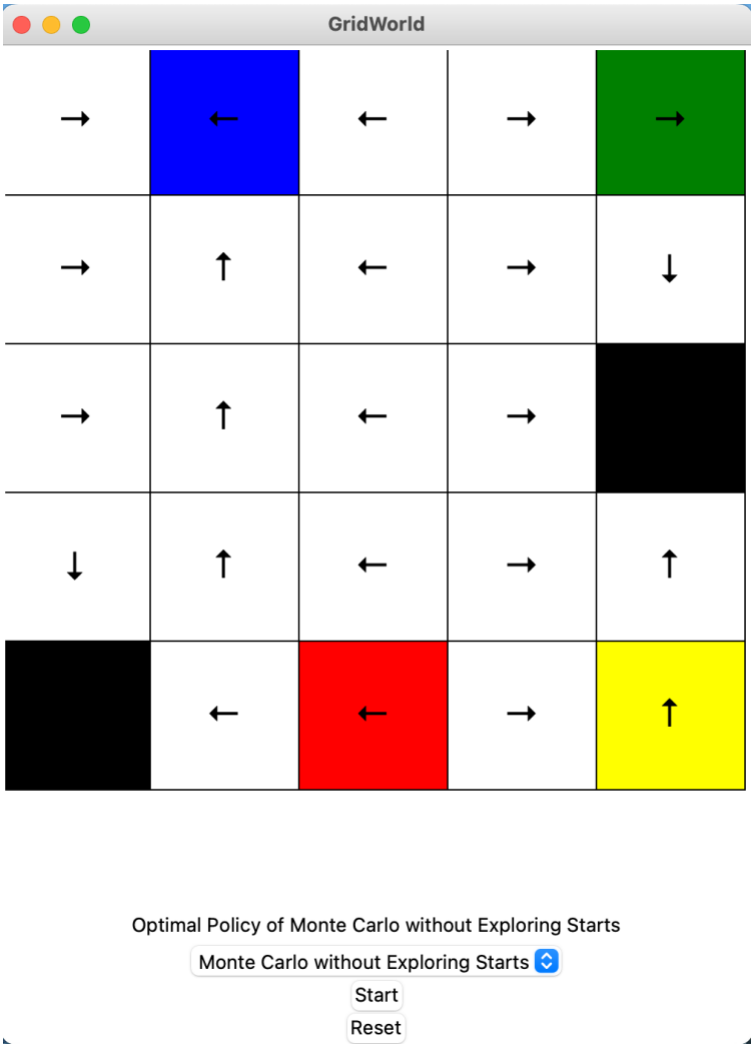

Figure 7: Monte Carlo without Exploring Starts Optimal Policy

Comparison of Monte Carlo Method with and without exploring starts:

Unlike the exploring starts approach, Monte Carlo without exploring starts method generates each episode from the same initial state. This can lead to less diverse exploration of the state space, particularly in environments with many states. However, it provides a consistent starting point, which can be beneficial for learning optimal policies from a specific initial state.

Conclusion based on the optimal policy of Monte Carlo:

If the agent is close to terminal states the optimal policy would guide it to the terminal states because transferring to white states have the reward of -0.2 and because that we have gamma of 0.95 the priority is to the immediate reward that the agent gets and not the one that it gets in long run if we change the value of gamma to lower value for example we changed it to 0.45 so that it is less dependent on immediate reward and cares more about the long term

reward. In case where gamma is 0.45 the optimal policy leads the agent to blue state even if it is near to the terminal state because the algorithm calculates that the agent might get a few small negative rewards but it will get a reward of 5 after getting into the blue state.

Additional Point:

In each episode generated, when the agent gets into the green state there is a probability of 0.5 of jumping into either red or yellow state. So that after each run of the Monte Carlo algorithm with or without exploring the policy for the green state changes according to the fact that if the agent randomly got into red or yellow state.

QUESTION 2 - Now use a behavior policy with equiprobable moves to learn an optimal policy. Note here the dynamics of the world are known exactly, so you can actually compute the importance weights needed for this.

SOLUTION - Off-policy methods consist of two policies, one that learned about and become the optimal policy and one that is more exploratory and is used to generate behavior. The policy being learned about is called target policy, and the policy used to generate behavior is called the behavior policy. In this case we say that learning is from data "off" the target policy, and the overall process is termed off-policy learning. In this section π is the target policy, b is behavioral policy, and both policies are considered fixed and given.

Off-policy methods utilize *importance sampling*, a general technique for estimating expected values under one distribution given samples from another.

In this part we used *weighted importance sampling* which uses a weighted average, defined as:

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

In the weighted-average estimate, the ratio $P_{t:T(t)-1}$ for the single return cancels in the numerator and denominator, so that the estimate is equal to the observed return independent of the ratio (assuming the ratio is nonzero). Given that this return was the only one observed, this is a reasonable estimate, but its expectation is $v_b(s)$ rather than $v_\pi(s)$, and in this statistical sense it is biased (though the bias converges asymptotically to zero). In fact, assuming bounded returns, the variance of the weighted importance-sampling estimator converges to zero even if the variance of the ratios themselves is infinite

We choose weighted estimator method because this method has dramatically lower variance compared to other methods like *ordinary importance-sampling estimator* so this method is strongly preferred in practice.

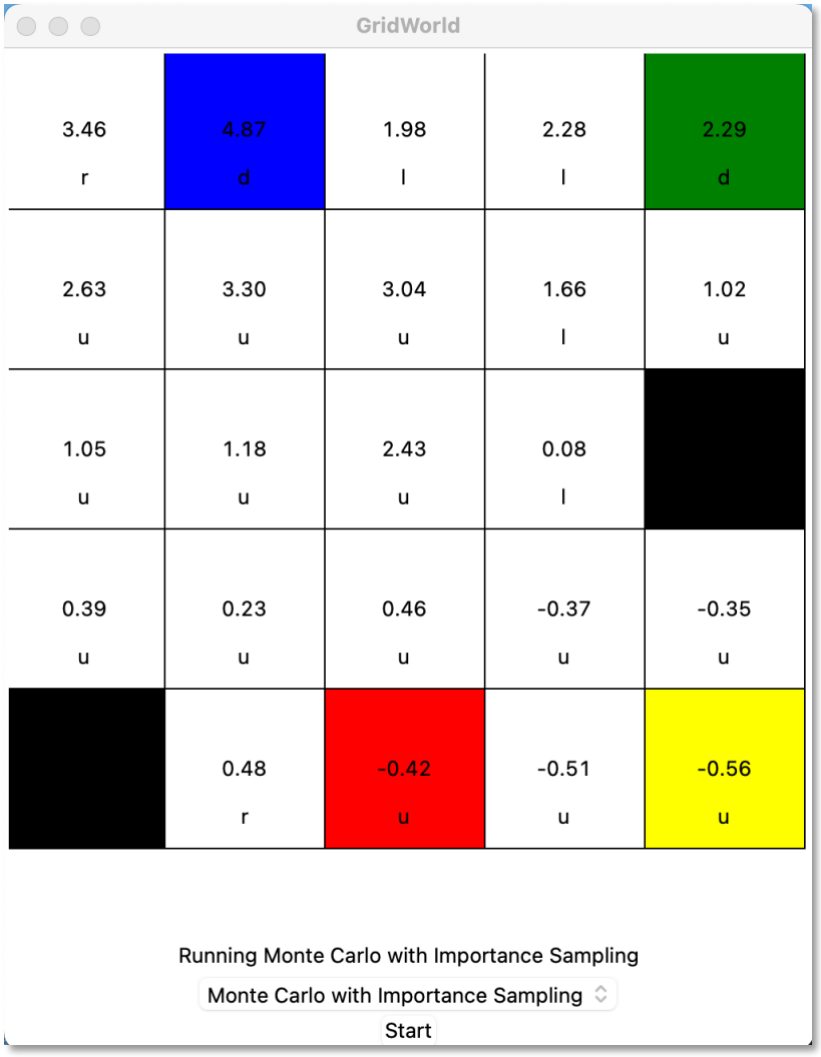

Figure 8: Monte Carlo with Importance Sampling

The implementation of the Monte Carlo with Importance Sampling algorithm in the GridWorld simulation successfully demonstrated the process of policy evaluation and improvement. Over 10,000 episodes, the value function for each state converged to accurate expected returns, with states near special states showing higher values due to their proximity to significant rewards. The initial equiprobable behavioral policy evolved into a refined target policy, favoring actions that led to higher returns. Special states, such as (0, 1) and (0, 3), with predefined transitions and rewards, notably influenced the values and decisions in their vicinity, illustrating the impact of these states on the learning process.

The algorithm effectively utilized importance sampling weights to adjust updates, ensuring accurate value estimates and reliable policy improvements despite differences between the behavior and target policies. Periodic updates every 100 episodes provided real-time insights into the learning dynamics, showing steady progress towards optimal value function convergence and policy stabilization. In summary, the Monte Carlo with Importance Sampling algorithm efficiently learned and optimized the agent's decision-making in the GridWorld environment, demonstrating robust capabilities in policy evaluation and improvement.

QUESTION 3 - Let's suppose that at every step, we permute the locations of the green and blue squares with probability 0.1, while preserving the rewards and transition structure as before. Use policy iteration to determine a suitable policy for this environment. How does it differ from the case where the squares stay where they are?

SOLUTION - In this experiment, the dynamics of the environment change and introduce permutation of blue and green squares with a probability of 0.1, and add the terminal states at [2,4] and [4,0]. In a static environment, the policy would converge more straightforwardly, however in this dynamic environment, it should be more robust to changes and keep a track of the terminal states that would end the task. In this scenario, we implemented policy iteration using policy evaluation and policy iteration. θ (theta) is a small positive threshold value used to determine the stopping criterion for the policy evaluation phase. It represents a measure of the maximum allowable difference between successive iterations of the value function. When the change in the value function is smaller than θ for all states, the policy evaluation is considered to have converged.

To be able to see the permutation, we changed the value of theta to 0.00001, to get the most accurate estimations. The algorithm, converged very quickly despite the permutations, terminal states and the small value of theta. After running the algorithm, the permutation of blue and green states is visible and the policy changes accordingly trying to estimate the optimal policy in this stochastic grid environment.

The output below displays the policy priorities actions that move the agent towards the state with the highest rewards (blue), while also accounting for the permutations. The policy avoids the actions that lead to the terminal state. The permutations reflect the agent's adaptation to the possibility of the stochastic state permutations and considers immediate and future rewards. We also tried different variations of the grid environment, by changing the position of the reward state, terminal states, and the policy always resulted in the same conclusion.

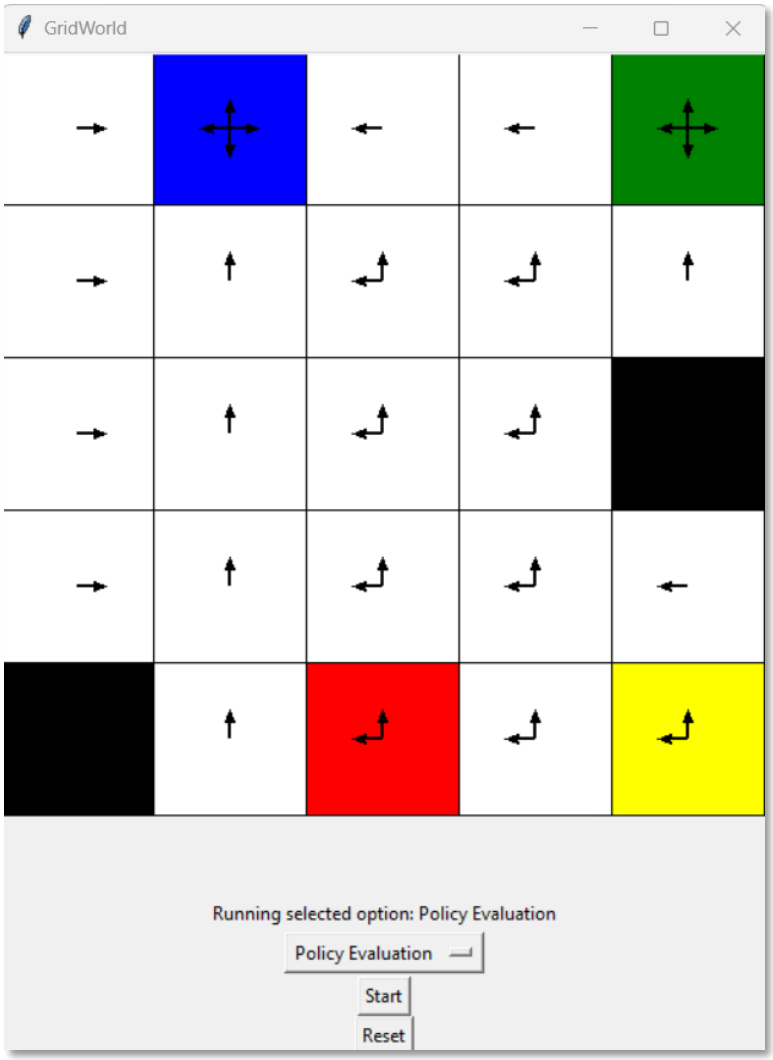

Figure 9: Optimal Policy using Policy Evaluation in Dynamic Environment

Additionally, we also used Monte Carlo and generated 10,000 episodes. The episode records the sequence of states, actions, and rewards until a terminal state is reached. The returns for each state-action pair are computed by discounting future rewards using gamma (0.95). For each episode, the algorithm updates the value function and policy based on the observed returns. The value function V(s) is updated incrementally to reflect the average returns observed for each state under the current policy. The policy is improved by selecting actions that maximize the expected return, and the action probabilities are adjusted using an ϵ-greedy approach to ensure exploration. This simulation takes longer to run compared to the policy iteration approach. The output was like policy iteration.

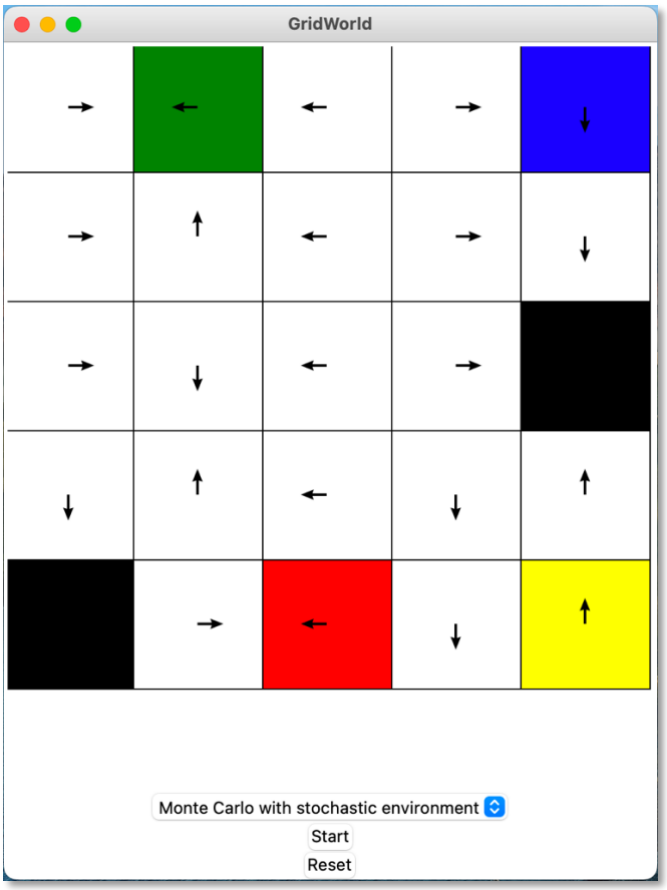

Figure 10: Monte Carlo algorithm in Dynamic Environment while running

How does this differ from the case where the squares stays where they are?

In this case, the stochastic environment which permutes the blue and green square, is the opposite of a deterministic environment. Such stochastic environments closely resemble real time environments. The next state is totally unpredictable for the agent. So, randomness exists in the environment. We noticed that, despite using a dynamic environment, the algorithm was able to converge to the optimal policy. The key difference between the two environments, that the dynamic environment would explore more, is to accurately estimate the transition probabilities and rewards in the face of these changes.

After permuting the environment, the optimal policy takes longer to converge and, in some scenarios, where the permutation is dramatic with higher probability of change, the optimal policy could be more complex. However, the algorithm is robust to these changes.

We also researched and investigated more and implemented Monte Carlo algorithm, for the dynamic environment and we concluded that the optimal policy using both algorithms, was the same.