Reinforcement Learning -Homework-2

In this homework set, we are to compute the algorithms for a Gambling game, as explained in the class.

The premise of the game :

We start if \$5 and based on three policies; we change the way we play the game. The game ends when we either have \$0 or \$10. In this homework set, we play the game in three ways. First using an *aggressive policy*, in which you always bet the maximum amount. Second, in a *conservative policy*, in which you always bet 1 dollar no matter how much money we have. Lastly, in a *random policy*, in which we randomly pick an amount to bet with uniform distribution. Lastly the probability of getting heads is *0.9* and the probability of getting tails is *0.1* , and we win when we get head and lose all money we bet if we get tails

Formula used :

To implement the value functions for the policy, we make use of the following method:
V(s)=

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

-------[1]

To allow for the algorithm to converge and not form a infinite recursive function, we taken specific steps to include constants like theta, more on which will be seen in the algorithm implementation.

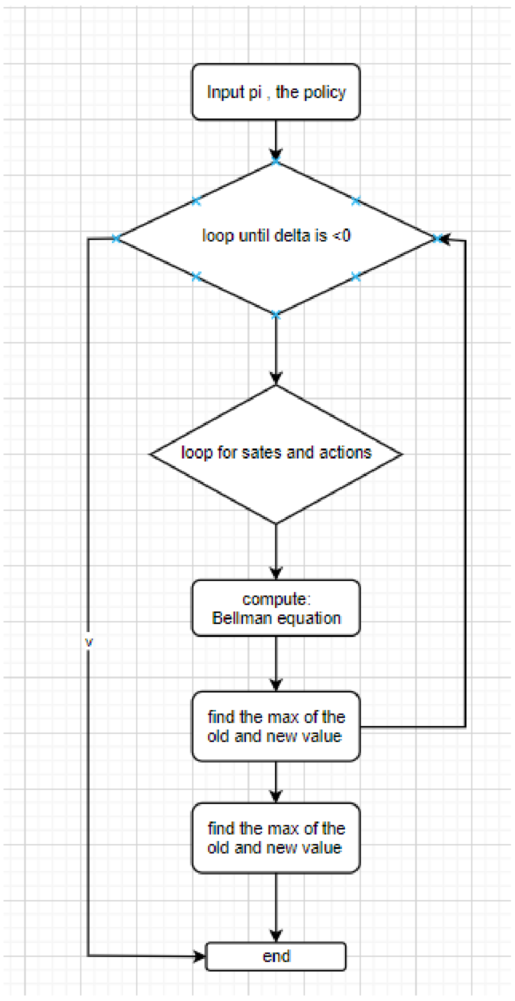
Algorithm implementation:

First, we define the matrix to implement the various policies. This is a matrix of probabilities, which has rows of actions and columns of states, and for example, in aggressive policy, it gives us the meaning of the aggressive policy, where depending on the state we are in, we bet that much. Say, for example, we are in the state 5, we would bet in the game \$5. This follows a similar suit for the other policies. Next, we define the function calls for the various policies and this helps us code in several polices for the same game using a similar reward pattern.

Next, we create a function that allows us to implement the various policies. Next, we have to initialize the array V, and this is randomly initialized, and I am initially initializing this to all 0's. Since the game ends when the values reach 0 and 10, we initialize the V values for these as zero. We introduce an algorithm parameter to know as theta, delta. Theta helps us determine the accuracy of the estimation. While delta helps in converging the code. Since we need to run the whole code until we get a value that is closer to the real value or the estimate. This can only take place over many iterations, which is why we run the code in an infinite loop until the delta value becomes less than the theta. So, we now create a loop to go from all the values of the states, inside which we define another loop through the number of actions present. In this particular game, the number of actions remains inside the number 10 since the game ends when we reach $10. To calculate the delta function, we require to update the value of the array_bufer. This value acts as a buffer to find a delta, which might be the maximum between delta and the absolute difference between array_bufer and value for a state. Next, we calculate a delta value and assign Value as value for the game. As discussed, the policy matrix can tell us with what probability we can bet in each state. We can use all this to define the action probability which is the pi of the bellman's equation. We then use the action probability and then we use this to update the value function equation as given above. [1].The various conditional statements using the else and else if help us check the terminal conditions like when Value(0). Using these conditional statements, we are able to update the value in algorithm using the bellman equation as given above . We make use of an *expected update* operation in the *Bellman equation* . Each iterative policy evaluations updates the value of every state to produce the new approximate vale function

Upon updating the value each step using the equation, which helps us to find the value of delta, which in turn helps us find the terminating condition for the infinite loop. In the end, we return the data to the value function.

In short, the algorithm implemented flows the flowing flowchart


Input pi , the policy
loop until delta is <0
loop for sates and actions
compute:
Bellman equation
find the max of the old and new value
find the max of the old and new value
end
v

Various polices implemented :

In the *aggressive policy* we bet the amount we currently have that is the state that we are currently in . The probability of getting a head to win is 0.9 whereas the probability of getting a tail in 0.1 . So, if a player wins, he reaches a higher state and when he loses goes into a lower state

In the *conservative policy* we no matter the state or that is the amount we hold always bet $1 . Also, the probability remains the same for this case

Lastly for the *Random policy* we bet a random amount that we posses , that is a random amount from our current state . Similarly, the probability remains the same

Value functions :

- Aggressive policy

```
9.4976    9.6640    6.7200    8.9600    4.0000    4.8000    5.6000    6.4000    7.2000
```

- Conservative policy

```
7.8889    7.8765    6.9863    5.9985    4.9998    4.0000    3.0000    2.0000    1.0000
```

- Random policy

```
8.6531    8.7257    8.1515    7.6107    6.6706    5.6261    5.0530    4.7502    4.6138
```