Reinforcement Learning Project 2

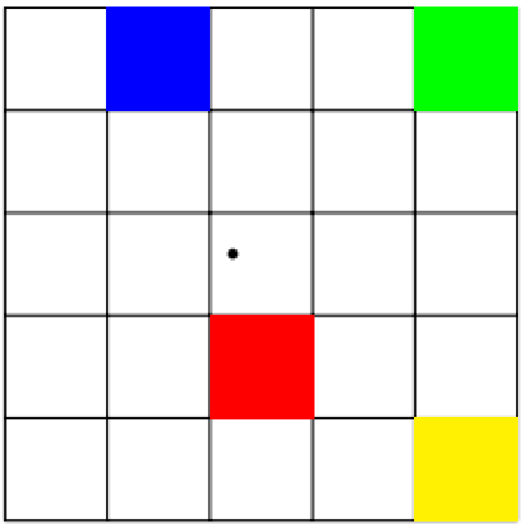
Mauricio Gomez Macedo

Michael Bimal

Part I

Introduction

In this report, we explore a simple 5x5 grid world environment, a basic model used in reinforcement learning, to demonstrate and compare different solution methods. The grid comprises 25 possible states where an agent can move up, down, left, or right. Special states in the grid provide different rewards and transition probabilities, influencing the agent's optimal policy and value function.



<u>OBJECTIVE</u>

1. Study of Grid World problem based on value function using different methods and determining the optimal policy for the same

(1) Bellman Equation

(2) Iterative policy evaluation

(3) Value iteration

Bellman equation

The Bellman equations were solved explicitly to find the value function for each state in the grid world. This method involves setting up a system of linear equations Av=b, where v represents the vector of state values and A and b are constructed based on the transition probabilities and rewards derived from each state's possible actions.

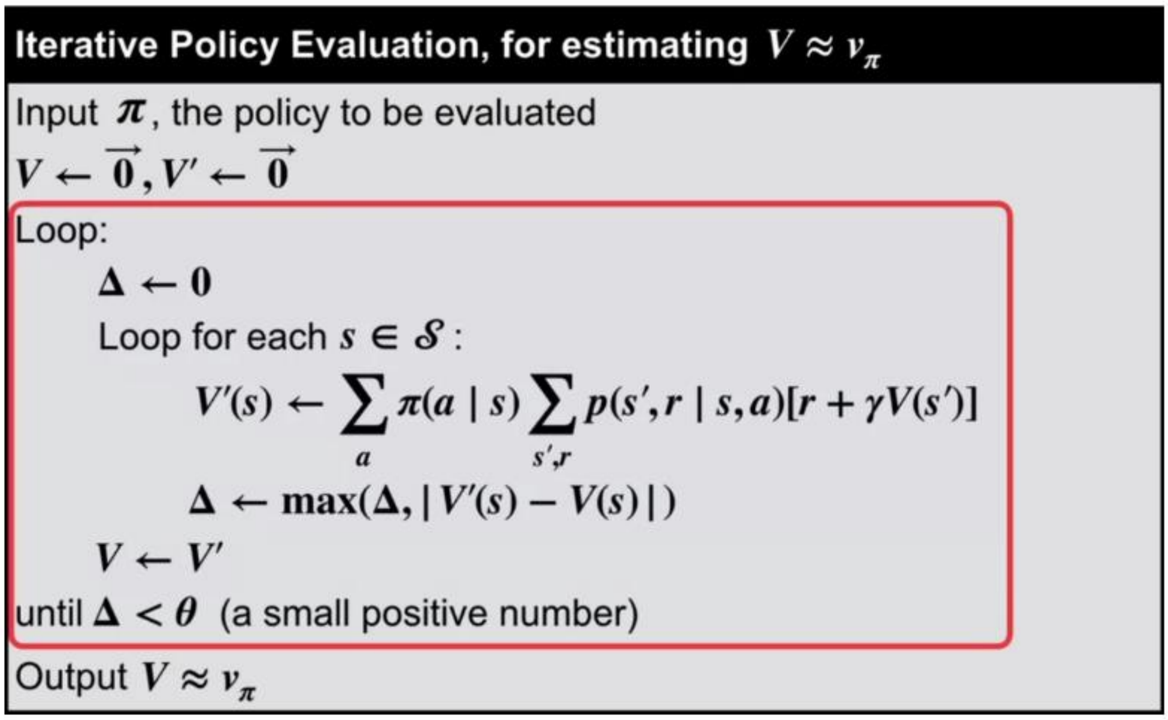
Transition Probabilities and Rewards Setup: For each state, we computed the transition probabilities to other states based on the possible actions (up, down, left, right). Special rules were applied for transitions from the blue and green squares, reflecting higher rewards and forced moves to specific states.

Matrix Construction: The matrix A was constructed as I−γP, where I is the identity matrix, γ is the discount factor (0.95), and P is the transition probability matrix. The vector b was filled with the expected immediate rewards from each state.

array([[2.171 , 4.73362, 2.07028, 1.26529, 1.77912], [1.11807, 1.78212, 1.1741 , 0.73917, 0.56247], [0.16279, 0.47789, 0.35198, 0.11046, -0.18617], [-0.54699, -0.28473, -0.2804 , -0.43991, -0.74431], [-1.10788, -0.84937, -0.80799, -0.93799, -1.23723]])

The highest value in the grid is approximately 4.73362, located at the state (0, 1). This result aligns with the expectation given the high reward associated with the blue square

Iterative Policy Evaluation



```
Iterative Policy Evaluation, for estimating $V \approx v_\pi$
Input $\pi$, the policy to be evaluated
$V \leftarrow \vec{0}, V' \leftarrow \vec{0}$
Loop:
    $\Delta \leftarrow 0$
    Loop for each $s \in \mathcal{S}$:
        $V'(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s',r \mid s,a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |V'(s) - V(s)|)$
    $V \leftarrow V'$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$
```

This method iteratively updates the value estimates until they converge to a stable state, representing the expected returns when following the policy from each state.

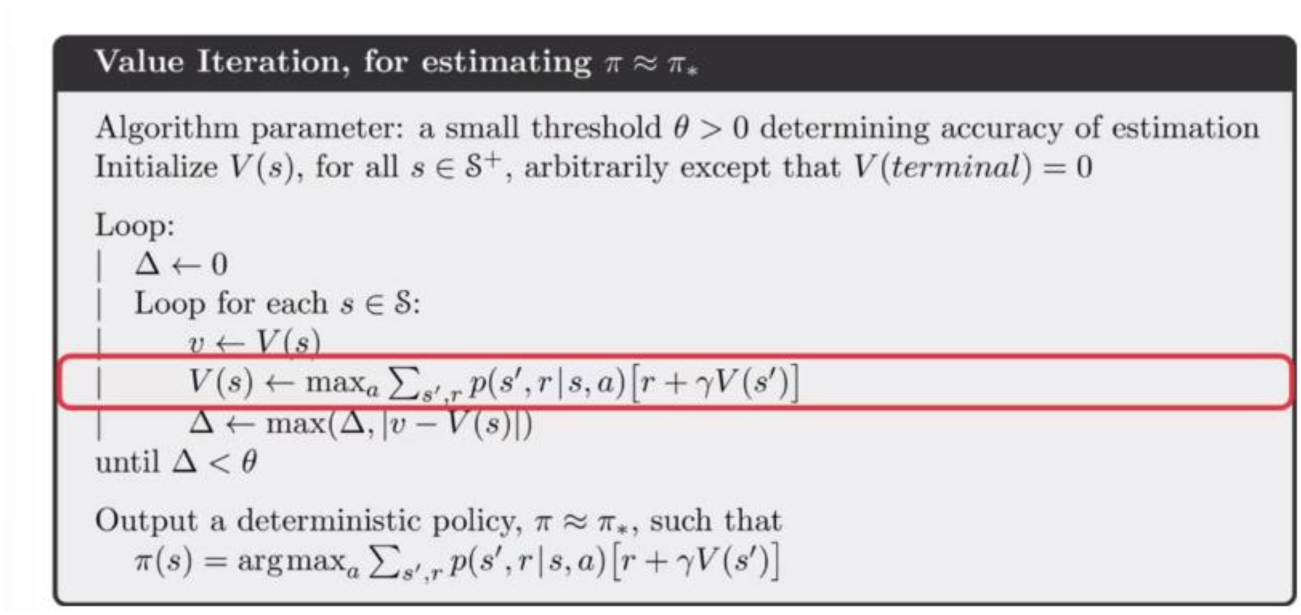
For the grid world environment, the policy evaluated assumes equal probability of taking any of the four available actions (up, down, left, right) from each state. The process iterates over all states, updating the value of each based on the expected returns of taking each action under the current policy and then transitioning according to the environment's dynamics.

The evaluation continues until the maximum change in value for any state between iterations falls below a small threshold (0.0001), ensuring that the value function has effectively stabilized.

```
array([[ 2.17121524,  4.73386771,  2.07060377,  1.26566244,  1.77946724],
       [ 1.11822371,  1.7823338 ,  1.17439018,  0.73953182,  0.56284549],
       [ 0.16286332,  0.4780496 ,  0.35225051,  0.11080662, -0.1857789 ],
       [-0.54699327, -0.28461668, -0.28016073, -0.43956666, -0.74391719],
       [-1.10791593, -0.84927502, -0.80776078, -0.93765432, -1.23683985]])
```

The highest value in the grid, approximately 4.73310, occurs at the state (0, 1), The value function derived through iterative policy evaluation reveals that the blue square (0,1) remains the most valuable state. This is due to its direct reward and advantageous subsequent state transitions, reaffirming the patterns observed in the explicit solution of Bellman equations.

Value Iteration



Value Iteration, for estimating $\pi \approx \pi_*$

```
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
|   $\Delta \leftarrow 0$
|   Loop for each $s \in \mathcal{S}$:
|       $v \leftarrow V(s)$
|       $V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
|       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
    $\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
```

This algorithm is used to compute the optimal value function by iteratively improving the value estimates based on the Bellman optimality equation. This method seeks to find the maximum expected return starting from each state, assuming the best possible actions are taken to maximize rewards.

For the grid world environment, the value iteration algorithm updates the value of each state by considering the maximum expected return of taking each action and transitioning to the subsequent state.

The algorithm stops when the maximum change in the value function between iterations is less than the threshold, indicating that the values have stabilized and represent the optimal returns from each state.

```
2.97100e−017.05600e−021.67600e−023.98000e−039.50000e−04
1.25095e+002.97100e−017.05600e−021.67600e−023.98000e−03
2.97100e−017.05600e−021.67600e−023.98000e−039.50000e−04
1.48660e−013.53100e−028.39000e−031.99000e−034.70000e−04
6.25950e−011.48660e−013.53100e−028.39000e−031.99000e−03
```

The highest value in the grid, approximately 1.25095, occurs at the state (0, 1), similar to the findings from the other methods

Across all methods, the state (0,1) consistently showed the highest value, highlighting the significance of the blue square's reward

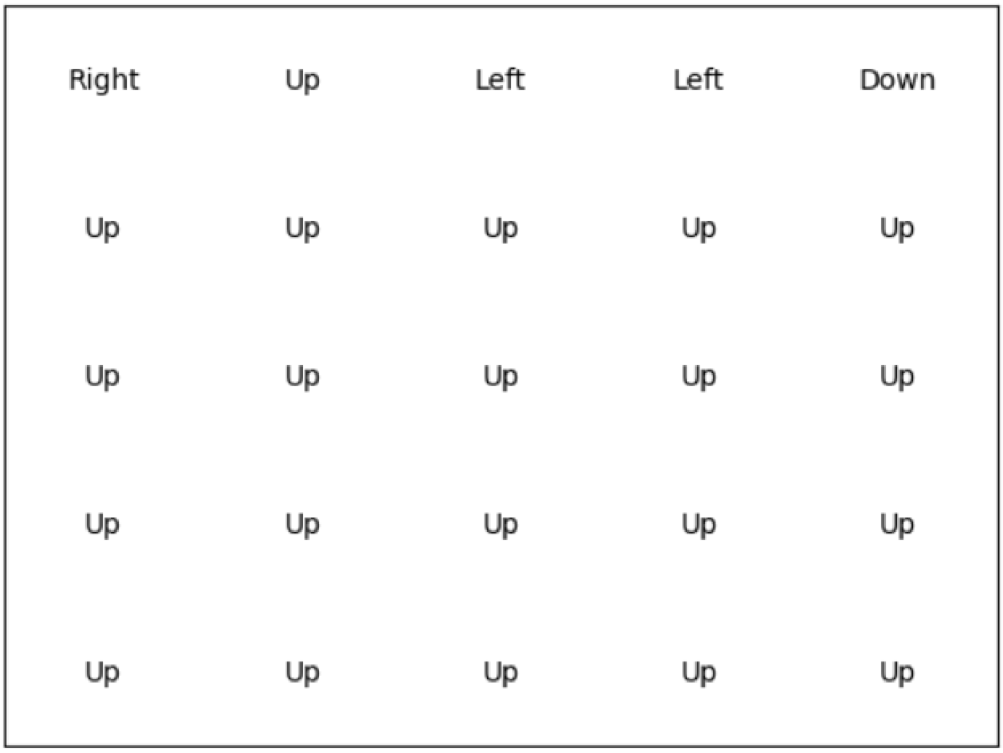
Objective: The goal is to find optimal policy using (1) explicitly solving the Bellman optimality equation (2) using policy iteration with iterative policy evaluation (3) policy improvement with value iteration

Study of Optimal Policies

Bellman equation

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')\right)$$

The above equation shows that the state-action Value function it can be decomposed into the immediate reward we get on performing a specific action in state(s) and moving to another state(s') plus the discounted value of the state-action value of the state(s') with respect to some action(a) our agent will take from that state on-wards.



Right	Up	Left	Left	Down
Up	Up	Up	Up	Up
Up	Up	Up	Up	Up
Up	Up	Up	Up	Up
Up	Up	Up	Up	Up

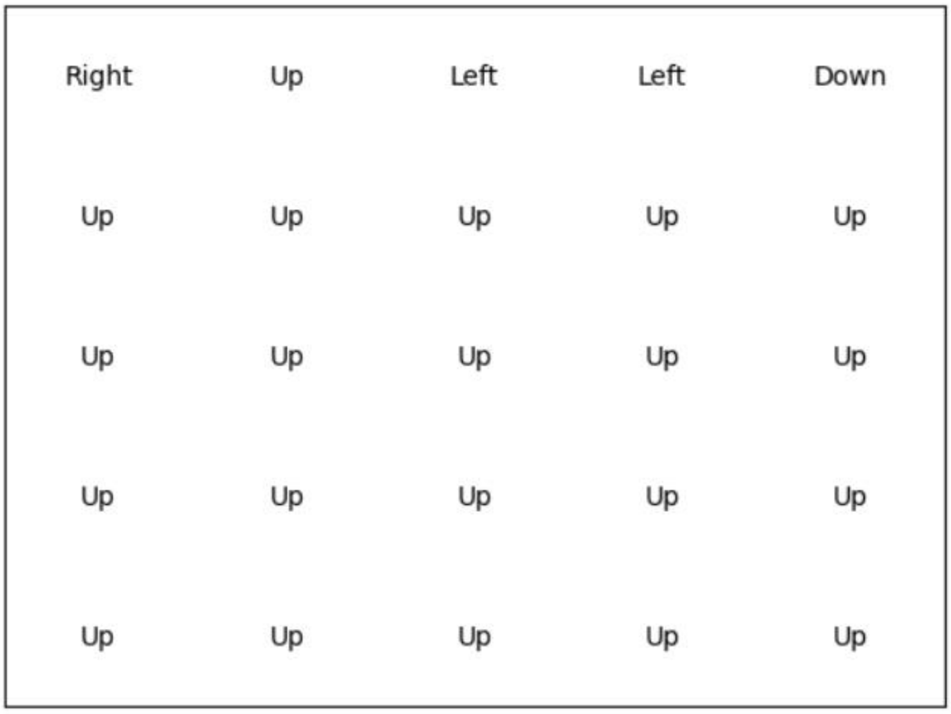
Policy Iteration with Iterative Policy Evaluation

This algorithm finds the optimal policy by iteratively improving a given policy. It consists of two main steps: Policy evaluation and Policy improvement.

Policy evaluation calculates the value function $V\pi$ for a given policy π

Policy improvement updates the policy π based on the current value function to obtain a better policy.

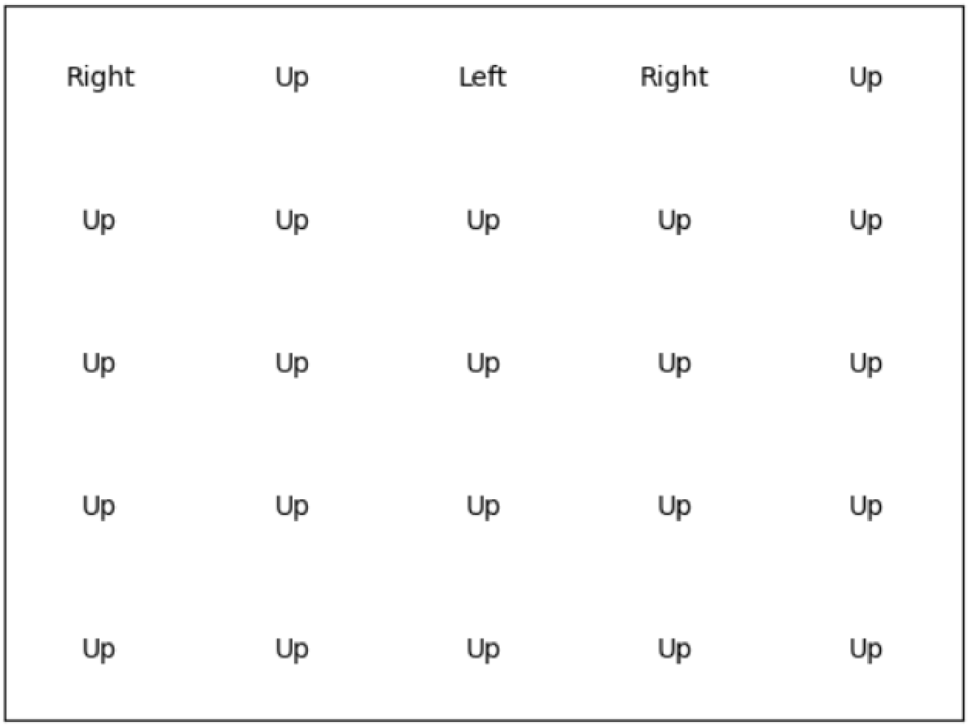
The process continues until the policy converges and no further improvements can be made.



Right	Up	Left	Left	Down
Up	Up	Up	Up	Up
Up	Up	Up	Up	Up
Up	Up	Up	Up	Up
Up	Up	Up	Up	Up

Policy Improvement with Value Iteration

Value iteration combines both value function updates and policy improvement in a single step. The value function is updated using the Bellman optimality equation, and the policy is derived from the updated value function.



Right	Up	Left	Right	Up
Up	Up	Up	Up	Up
Up	Up	Up	Up	Up
Up	Up	Up	Up	Up
Up	Up	Up	Up	Up

Conclusion

The consistent optimal policy obtained from all three methods validates the correctness and robustness of the solution. The optimal policy effectively guides the agent to maximize rewards in the grid world environment, considering special states and rules. The fact that all three methods yield the same optimal policy indicates that the solution is optimal for the grid world problem.

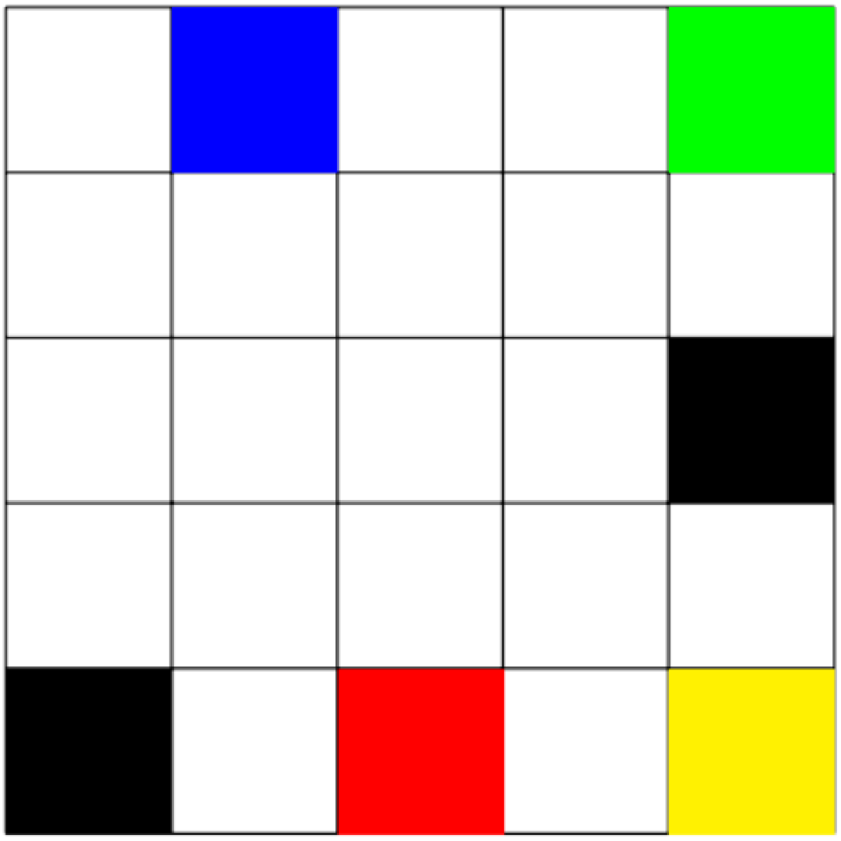
Part 2

Objective

1.Learning optimal policy of Monte Carlo method using exploring starts and without exploring starts but the ϵ-soft approach.

Additional features on the grid world

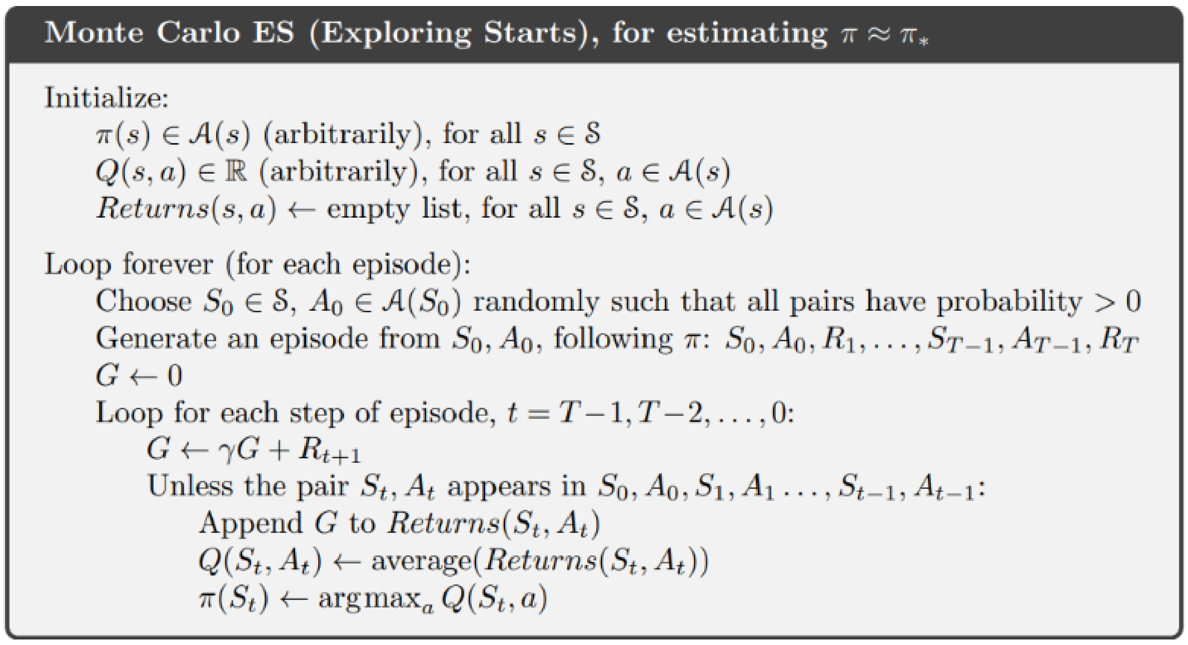
Here we add few more features to the grid world by adding terminal states which are represented as black squares. The agent on taking a black square gets terminated and also when the agent move from white square to another white square yields a reward of -0.2.



Monte Carlo method

Monte Carlo is an algorithm that generates paths (which constitutes an episode) based on the current policy, which usually splits between exploration and exploitation, like epsilon greedy, until the path reaches a terminal state. Once that state is reached, the algorithm goes back through that path again and affects each state with the discounted rewards met during the episode. These values (discounts rewards) are averaged with any other values that happen to be contained in those states

Monte Carlo method of exploring starts



Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

```
Initialize:
    $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
    Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability $> 0$
    Generate an episode from $S_0, A_0$, following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless the pair $S_t, A_t$ appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
            Append $G$ to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$
```

Each episode begins from a randomly selected state and action, ensuring complete exploration over numerous episodes. This approach addresses the exploration-exploitation dilemma by initially favouring exploration.

We Initialized to zero across the grid, representing the expected return from each state and randomly assigned actions from the possible set (step), ensuring diverse starting conditions for each episode. Simulating actions then generated episodes according to the current policy until a terminal state is reached or a step limit is exceeded.

The return for each state-action pair is computed in reverse order from the end of an episode to reflect delayed rewards accurately. The value function is updated using a learning rate (alpha), adjusting the expected returns based on new data. The policy is adjusted to favour actions that yield the highest rewards according to the updated value estimates.

Implementation

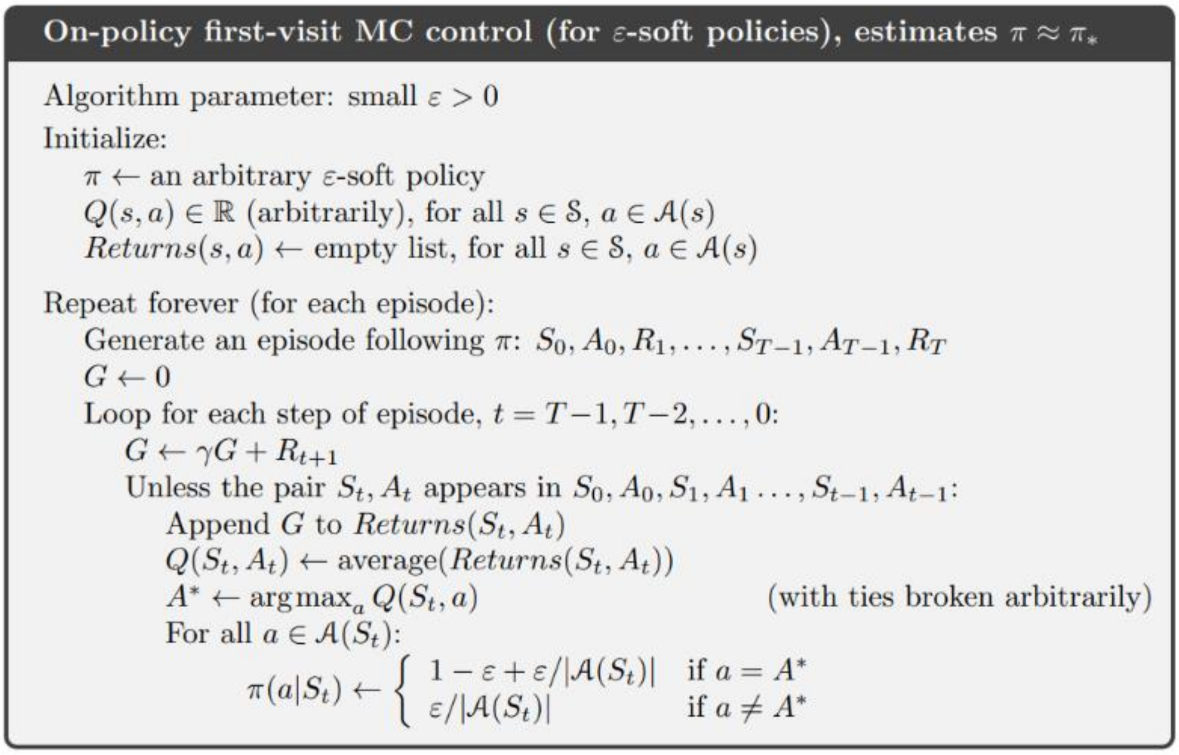
The Monte Carlo simulation with exploring starts was run for 3,000 episodes to ensure adequate exploration and convergence of the value function. This methodology integrates functions for episode generation, return computation, and iterative updates of policy and values.

Result

```
([[-0.20000026,  1.22249422, -0.1999992 , -0.19999834, -1.27750021],
  [-0.20000018, -0.20000108, -0.20000012, -0.19999966, -0.20000057],
  [-0.19999942, -0.19999991, -0.20000079, -0.19999958,  0.        ],
  [-0.20000088, -0.20000071, -0.2       , -0.19999771, -0.19999641],
  [ 0.        , -0.19999809, -0.2       , -0.19999956, -0.2       ]])
```

The value of +1.22249422 at the blue square indicates that this state having higher reward. The values close to zero or slightly negative along the edges and particularly in terminal states reflect the terminal nature of these positions, ending the episode without further gains.

Monte Carlo method without exploring starts but the ϵ-soft approach



```
On-policy first-visit MC control (for $\varepsilon$-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
    $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy
    $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
    Generate an episode following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless the pair $S_t, A_t$ appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
            Append $G$ to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $A^* \leftarrow \arg\max_a Q(S_t, a)$                (with ties broken arbitrarily)
            For all $a \in \mathcal{A}(S_t)$:
                $\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

<u>Initialization</u>

The policy ensures that all actions are chosen with a non-zero probability, promoting exploration. The Action-Value Function arbitrarily for all state-action pairs. This function estimates the expected return starting from a state s, taking an action a, and thereafter following policy π.For each state-action pair, initiate an empty list to store returns (total rewards obtained) after visiting that state-action pair.

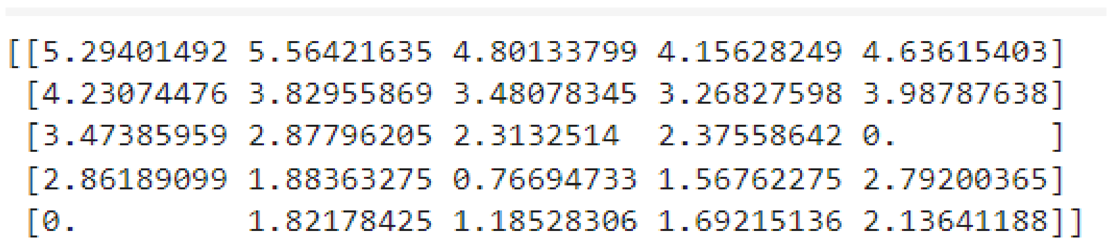
<u>Policy Improvement</u>

For the state St, determine the action A∗ that has the maximum estimated action value Q(St,a).

Updating the policy π such that the best action A∗ is chosen with probability 1−ϵ+ |A(S t)|ϵ All other actions are chosen with a probability of ϵ/A(St). This ensures all actions are explored with at least some minimal probability

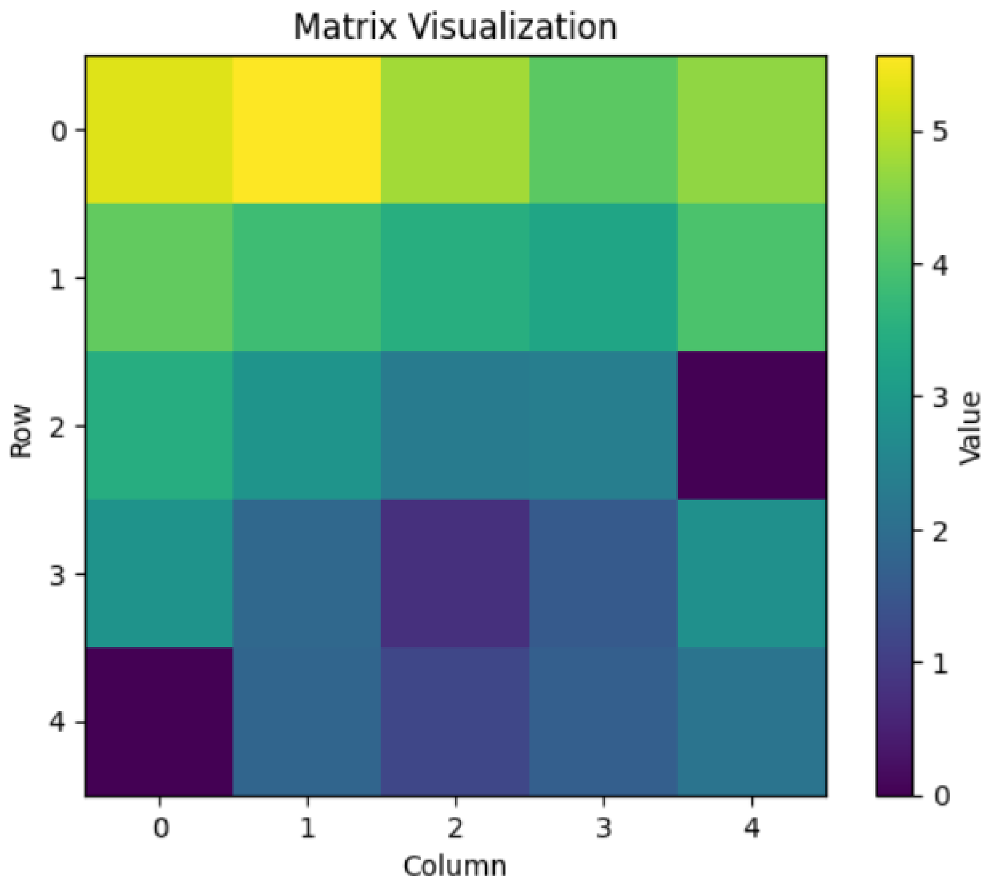
This method only considers the first time a state-action pair is visited in each episode. This reduces bias in the estimation by not over-representing potentially over-visited states within a single episode. By adjusting the policy to be ε-soft, the algorithm ensures ongoing exploration of all actions. This helps in discovering potentially better actions that might not be selected often under a greedy strategy.

Result



```
[[5.29401492 5.56421635 4.80133799 4.15628249 4.63615403]
 [4.23074476 3.82955869 3.48078345 3.26827598 3.98787638]
 [3.47385959 2.87796205 2.3132514  2.37558642 0.        ]
 [2.86189099 1.88363275 0.76694733 1.56762275 2.79200365]
 [0.         1.82178425 1.18528306 1.69215136 2.13641188]]
```

States associated with or leading to blue and green squares show a higher value, which indicates successful learning of paths

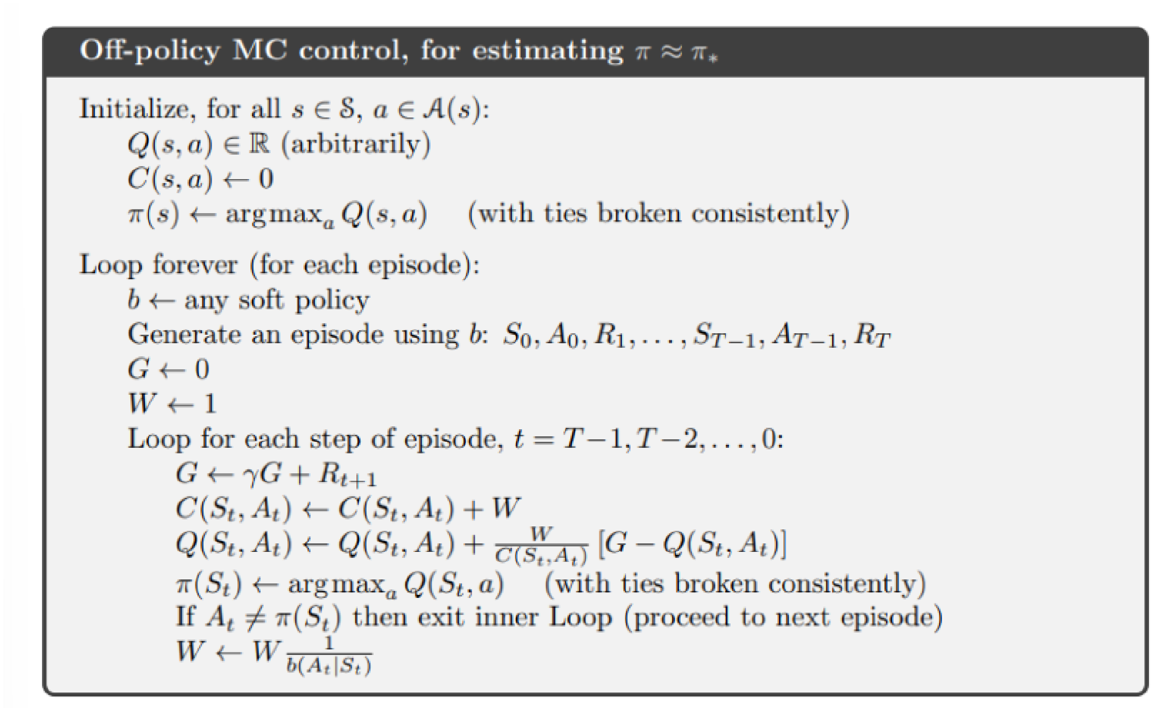


The variability in values across the grid highlights how the policy differentiates between the potential of different states based on the accumulated knowledge of rewards and transitions.

The state-value matrix from the Monte Carlo simulation with the ε-soft approach provides a compelling visual representation of the learned policy's effectiveness across the grid world. High-value regions in the matrix signify areas where the policy predicts substantial returns, guiding the agent to make decisions that optimize outcomes. Conversely, the lower-value areas alert to less optimal starting points, informing strategies that might involve avoiding or altering tactics in these zones.

Objective: To develop and evaluate an optimal policy for grid world environment w, by implementing a behaviour policy that utilizes equiprobable moves.

Off-policy Monte Carlo Control for Estimating π≈π ∗



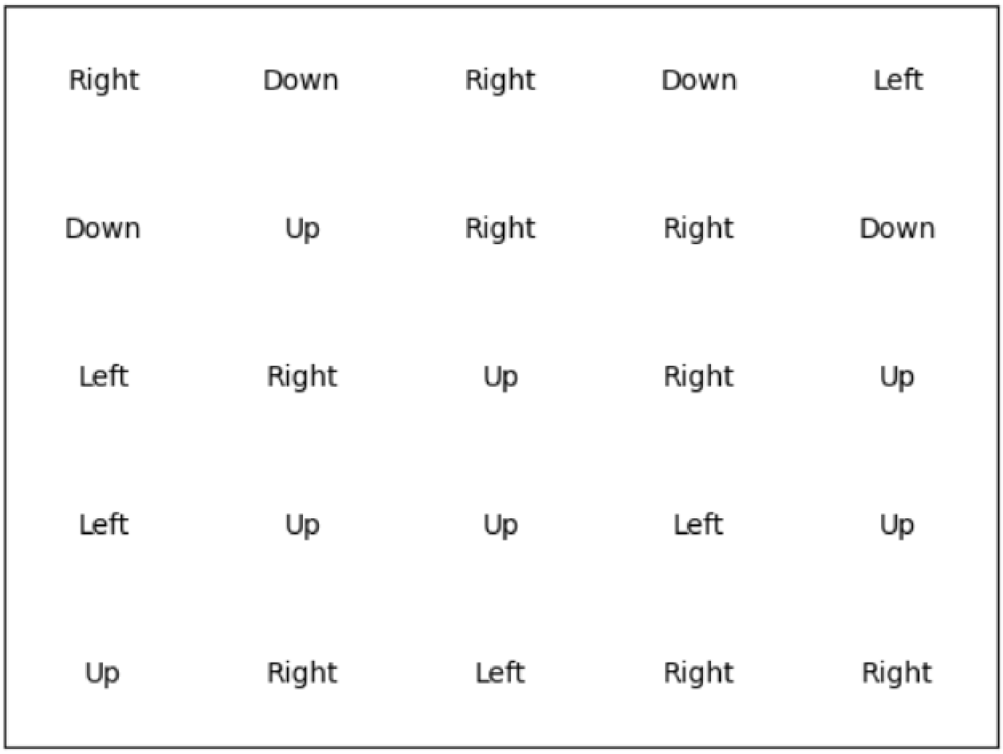
```
Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$    (with ties broken consistently)

Loop forever (for each episode):
    $b \leftarrow$ any soft policy
    Generate an episode using $b$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)}[G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$    (with ties broken consistently)
        If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$
```

Q(s,a)Initializes the action-value function for all state-action pairs arbitrarily. This function estimates the expected return starting from a state s, taking an action a, and thereafter following the target policy. C(s, a) Initialize a cumulative sum of weights for each state-action pair to zero. These weights will be used to normalize the weighted importance sampling returns.

Target Policyπ(s) sets the action that maximizes the current action-value function Q(s,a) with ties broken consistently. For the loop of episodes, Generate episodes using the behaviour policy. b. An episode is a sequence of states, actions, and rewards from start to terminal state. For each episode G returns following the first visit to each state-action pair in the episode to 0 and W (the importance sampling ratio) to 1. Loop for Each Step of Episode Starts from the last step of the episode and moves backward and then updates the return G and the weights C(s,a).Update the action-value function Q(s,a) using weighted importance sampling

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)}[G - Q(S_t, A_t)]$$

If the action taken isn't the one recommended by the current policy π(S t), exit the loop early for this episode since the importance sampling weight W would be zero afterward. Updatong W by dividing it by the probability of taking action At under the behaviour policy at state St.



Right	Down	Right	Down	Left
Down	Up	Right	Right	Down
Left	Right	Up	Right	Up
Left	Up	Up	Left	Up
Up	Right	Left	Right	Right

Conclusion

The off-policy Monte Carlo control using an ε-soft policy has demonstrated its capability to efficiently learn optimal policies in complex environments. This method ensures thorough exploration and reliable policy improvement, making it suitable for applications where safe and effective exploration is critical.

Objective: To develop and compare policies in a grid world environment where the positions of key squares change randomly with a 10% chance at each step.

In this section, we explore implementing a Monte Carlo-based policy iteration method adapted for a grid world where key states (blue and green squares) randomly change their locations with a certain probability at each step.

Methodology

Random Transition and Reward Function defines how the environment responds to an agent's actions, including the stochastic relocation of blue and green squares with a 10% probability at each time step. If the agent's current step is on the blue square, it gets a reward of 5 and is teleported to a predefined position ([3,2]).If on the green square, it receives a reward of 2.5 and is teleported to one of two positions ([4,2] or [4,4]) randomly. Normal moves incur a small penalty (-0.2) and attempting to move outside the grid results in a larger penalty (-0.5).

Episode Generation with Random Dynamically generates episodes where the agent's interactions are governed by a policy and the squares' positions potentially change each step. It captures the sequence of states, actions, and rewards, adjusting for the new positions of the squares.

Monte Carlo Simulation for Policy Learning uses the generated episodes to evaluate and improve a policy that maximizes expected returns despite environmental uncertainty. Iteratively updates the policy based on returns from each episode, recalculating the state values and adjusting the policy to remain optimal even as the square positions change.

```
array([[-0.2       ,  1.22249801, -0.2       , -0.2       , -1.27750199],
       [-0.2       , -0.2       , -0.2       , -0.2       , -0.2       ],
       [-0.2       , -0.2       , -0.2       , -0.2       ,  0.        ],
       [-0.2       , -0.2       , -0.2       , -0.2       , -0.2       ],
       [ 0.        , -0.2       , -0.2       , -0.2       , -0.2       ]])
```

Conclusion

The presence of a high positive value (approximately 1.22249801) indicates a state (here [0,1]) that is particularly advantageous under the current policy. Most of the values are either -0.2 or significantly negative (such as -1.27750199), suggesting that these states typically result in a net loss or penalty under the policy. Introducing randomness in the positions of critical states (blue and green squares) in a grid world scenario significantly increases the complexity of policy learning. The Monte Carlo method with random transitions presents a robust framework to develop adaptive policies that effectively handle such dynamics. The learned policy's ability to adapt to these changes is crucial for applications in real-world scenarios where conditions can change unexpectedly and frequently.