HW 11 and HW 12 due 06/08/2023

Problem 1(a) 30% (b) 30% (c) 20 % total 80%

Problem 2 (a) 20% (b)20% total 40%

Problem 3 (a) 30%(b) 30% (c) 20% total 80%

Problem 1. Monte Carlo is one scheme in reinforcement learning. In the appendix, you can find the description of the python code. Here I give you an example code in python. You can easily handcraft the position of the obstacle the way you want and see what happens. The problem is two dimensional grid problem. You can imagine a robot moving in three possible directions, i.e., up, righ, and left. Each direction has associated probabilities but the total sum of these values are 1.

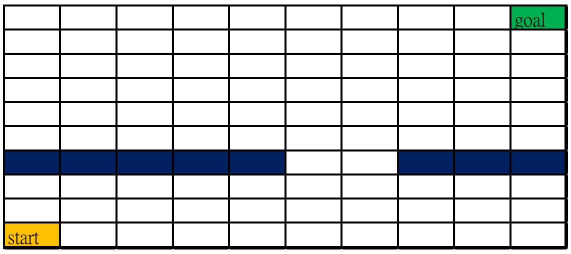
```
PI= [0.6,0.2,0.2] #probability of moving up, moving right, and moving down
```

We want to limit maximal trials to be 100. So do not change the number 100 in this line

```
    for i in range(100): # the maximal number of trial need to be fixed to 100
            p=random.uniform(0,1) #generate another random number
```

In the end, the shortest trajectory will be stored and in the end the result will be printed out.

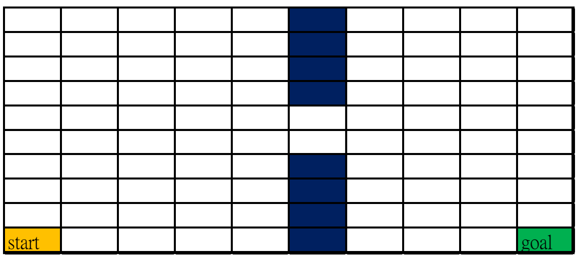
(a) Modify the program to change the obstacle such that the new grid world is given by



Do the simulation for 1000 iteration and print out the result. Plot the movement trajectory of the robot in the grid world to verify it indeed reached the goal.

(b) Change the goal state to another corner

Change the goal state to



Modify the program so that the robot is allowed to move only in three directions up, right, and *down* with corresponding probabilities 0.6, 0.2 and 0.2.

Do the simulation for 1000 iteration.

```
For k in range(1000):
```

Note that reaching the goal becomes a rare event so you may need to run several times to get one working trajectory that robot indeed reach the goal state. You need to unmask this line to check if the goal is reached.

```
Print('iteration',k, 'move', count_move, 'goal reached')
```

Otherwise, you do not know if the goal is reached or not.

Plot the movement trajectory of the robot in the grid world to verify it indeed reached the goal.

(b) Now change the number of iteration to 10000 and change the sampling code to sample every 1000 iteration

```
if k % 1000==0:    #sample result of improvement
        # change this sampling to 1000 when using 10000 iteration
        print('iter', k, 'result', count_move_prev)
```

How many goal reaching trajectories did you catch? Show the results of successful "goal reaching" trajectory with number of moves. For example,

iteration XX move 39 goal reached

iteration XX move 53 goal reached

Markov Chain with transitional probabiity

Problem 2 You may need to run this in google colab if you do not have the library in your computer.

If we change the transition probability to be

```
transition_probabilities = [ # shape=[s, s']
        [0.7, 0.2, 0.0, 0.1],    # from s0 to s0, s1, s2, s3
        [0.0, 1.0, 0.0, 0.0],    # from s1 to …
        [0.0, 0.9, 0.0, 0.1],    # from s2 to …
        [0.8, 0.2, 0.0, 0.0]]    # from s3 to …
```

(a) which state out of {s0, s1, s2,s3} will the terminal state?

(b) Run the markov chain simulation to print out the result for 10 iteration.

(you may want to unmask the line and input the expected terminal state

3. Markov Decision Problem

In the example code, the following markov decision problem is given

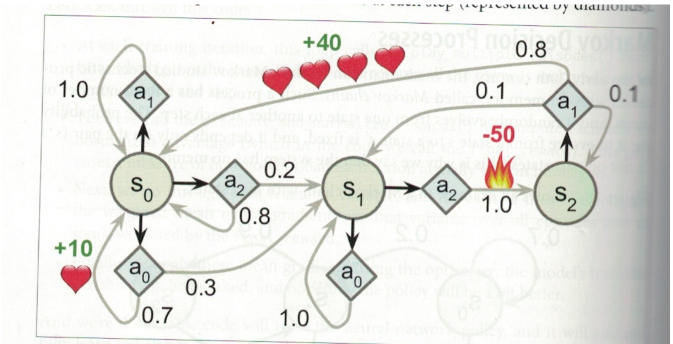


Figure 18-8. Example of a Markov decision process

\# markov decision process

transition_probabilities = [# shape=[s, a, s']

[[0.7, 0.3, 0.0], [1.0, 0.0, 0.0], [0.8, 0.2, 0.0]],

[[0.0, 1.0, 0.0], None, [0.0, 0.0, 1.0]],

[None, [0.8, 0.1, 0.1], None]]

Please read the figure and identify that [0.7, 0.3, 0.0] corresponds the transitional probability to s0, s1, and s2 when a0 is taken and current state is s0. The 0.7 probabilty correspond to reward +10 (a red heart symbol).

In the second row of the array, None means a1 is forbidden when in state s1.

In the third row of the array, two None means a0 and a2 are both forbidden.

Rewards = [# shape=[s, a, s']

[[+10, 0, 0], [0, 0, 0], [0, 0, 0]],

[[0, 0, 0], [0, 0, 0], [0, 0, +50]],

[[0, 0, 0], [+40, 0, 0], [0, 0, 0]]]

The default discount rate gamma is 0.9. (Later you will change this number in (c))

(a) Now we want to change the -50 reward to +50 reward with the state action pair (S1, a2). (You can see the fire symbol in the figure.)

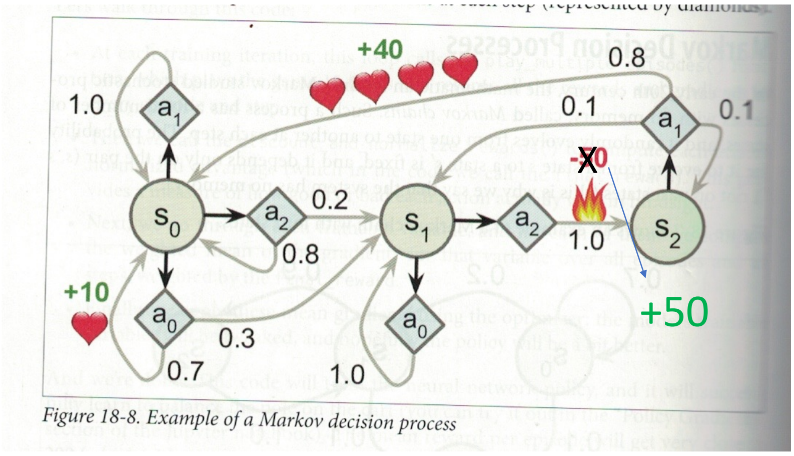


Figure 18-8. Example of a Markov decision process

What is the expected optimal policy? First, guess and then calculate Q(s,a) and use Q(s,a) to find out the optimal policy. Express the optimal policy in terms of the following table

State	action need to be taken
s0	
s1	
s2	

(b)

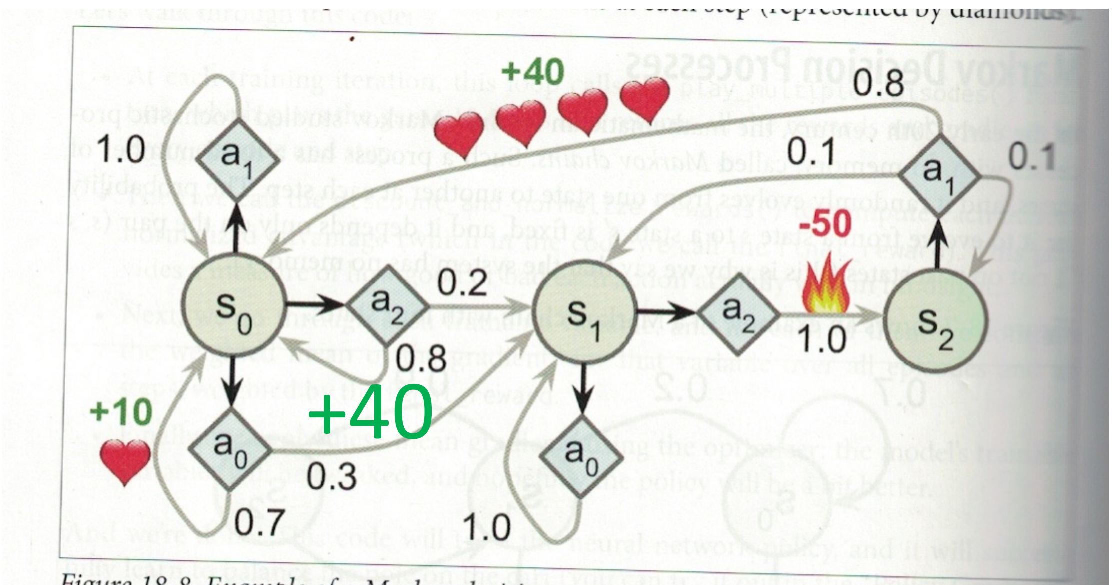


Figure 18-8. Example of a Markov decision process

Now change the +50 reward back to -50. Now we would like to add reward +40 when state is in S0 and a2 is taken. The corresponding probability is 0.8. Calculate Q(s,a) and find the optimal policy.

State	action need to be taken
s0	
s1	
s2	

(b) Now we switch back to the old problem but changing the discount rate gamma to 0.8 to calculate Q(s,a). And change gamma to 0.95 and recalculate Q(s,a). Show Q for each case. List the optimal policy for each case(gamma=0.8 and 0.95). In other words, fill the table

State	action (gamma =0.8)	Action(gamma =0.95)
s0		
s1		
s2		

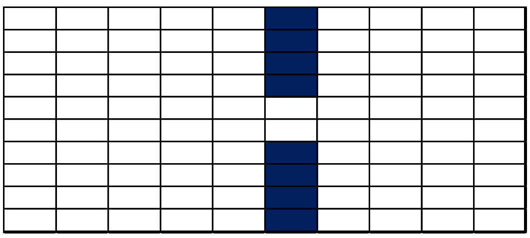
What has been change in optimal policy? What is the plausible reason?

Appendix: Additional explanation for the code in problem 1. Here I give you an example code in python. You can easily handcraft the position of the obstacle the way you want and see what happens. The problem is two dimensional grid problem. You can imagine a robot moving in four possible directions, i.e., up, down, left and right. Each direction has associated probabilities but the total sum of these values are 1. In the simulation, a random number of uniform distribution between zero and one, i.e, U(0,1) is used to determine the outcome.

In the example code, the default terminal state is up and right corner and the robot is allowed to move only in three directions up, right, and left with corresponding probabilities 0.6, 0.2 and 0.2.

```
PI= [0.6,0.2,0.2] #probability of moving up, moving right, and moving left
```

The size of the grid is 10 x 10 with obstacle to partition the grid word as follows.



To implement the idea of a wall and obstacle.

We define a function called insidewindow to return TRUE if the state x and state y is within the window. Similarly, we define a function obstacle to check if the robot hits the obstacle.

```
if p>=0 and p<=PI[0]:
            state_y=state_y+1    # MOVE UP
            if insidewindow(state_x,state_y) and obstacle(state_x,state_y):
                current_trajectory[count_move]= 1
                count_move=count_move+1
```

```
        else:
            state_y=state_y-1
```

There is also a line within the loop to check if the goal is reached. If the goal is reached, the loop simply “breaks”.

```
if state_x==width-1 and state_y==length-1:
        # print('iteration',k, 'move', count_move, 'goal reached')
        break
```

An array is used to keep track of the movement. The maximal length is ~100, enough for our application. In the end, the trajectory can be printed out.