HW 11 and HW 12

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%

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.

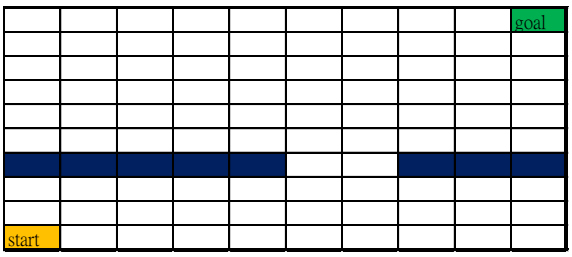
PI= [0.6,0.2,0.2] #probability of moving up, moving right, and moving down

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



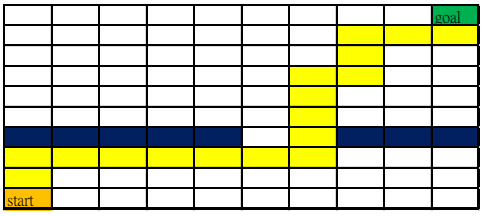
Do the simulation for 1000 iteration and print out the result. Plot the movement trajectory of the robot in the grid world to verify it indeed reached the goal.

Answer: modifying the obstacle function

Obtacles are (0,3) (1,3) (2,3) (3,3) (4,3) (7,3) (8,3) (9,3)

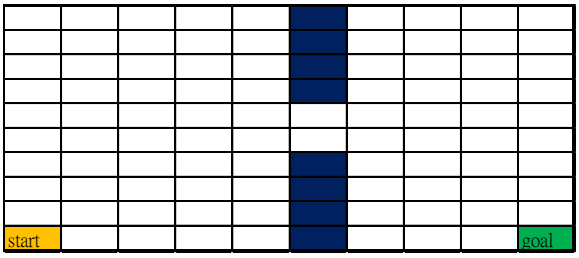
```
def obstacle(x,y):
    if x==0 and y==3:
        return False # hitting the obstacle
    elif x==1 and y==3:
        return False
    elif x==2 and y==3:
        return False
    elif x==3 and y==3:
        return False
    elif x==4 and y==3:
        return False
    elif x==7 and y==3:
        return False
    elif x==8 and y==3:
        return False
    elif x==9 and y==3:
        return False
    else:
        return True
```

```
Python 3.10.7 (v3.10.7:6cc6b13308, Sep  5 2022, 14:02:52) [Clang 13.0.0 (clang-1
300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/jack/Dropbox/課程規劃/machine learning 2023/Homework/HW 11 & 12
/code example/MC 053123 Prob 1a.py
iter 0 result 38
iter 100 result 22
iter 200 result 18
iter 300 result 18
iter 400 result 18
iter 500 result 18
iter 600 result 18
iter 700 result 18
iter 800 result 18
iter 900 result 18
index 0 movement 1
index 1 movement 1
index 2 movement 2
index 3 movement 2
index 4 movement 2
index 5 movement 2
index 6 movement 2
index 7 movement 2
index 8 movement 1
index 9 movement 1
index 10 movement 1
index 11 movement 1
index 12 movement 2
index 13 movement 1
index 14 movement 1
index 15 movement 2
index 16 movement 2
index 17 movement 1
```



(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.

Answer change the goal sate to (9,0)

And probability to 0.6, 0.2,0.2

You need to modify several part of the code:

```
else:
                state_y=state_y-1 #move down
                if insidewindow(state_x,state_y)and obstacle(state_x,state_y):
                        current_trajectory[count_move]= 3
                        count_move=count_move+1
                else:
                        state_y=state_y+1
#goal checking
        if state_x==9 and state_y==0:
        #unmask the following line to check if goal is reached
                print('iteration',k, 'move', count_move, 'goal reached')
                break


#goal checking
        if state_x==9 and state_y==0:
        #unmask the following line to check if goal is reached
                print('iteration',k, 'move', count_move, 'goal reached')
                break
```

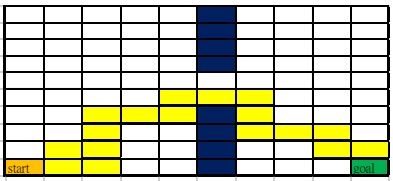
Problem 1(b) and (c)

Answer:

```
cout move 72
iter 0 result 72
cout move 43
cout move 41
cout move 33
cout move 32
iter 100 result 32
iter 200 result 32
iteration 208 move 21 goal reached
cout move 21
iter 300 result 21
iter 400 result 21
iter 500 result 21
iter 600 result 21
iter 700 result 21
iter 800 result 21
iter 900 result 21
```

only one iteration the robot reached goal. The rest of them failed.

index	0	movement	2	right	
index	1	movement	1	up	
index	2	movement	3	down	
index	3	movement	2	right	
index	4	movement	1	up	
index	5	movement	1	up	
index	6	movement	1	up	
index	7	movement	2	right	
index	8	movement	2	right	
index	9	movement	1	up	
index	10	movement	2	right	
index	11	movement	2	right	
index	12	movement	3	down	
index	13	movement	3	down	
index	14	movement	1	up	
index	15	movement	3	down	
index	16	movement	2	right	
index	17	movement	2	right	
index	18	movement	3	down	
index	19	movement	2	right	
index	20	movement	3	down	



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

(c) 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

Answer: typically you will get ~5 trajectories

```
cout move 58
iter 0 result 58
cout move 56
cout move 45
cout move 43
cout move 41
cout move 40
cout move 27
iter 1000 result 27
iter 2000 result 27
iteration 2586 move 39 goal reached
iteration 2959 move 53 goal reached
iter 3000 result 27
cout move 25
iter 4000 result 25
iter 5000 result 25
iteration 5131 move 39 goal reached
iteration 5247 move 75 goal reached
iter 6000 result 25
iteration 6751 move 33 goal reached
iteration 6981 move 41 goal reached
iter 7000 result 25
iteration 7644 move 25 goal reached
iter 8000 result 25
iter 9000 result 25
```

```
index 0 movement 1
index 1 movement 1
index 2 movement 1
index 3 movement 1
index 4 movement 1
index 5 movement 1
index 6 movement 2
index 7 movement 1
index 8 movement 2
index 9 movement 3
index 10 movement 1
index 11 movement 2
index 12 movement 1
index 13 movement 1
index 14 movement 2
index 15 movement 3
index 16 movement 1
index 17 movement 3
index 18 movement 1
index 19 movement 3
index 20 movement 1
index 21 movement 3
index 22 movement 1
index 23 movement 3
index 24 movement 1
```

Transitional probability

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 …

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

Answer : s1

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

both ans 1 and ans2 are correct anwer!

ans1: the result looks like this if you do not unmake the code for terminal state

```
States: 0 0 3 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
States: 0 3 0 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
States: 0 0 0 0 0 3 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
States: 0 3 0 0 0 3 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
States: 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
States: 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
States: 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
States: 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
States: 0 0 3 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
States: 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ...
```

ans2: because we know the terminal state is s1, we can unmask the code to break the loop when the markov chain reaches state s1.

```
States: 0 0 3 0 0 0 0 0 1
States: 0 0 1
States: 0 0 3 1
States: 0 0 0 0 0 0 0 0 0 0 0 0 0 1
States: 0 0 0 0 0 0 0 0 3 1
States: 0 1
States: 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 3 0 3 1
States: 0 0 3 0 0 0 0 0 0 1
States: 0 0 0 0 0 1
States: 0 0 3 0 0 0 1
```

3. Markov Decision Problem

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

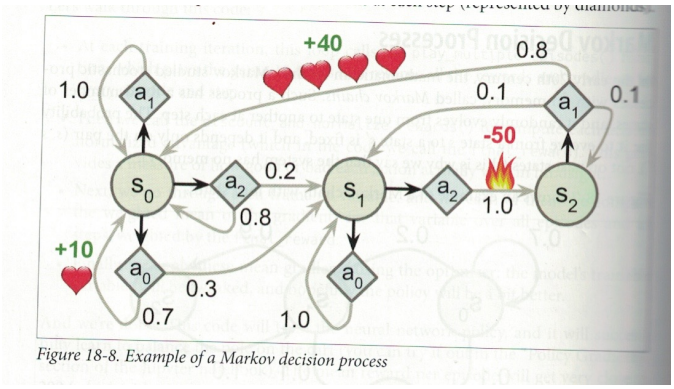


Figure 18-8. Example of a Markov decision process

```
# 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]]]
```

(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.)

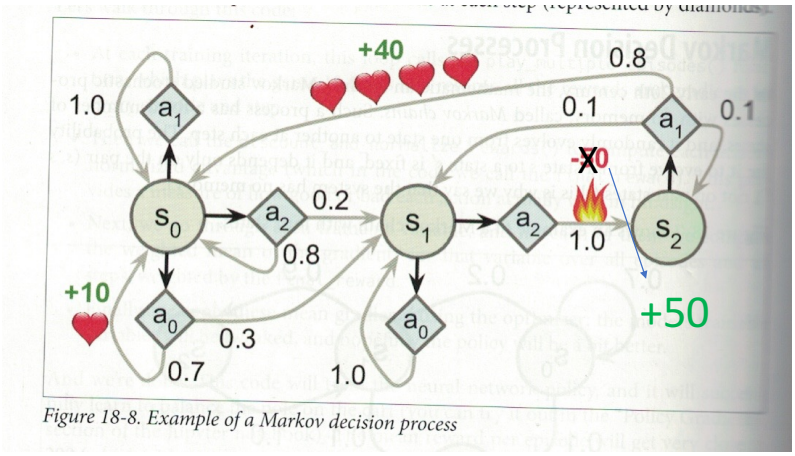


Figure 18-8. Example of a Markov decision process

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	

Answer:

You need to modify rewards as follow

```
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]]]
```

#change reward to +50 from -50

```
array([[195.20173993, 175.57359999, 183.99235995], [217.66739979,
-inf, 241.9726286 ], [ -inf, 213.42288283, -inf]])
```

State	action need to be taken
s0	a0
s1	a2
s2	a1

(b)

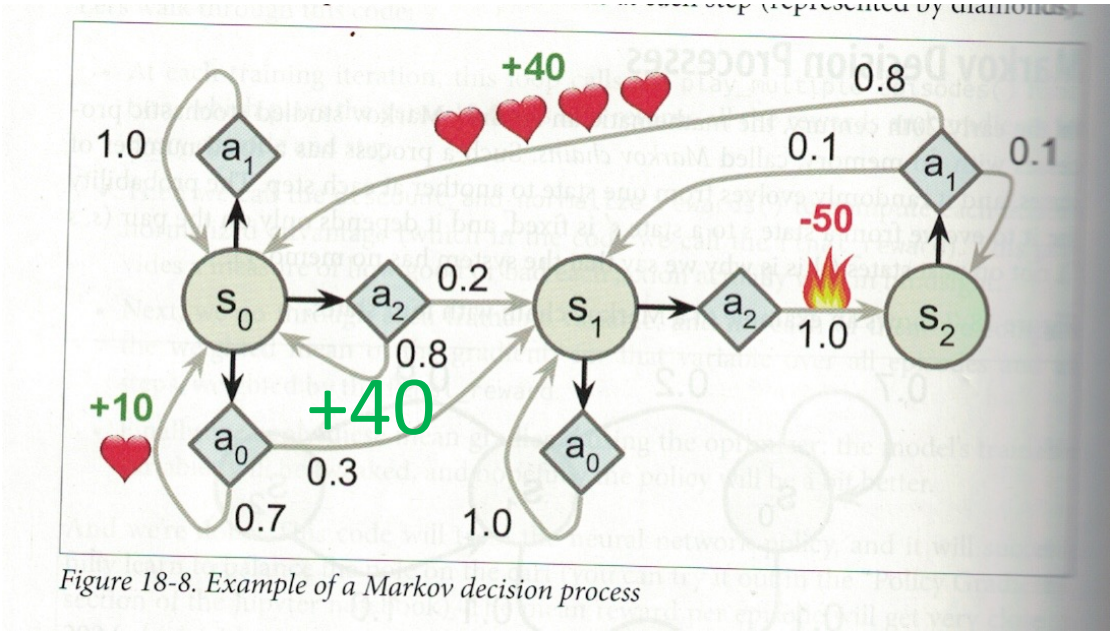


Figure 18-8. Example of a Markov decision process

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.

Answer:

```
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]]
rewards = [ # shape=[s, a, s']
                [[+10, 0, 0], [0, 0, 0], [40, 0, 0]],
                [[0, 0, 0], [0, 0, 0], [0, 0, -50]],
                [[0, 0, 0], [+40, 0, 0], [0, 0, 0]]]
possible_actions = [[0, 1, 2], [0, 2], [1]]
```

```
array([[171.93310092, 182.37975835, 202.7486534 ], [124.22423359,
-inf, 138.13140366], [ -inf, 209.13937041, -inf]])
```

State	action need to be taken
s0	a2

s1	a2
s2	a1

(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). List the optimal policy for each case. 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?

Answer: change the reward and transitional proablity back

```
# 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]]
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]]]
possible_actions = [[0, 1, 2], [0, 2], [1]]
```

you only need to modify gamma =0.8

```
gamma = 0.8    # the discount factor
```

```
array([[ 15.90909091, 12.72727273, 10.18181818], [ 0. , -inf, -
13.3201581 ], [ -inf, 45.84980237, -inf]])
```

run for gamma =0.95, you will obtain

array([[21.73304188, 20.63807938, 16.70138772], [0.95462106, -inf, 1.01361207], [-inf, 53.70728682, -inf]])

optimal policy: I marked yellow the change in policy.

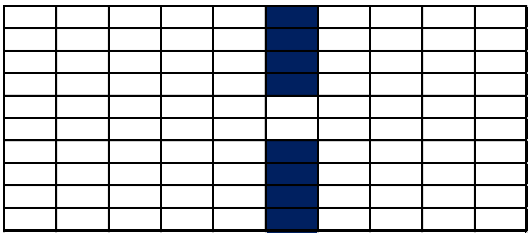
When the chain is in s1, the optimal action changes from a0 to a1. We can see that when discount rate gamma change to 0.95, future reward is valued more so that the robot is willing to go through fire (-50 reward) to obtain future reward.

State	action (gamma =0.8)	action(gamma =0.95)
s0	a0	a0
s1	a0	a1
s2	a1	a1

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.