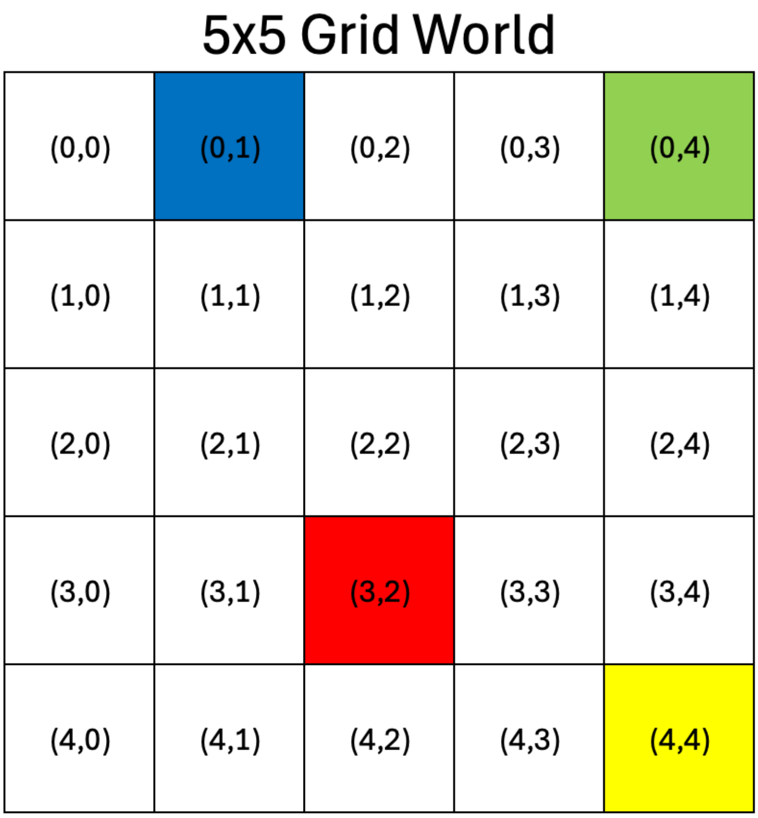
Reinforcement Learning: Grid world

Charuka Amarappulige | Chinonso Imochukwu

Part 1

In this section we take 5 x 5 grid world with specific characteristics to compare different reinforcement learning methodologies. Property of grid world environment as below. Agent in environment can take a step up, down, left or right actions. However, If the agent attempts to step off the grid, the location of the agent remains unchanged with reward of −0.5.
The blue, green, red and yellow squares represent special states. At the blue square, any action yields a reward of 5 and causes the agent to jump to the red square. At the green square, any action yields a reward of 2.5 and causes the agent to jump to either the yellow square or the red square with probability 0.5. Taking any action in white, red and yellow squares will not yield any reward unless it is not trying to step off the grid.
For location identifying purpose each state is numbered as below figure in most of the cases unless different requirement arises.



Question 1

1. Solving the system of Bellman equations explicitly

In order to solve bellman equation fixed policy π and solve it explicitly, we have broken down original Bellman equation.

$$V(s) = \sum_{s\prime \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + \gamma V(s')]$$

Writing the equation in matrix from

$$V = R + \gamma P V$$

where R and P are expected reward matrix and transition probability matrix according to given policy.

$$R(s) = \sum_{s\prime \in S} P(s'|s, \pi(s)) R(s, \pi(s), s')$$
$$P(s, s') = P(s, \pi(s), s')$$

By calculating the R and P we get.

Transition probability from one state another $P = \begin{bmatrix} 0.5 & 0.25 & \cdots & 0 \\ 0.25 & \ddots & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0.5 \end{bmatrix}_{25x25}$

Expected reward for each state $R = \begin{bmatrix} -0.25 \\ 5 \\ \vdots \\ -0.25 \end{bmatrix}_{25x1}$

Rearranging the equation

$V = R + \gamma P V \implies (I - \gamma P)V = R \implies V = (I - \gamma P)^{-1} R$

Functions used

get_transition_rewards

Objective of this function is to calculate transition probabilities, rewards, expected value and individual values for each action once it is once it was given the state, policy and given value function.

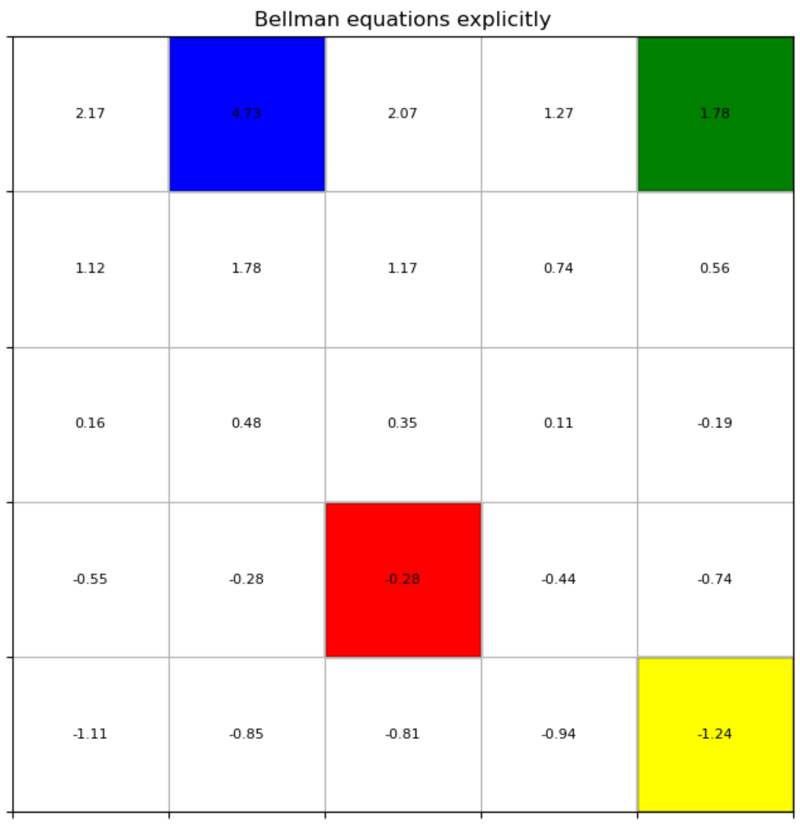
plot_gridworld

This function is used to plot the gridworld and its values accordingly once finalized value given.

SciPy.linalg

Module is used to calculate matrix inverse and multiplication.

Once value function is solved finalized values (V_π) rounded up to two decimal places is looks like in below figure. Highest valued state came as blue state while rest of the values distributed toward rest of the area.



2. Iterative policy evaluation

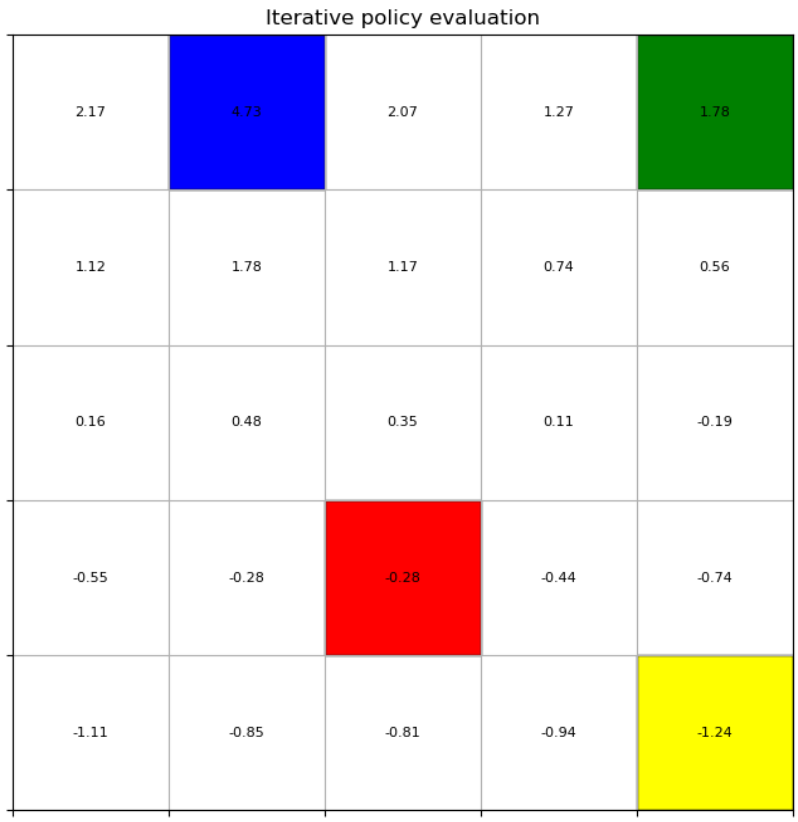
In order to calculate (V_π) using iterative policy evaluation we have used the below algorithm extracted from Chapter 4.1 of Reinforcement Learning book by Richard S. Sutton and Andrew G. Barto. The same ‘get_transition_rewards’ function being used to calculate expected values in each state

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
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 $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

The output (V_π) came as below figure which is identical to the output in pervious case.



3. Value iteration

In this section we have tried to calculated value function using value iteration methodology similarly we have used algorithm extracted by the Reinforcement Learning book chapter 4.4. How ever it was obvious that value iteration method is not suitable to evaluate a policy. It is much suitable to calculate optimum value function.

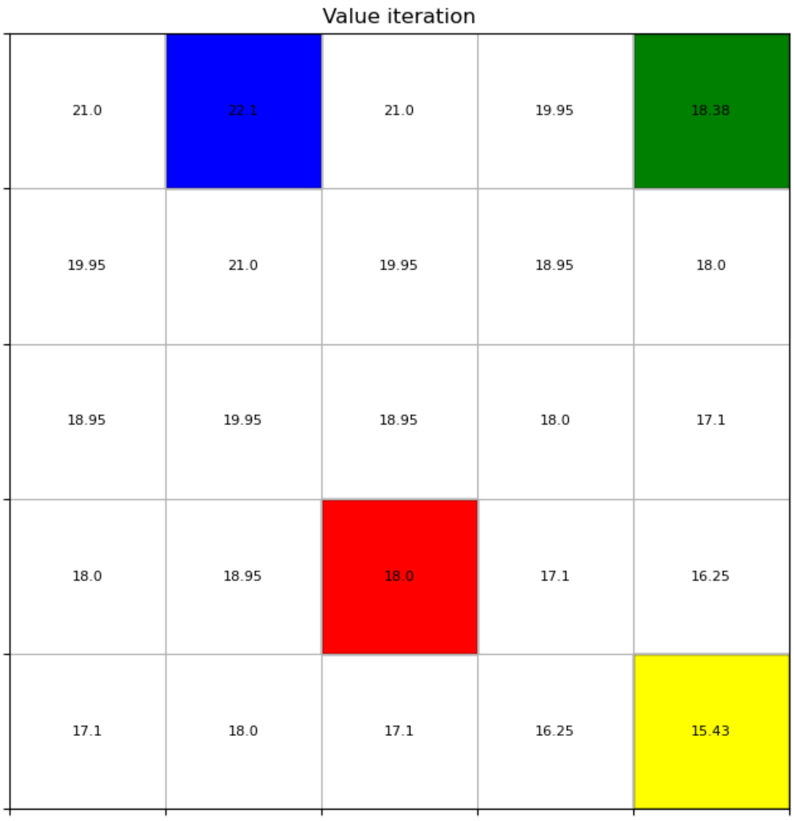
Value Iteration, for estimating $\pi \approx \pi_*$

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

The results for value iteration function can be illustrate as below figure



From all the three algorithm it was very clear that highest valued sate is state (0,1) which is blue demarcated state. It was not a surprise that since any action taken from blue state incur reward of 5 to the agent. Due to this reason, it is quite obvious the highest valued state is the blue state.

You can recreate the above results from referring to the file A2P1Q1.ipynp and it is available in the below git hub repository

Link: https://github.com/CharukaP/RL_Gridworld

Question 2

1. Optimum policy by explicitly solving the Bellman optimality equation

This segment explain how optimum policy is obtained by explicitly solving bellman equation. Two methods being used to check the possibility of solving this problem. Two new function being used in order to support this evaluation. In method 1 we have carried out the solution in four steps.

Functions used

get_next_state_and_reward

This function work on calculating the reward and the next state according to the environment characteristics once it was given the current state and the action.

action_search

This function will return concatenated actions string if more than one action is optimum.

Method 1

Step 1: Adjustment to Bellman optimality equation

The Bellman equation for a state s can be written as:

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

Adjusted Bellman equation.

$$V(s) - max_a(R(s,a) + \gamma \sum_{s\prime} P(s'|s,a)V(s')) = 0$$

Step 2: Genarate 25 equation for each 25 states.

Using the adjusted Bellman equation generate 25 equations.

Step 3: Solve the set of equations

Solving the system of equations using the ‘fsolve’ function from SciPy will give you the V*

Step 4: Solve the set of equations

Use calculated V* and perform one step look ahead action to identify optimum action π*

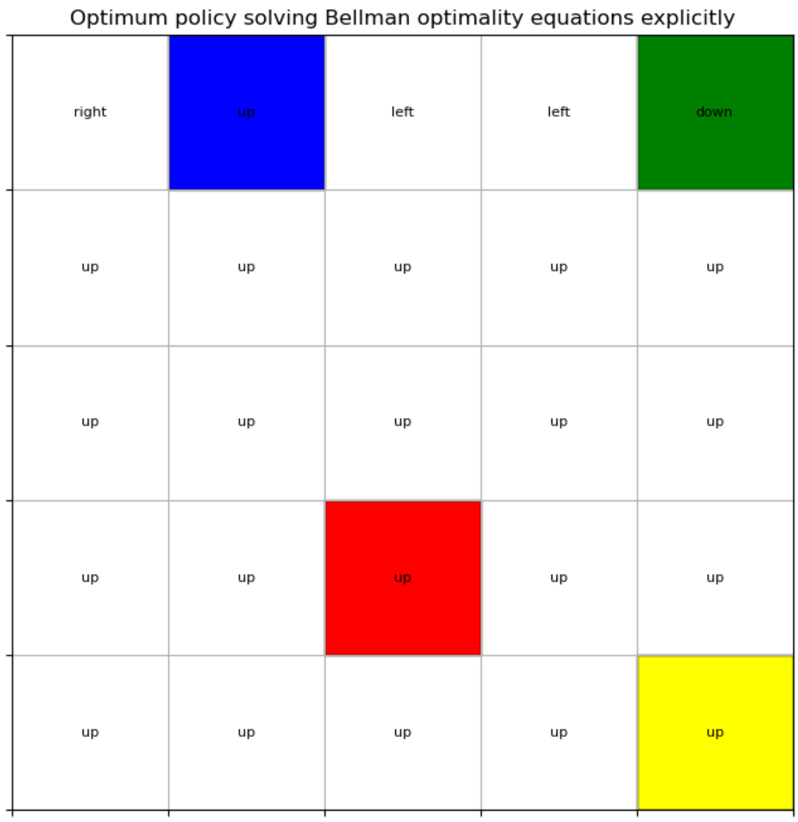
Once calculated the optimum actions can be illustrate as below figure. Optimum policy is leading the agent towards the blue state obviously.

Method 2

In method 2 we used the same marix solution we used for part1 question 1 section with slight change to reward matrix here we used reward as maximum reward instead of expected reward. As below

$R(s) = max_a R(s, a)$

Solving $V = (I - \gamma P)^{-1}R$ equation with SciPy.linalg module to calculate matrix inverse and multiplication. We were able to achieve the below results. Code for the second methodology is uploaded in git hub link under special file name A2P1Q2_part1_optional.ipynb



2. Policy iteration with iterative policy evaluation

Calculation of optimum policy using iterative policy evaluation. Was carried out according to algorithm extracted by Chapter 4.3 in Reinforcement Learning book where the algorithm can be formulated as below figure.

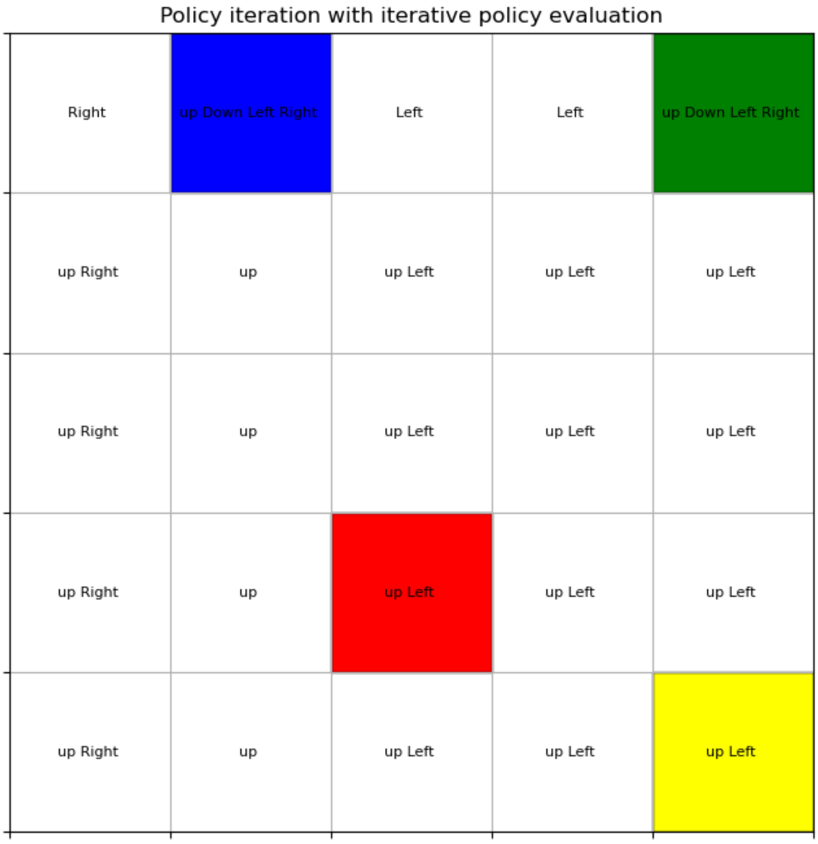
Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Once V* being calculated we used multiple action searching function to identify action with same values. The results are illustrated in below figure.



3. Policy improvement with value iteration

Identify the optimum policy via value iteration just extension of last sub question of Question 1. Since we have already calculated the V* values it's just a matter of carrying out one step look had and get the optimal policy. We have used the extension of algorithm extracted from same 4.4 chapter in RL book.

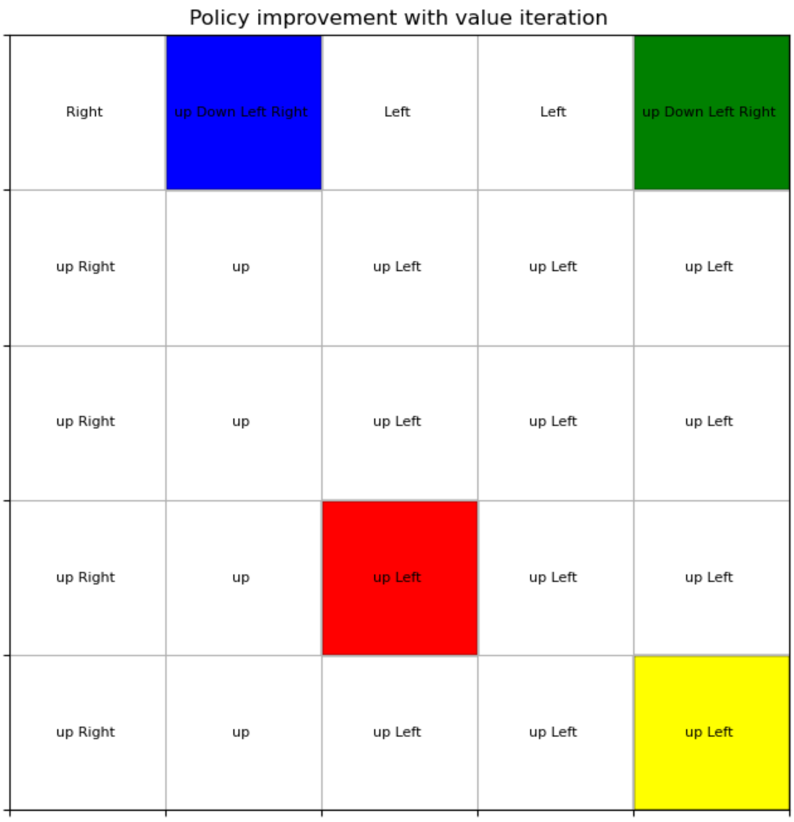
Value Iteration, for estimating $\pi \approx \pi_*$

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

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

The results show the similar to the second part where π* leads all the action towards the blue state.

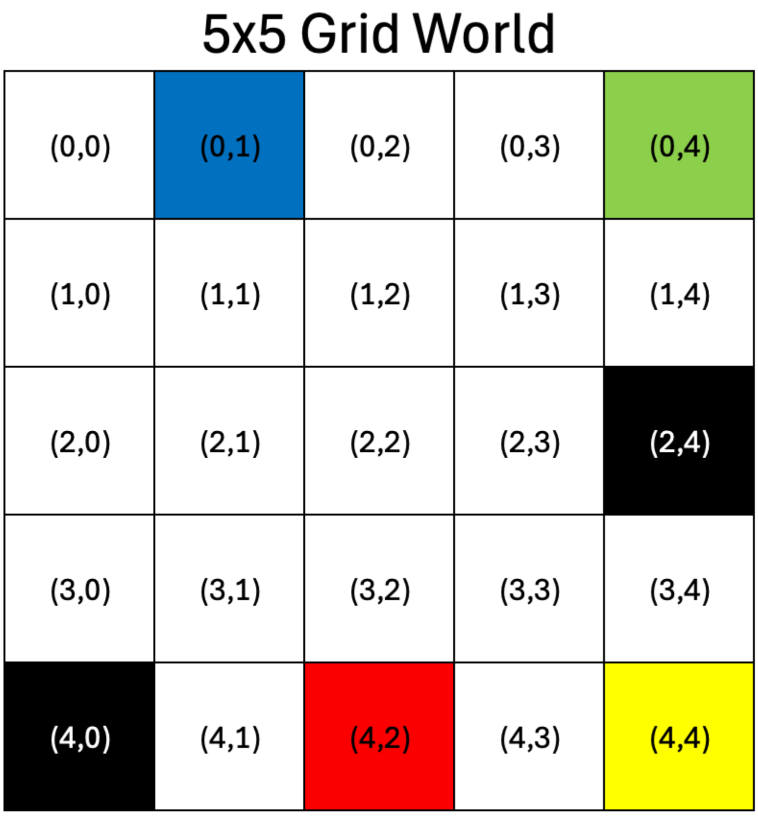


You can recreate the above results from referring to the file A2P1Q2.ipynp and it is available in the below git hub repository

Link: https://github.com/CharukaP/RL_Gridworld

Part 2

There have been slight changes into environment is introduced in this section. Two terminal states were introduced in black where episode terminate if agent visits that cell. Previous general movement incurred zero reward has changed to -0.2 compared to part 1. Same state identifying notations were carried out in here as well which is illustrated in below figure.



Question 1

Apart from 'get_next_state_and_reward' function and 'plot_gridworld' functions there are new supporting functions have been used to fulfill the task of finding optimum policy.

Functions used

Random_start

This function is used to get a random starting state except the terminal states

generate_episode

This function is used to generate series of steps according to equiprobable actions till agent reaches to terminal point and finally return the episode to main function.

choose_action

soft action selection was carried out from this function where it will receive state and policy, and it returns the action chosen

generate_episode_soft

This function used to generate series of steps till agent reach to terminal point while taking action from choose action function with soft policy.

1. Monte Carlo method with exploring starts

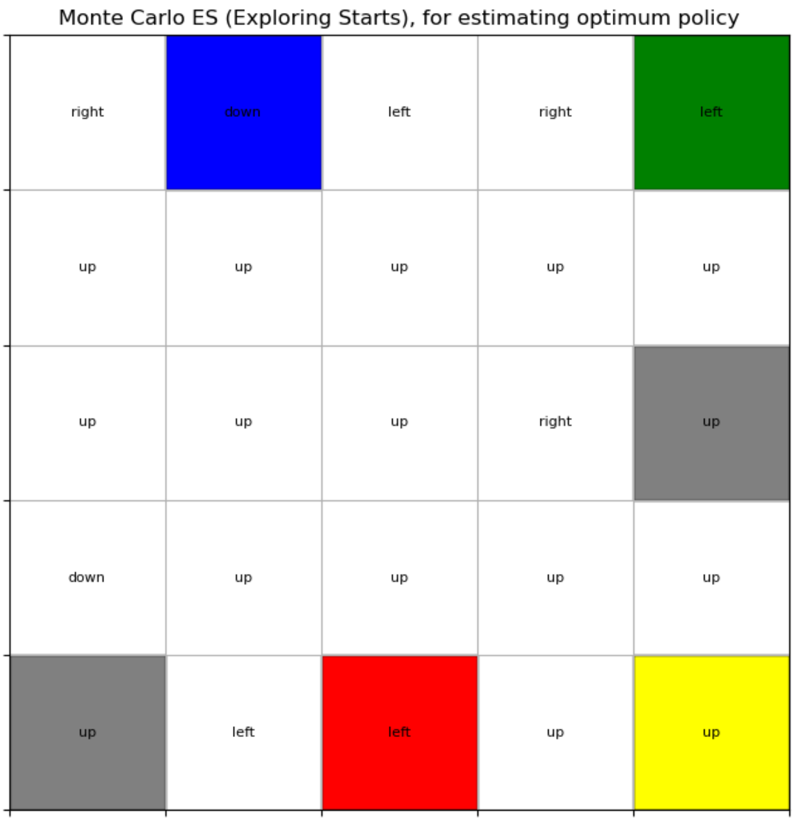
Usage of Monte Carlo method to identify optimal policy is explained in chapter 5.3 in Reinforcement Learning book and I have extracted the algorithm for it as in below figure.

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 π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  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 $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

Using γ=0.95 and running the algorithm will give you the optimum policy as below grid.



2. Without exploring starts but the ϵ-soft approach

We have used algorithm from chapter 5.4 as illustrated in below figure.

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

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-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 π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

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 $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

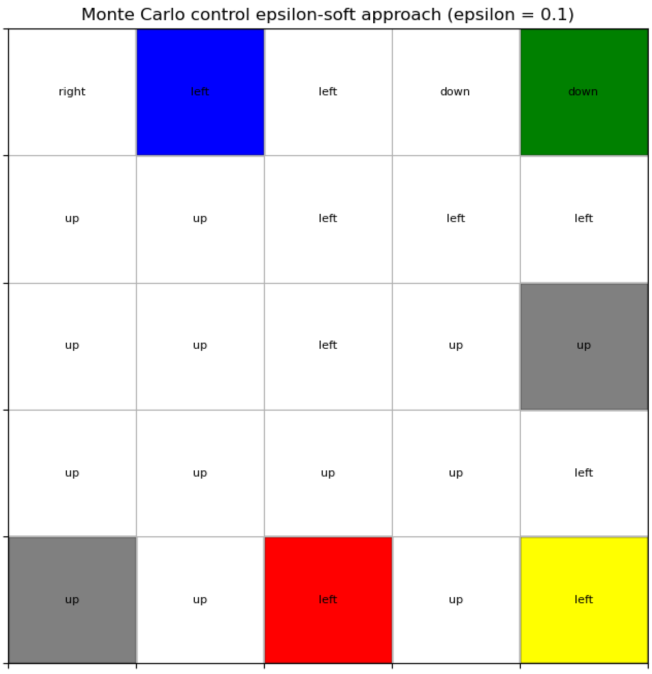
$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

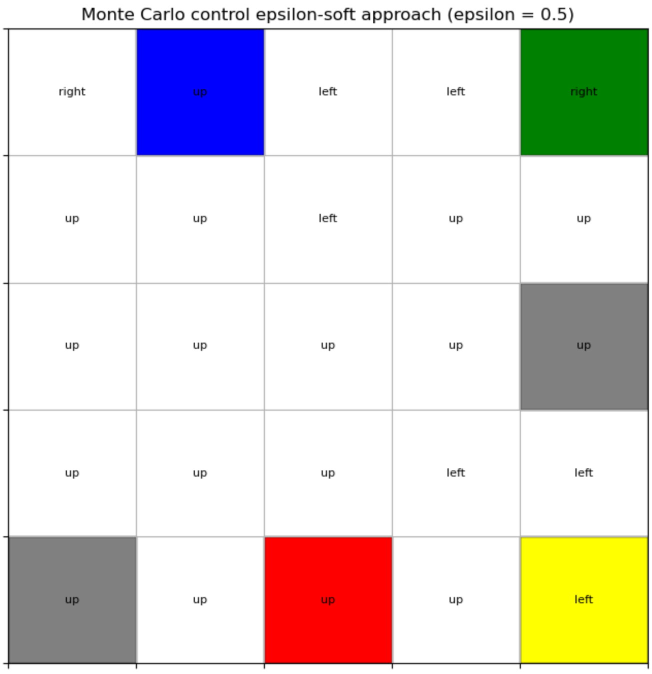
$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

In order to determine appropriate epsilon value multiple exercises were carried out with values 0.1, 0.5, 0.8 and 0.9 for 1000 episodes.

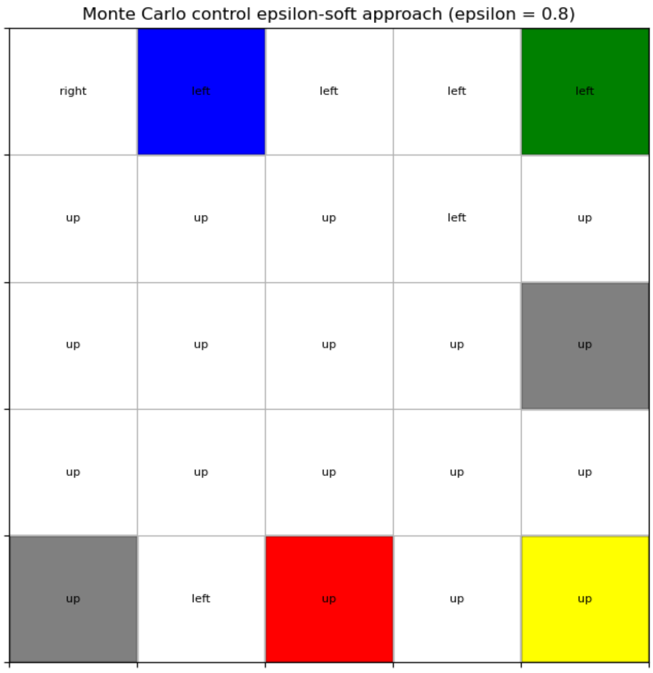
Epsilon = 0.1



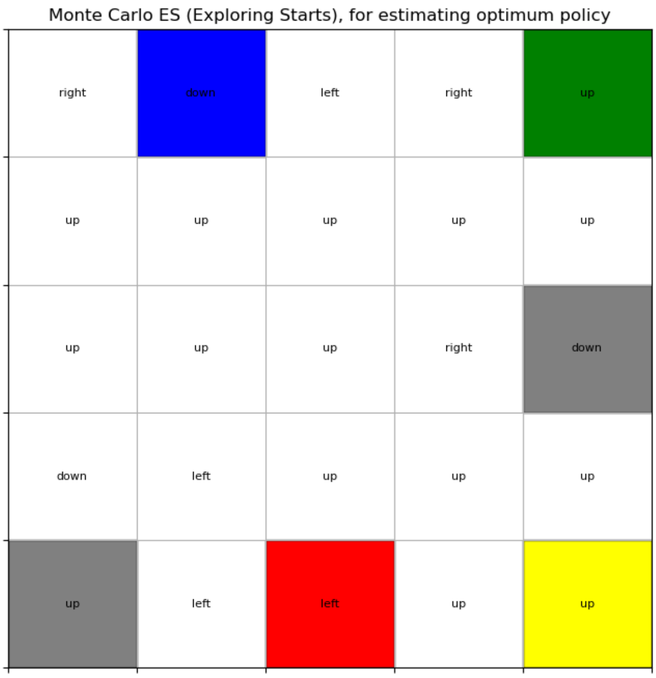
Epsilon = 0.5



Epsilon = 0.8



Epsilon = 0.9



From the multiple runs it was observed that Epsilon value closer to 1 (0.9) give the most accurate results compared to other evaluations.

You can recreate the above results from referring to the file A2P2Q1.ipynp and it is available in the below git hub repository

Link: https://github.com/CharukaP/RL_Gridworld

Question 2

This question was addressed using off policy Monte Carlo control methodology extracted from chapter 5.7 in Reinforcement learning book. Algorithm steps are mentioned in the below figure.

Due to constructed methodology in algorithm the learning rate is slow, and we need to use higher episode count to get near optimal policy. Running episodes for below 100,000 will not give an optimum policy. Furthermore, arbitrary starting value for Q table is taken as -3 after comparison with multiple experiments.

You can recreate the above results from referring to the file A2P2Q2.ipynp and it is available in the below git hub repository

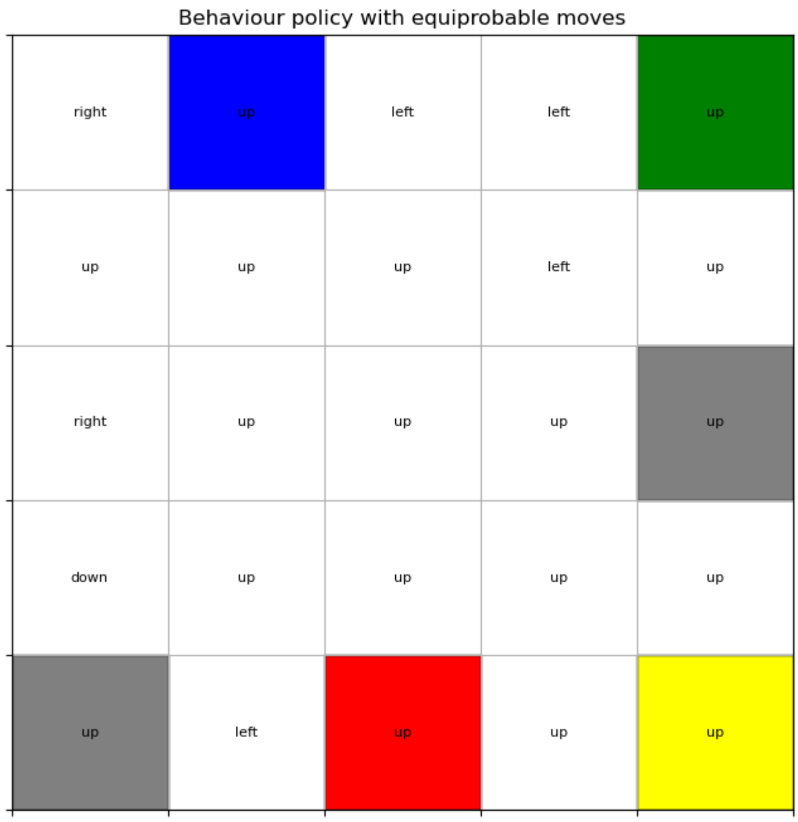
Link: https://github.com/CharukaP/RL_Gridworld

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)
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$

The results obtained by running algorithm for 10,000,000 episodes will give a policy similar to below grid. Due to high number of iteration required for get an accurate result this algorithm is not suitable for quick evaluations.



Question 3

In this segment new characteristic performance was introduced to gridworls where green and blue states interchange with each other with probability of 0.1 every time special function is called. It can be either steps or episode. We have introduced new function in order to facilitate this requirement. We have use two methods to evaluate policy

1. Policy iteration with iterative policy evaluation
2. Monte Carlo method with exploring starts

Functions used

permute_location

The primary objective of this function is to get current blue and green stats as variables and swap it. Once swap completed rewrite the characteristics of those special states.

Method 1

The requirement of the sub section is to determine suitable policy for this dynamic environment via policy iteration as we explained earlier policy iteration methodology, we discussed in part 1 question 2 sub section 2. The specially adjusted algorithm for the evaluation is

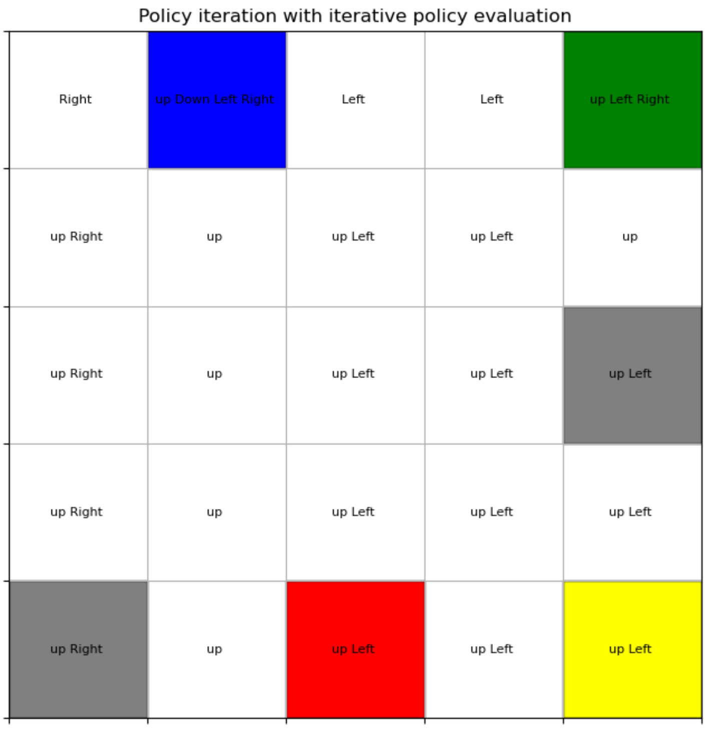
Adjusted Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi^*$

1. Initialization
   $V(s) \in R \; and \; \pi(s) \in A(s) arbirararily \; for \; all \; s \in S$
2. Policy Evaluation
   Loop:
   - $\Delta \leftarrow 0$
   - Loop for each $s \in S$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} P(s', r | s, \pi(s))[r + \gamma V(s')]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   Permute special states with 0.1 probability
   Until $\Delta < \theta$
3. Policy Improvement
   policy stable $\leftarrow$ true
   For each $s \in S$:
   - Old action $\leftarrow \pi(s)$
   - $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} P(s', r | s, a)[r + \gamma V(s')]$
   - If old action $\neq \pi(s)$, then policy stable $\leftarrow$ false

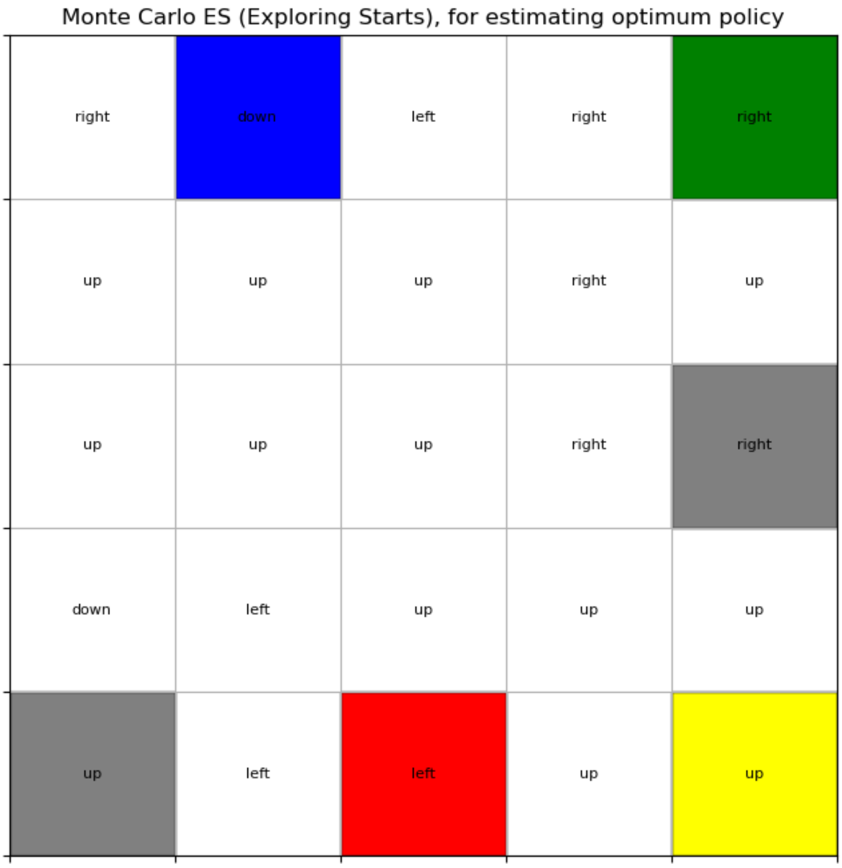
   If policy stable then stop and return $V \approx v^*$ and $\pi \approx \pi^*$ else go to 2

Here the θ value plays a significant role the lower the theta value the higher the time it will get to settle for an optimal policy. From the obtained results in figure below it is visible that policy evaluation will not give you a good out come when there is a dynamic environment is present. It will try to move to initial highest valued blues state most of the time.



Method 2

Her we have used Monte Carlo method with exploring start algorithm used in Part 2 Question 1 sub section 1 with adjustment to episode generation function where in each step of the episode permutation function is called. By running the algorithm for 10,000 episodes gives you some accurate representation of the optimum policy. According to results it was observed that agent tried to move towards either blue and green states whenever they are close by. And also when the agent is closer to terminal points instead of going to blue and green states agent move to terminal state where reward is 0. Since taking further steps incur -0.2 reward it is ideal fo get 0 rather than -0.2.
From the both methodologies suggested it can be observed that methodology 2 is more suitable for this kind of dynamic environments. Policy obtained by method 2 is illustrated in below figure.



You can recreate the above results from referring to the file A2P2Q3.ipynp and it is available in the below git hub repository

Link: https://github.com/CharukaP/RL_Gridworld