COMP3702

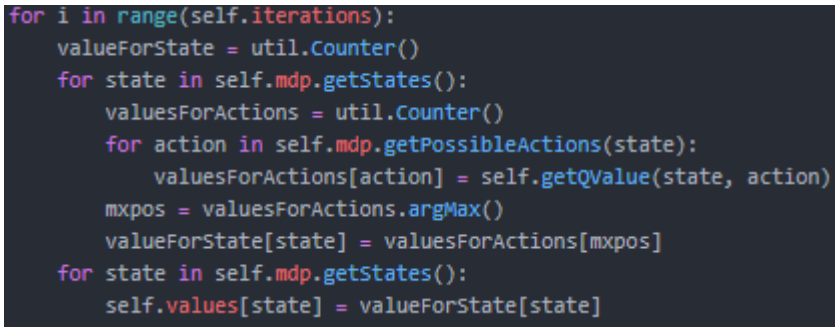
Assignment 3

By:

Fan Yang 44594213

Q1:

For this question, first the iteration value in init function needs to be initialized.



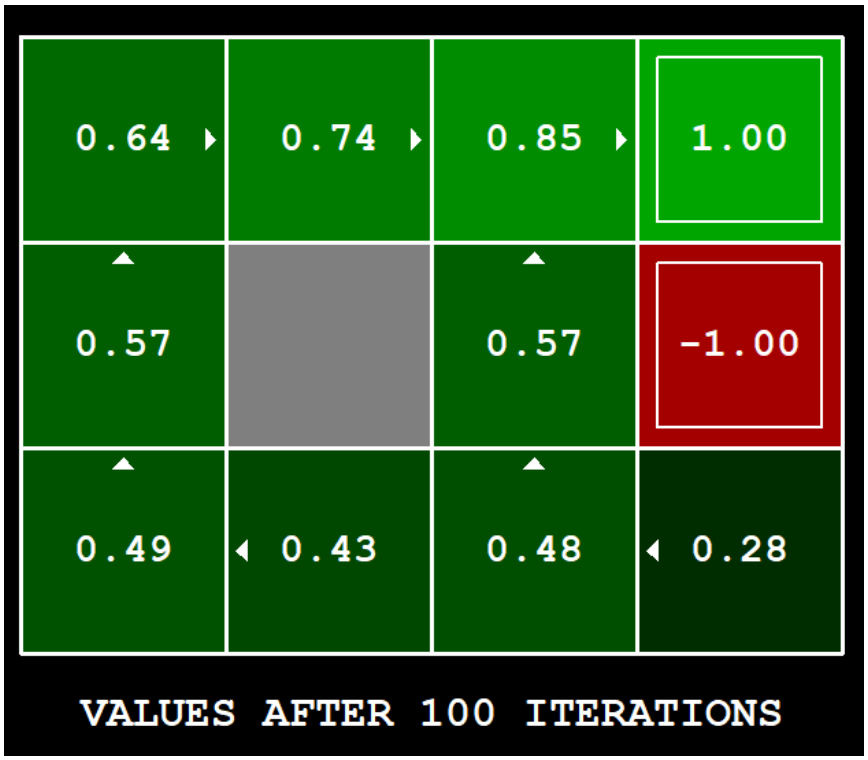
```
for i in range(self.iterations):
    valueForState = util.Counter()
    for state in self.mdp.getStates():
        valuesForActions = util.Counter()
        for action in self.mdp.getPossibleActions(state):
            valuesForActions[action] = self.getQValue(state, action)
        mxpos = valuesForActions.argMax()
        valueForState[state] = valuesForActions[mxpos]
    for state in self.mdp.getStates():
        self.values[state] = valueForState[state]
```

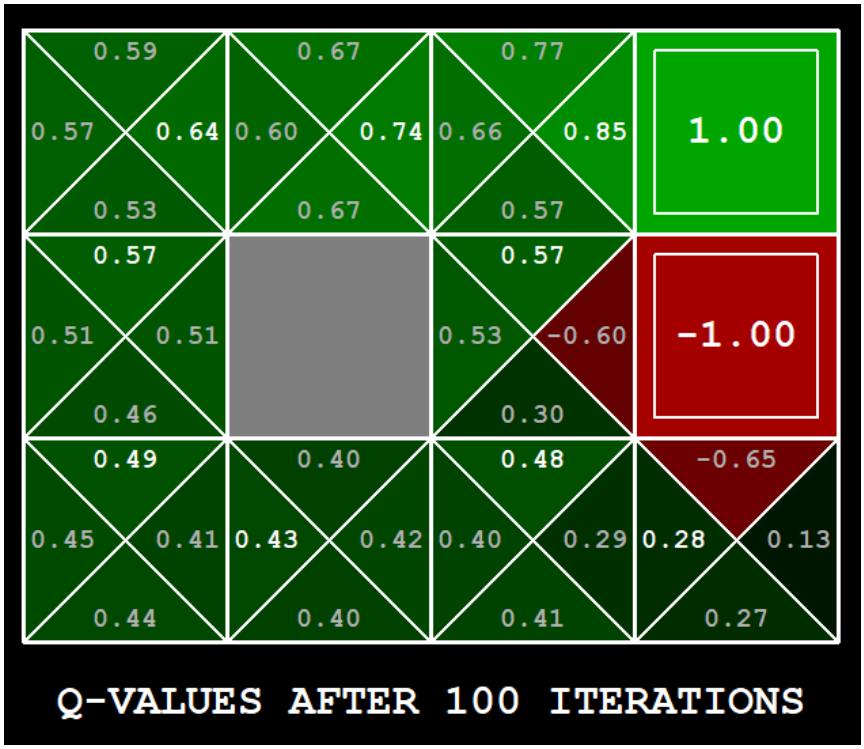
Then as shown in figure above, after GetPossibleActions is performed to get the accurate action, Q-value of an action is evaluated. The required 3 functions are shown as below.

1) getValue returns the value of a state and the implementation is simply:
   getValue(self, state):
    return self.values[state]
2) getPolicy returns the best action according to computed values and is implemented as below. We can see that it's the max value in return value:
    if len(self.mdp.getPossibleActions(state)) != 0:
        valuesForActions = util.Counter()
        for action in self.mdp.getPossibleActions(state):
            valuesForActions[action] = self.getQValue(state, action)
        return valuesForActions.argMax()
    else:
        return None
3) getQValue returns the Q-value of the (state,action) pair and the implementation is shown as below:
    for transition in self.mdp.getTransitionStatesAndProbs(state, action):
        qValue = qValue + transition[1] * (self.mdp.getReward(state, action,
                transition[0]) + self.discount * self.values[transition[0]])
        return qValue

   The Q-Value is rewarded and (discount*values) is the value in this action.

The results can seen from the figures below:





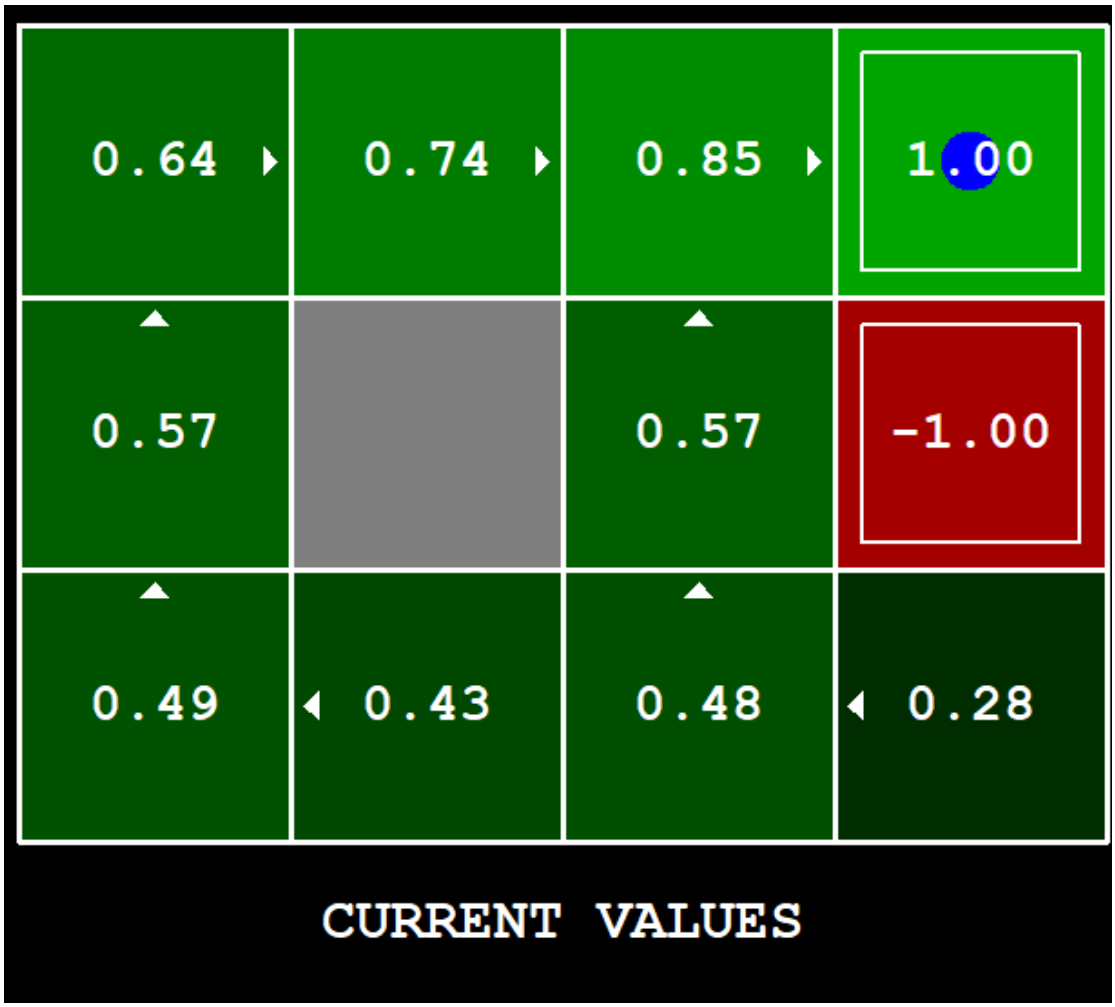
The running screenshot:

```
Started in state: (2, 2)
Took action: east
Ended in state: (3, 2)
Got reward: 0.0

Started in state: (3, 2)
Took action: exit
Ended in state: TERMINAL_STATE
Got reward: 1

EPISODE 10 COMPLETE: RETURN WAS 0.47829690000000014


AVERAGE RETURNS FROM START STATE: 0.51127517601
```



Addition question:

1) states

All the possible locations is in state space. In this case, there are 11 and we can represent as: S = {(1,1), (1,2) ... . (4,3)}

2) Action:

Every legal move counts in the action which can be represented as :
A={$Left, right, up, down$}

3) Reward function:

Reward function R(s,a) can be represented as the table below:

	Right	Left	Up	down
(1,1)				
……				
(4,3)				

And the blank is the return value of each point.

4) Transition function:

The transition function T(s,a,s') can be represented as the table below:

	(1,1)	……	(4,3)
(1,1)	P	P	P
…….			
(4,3)			

Where the blank contains the probabilities for the next states s', after taking action from state s.

5) Bellman equation:

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} \mathbf{v}^k$$

Q2&3:

We need to compute a rate of the problem. And when the problem is in cliff problem we can get a question of -10.0 the path we can compute as the answer. The right value needs to be found and we can use a test function to beigin in 0 -1 and every time 0.1 is added and check whether it satisfy the requirement. There's a 10*10 = 100 and the answer is shown as below.

```
1.  def question2():
2.    answerDiscount = 0.9
3.    answerNoise = 0.0
4.    return answerDiscount, answerNoise
```

```
5.
6.  def question3a():
7.    answerDiscount = 0.2
8.    answerNoise = 0.0
9.    answerLivingReward = 0.0
10.   return answerDiscount, answerNoise, answerLivingReward
11.   # If not possible, return 'NOT POSSIBLE'
12.
13. def question3b():
14.   answerDiscount = 0.2
15.   answerNoise = 0.2
16.   answerLivingReward = 0.0
17.   return answerDiscount, answerNoise, answerLivingReward
18.   # If not possible, return 'NOT POSSIBLE'
19.
20. def question3c():
21.   answerDiscount = 0.8
22.   answerNoise = 0.0
23.   answerLivingReward = 0.0
24.   return answerDiscount, answerNoise, answerLivingReward
25.   # If not possible, return 'NOT POSSIBLE'
26.
27. def question3d():
28.   answerDiscount = 0.8
29.   answerNoise = 0.2
30.   answerLivingReward = 0.0
31.   return answerDiscount, answerNoise, answerLivingReward
32.   # If not possible, return 'NOT POSSIBLE'
33.
34. def question3e():
35.   answerDiscount = 0.0
36.   answerNoise = 0.0
37.   answerLivingReward = 1.0
38.   return answerDiscount, answerNoise, answerLivingReward
```

```
# If not possible, return 'NOT POSSIBLE'
```

Q4:

First, as for our q-value update, we define update function:

```
answer = (1-self.alpha) * self.getQValue(state,action) + self.alpha * (reward + self.gamma * self.getValue(nextState))
self.qValues[(state,action)] = answer
```

And getAction function:

```
legalActions = self.getLegalActions(state)
action = None
randomAction = random.choice(legalActions)
bestAction = self.getPolicy(state)
if random.random() < self.epsilon:
    return randomAction
else:
    return bestAction
```

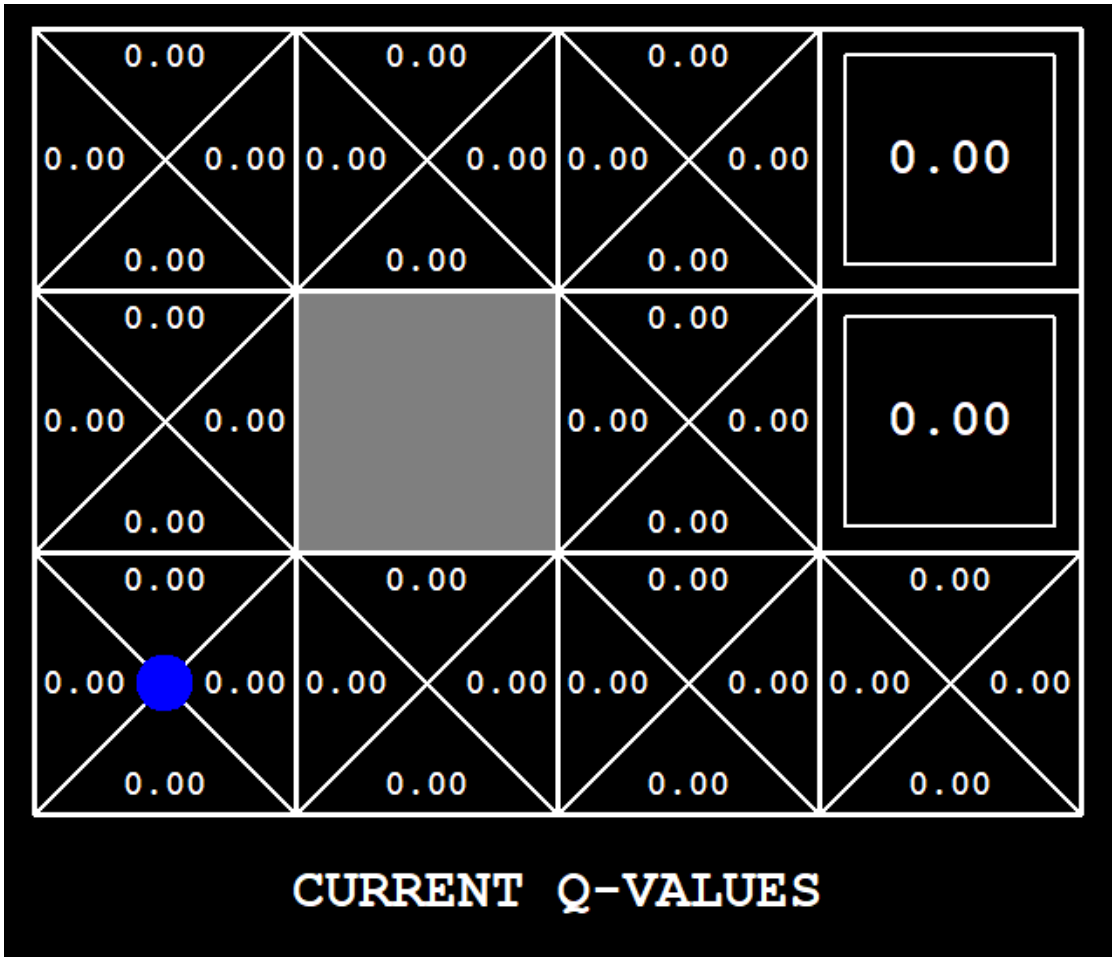
Random is used to find if the action needs to be chosen from random function and several functions are used to evaluate the value of each function as shown in above figure.

```
valuesForActions = util.Counter()
for action in self.getLegalActions(state):
    valuesForActions[action] = self.getQValue(state,action)
return valuesForActions.argMax()
```

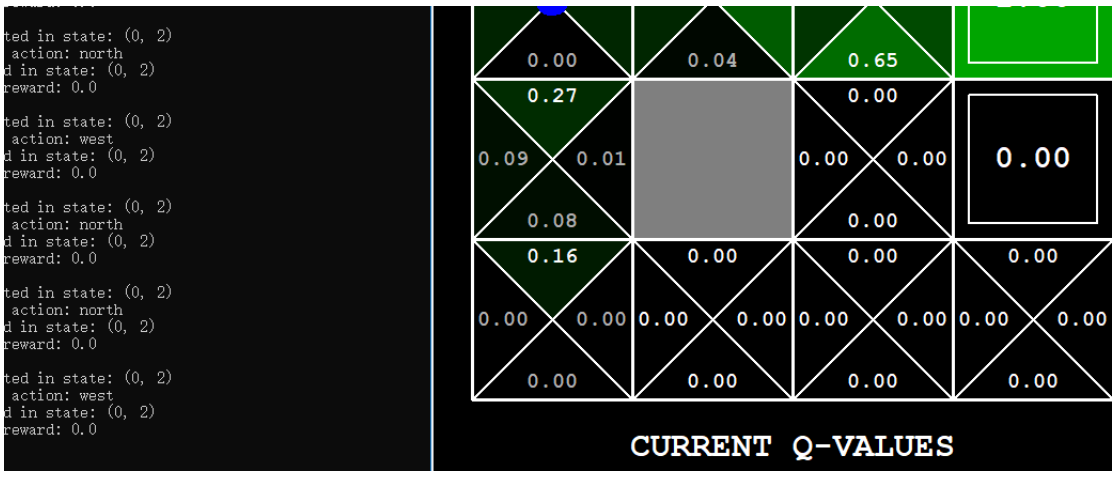
And then this function to return the value of max_action.

Output is shown as below:

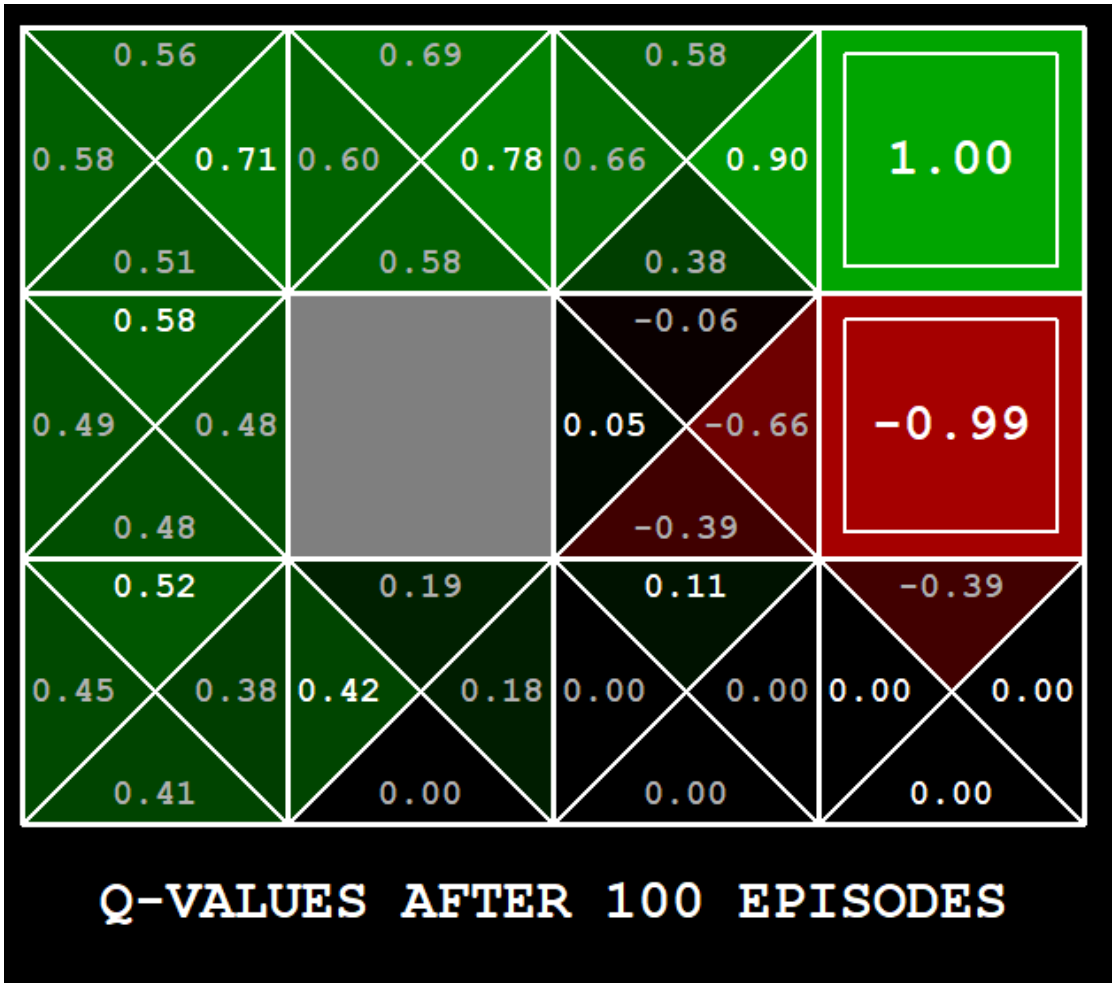
python gridworld.py -a q -k 5 -m



python gridworld.py -a q -k 100
Running screenshot



And the final



In each term Q learning choose a way to find the action. If get rewarded then the path node q value will increase. In opposite, the path node get minus will decrease and in this way we can tell whether the value is decreased. This is a model-free approach and it's also active.

Q5:

Epsilon greedy function:
First we define an epsilon. When random value is greater than epsilon, the value of this variable is known. Then we can know that:
1) [0 - epsilon] is exploration
2) [epsilon -1] is exploitation

As for alternative method, we can use the method which only exploration and Epsilon =1. Or using other distribution in random.

Q6:

The way to compute these values are quite simple. Several samples are run and then observe [0-1] [0.3-0.5]. And it’s found that at the head and end, it can’t come up with that values. So the answer is:

```
def question6():
    answerEpsilon = None
    answerLearningRate = None
    return "NOT POSSIBLE"
```

Q7:

Use the Q learning agent we already have done:

```
py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid
Beginning 2000 episodes of Training
Reinforcement Learning Status:
        Completed 100 out of 2000 training episodes
        Average Rewards over all training: -510.69
        Average Rewards for last 100 episodes: -510.69
        Episode took 0.65 seconds
Reinforcement Learning Status:
        Completed 200 out of 2000 training episodes
        Average Rewards over all training: -511.66
        Average Rewards for last 100 episodes: -512.63
        Episode took 0.88 seconds
Reinforcement Learning Status:
        Completed 300 out of 2000 training episodes
        Average Rewards over all training: -502.16
        Average Rewards for last 100 episodes: -483.16
        Episode took 1.00 seconds
```

Running screenshot

And the final is:

```
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Average Score: 500.6
Scores:        495, 503, 503, 503, 495, 503, 503, 503, 503, 495
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

Q8:

We want to train a pacman in this part and here are several functions implemented for this to work. We write to Init weight value:

```
self.featExtractor = util.lookup(extractor, globals())()
PacmanQAgent.__init__(self, **args)
self.weights = util.Counter()
```

And it's how our update function works:

```
difference = reward + self.gamma * self.getValue(nextState)
                    - self.getQValue(state, action)

features = self.featExtractor.getFeatures(state, action)
for key in features.keys():
    self.weights[key] = self.weights[key] + self.alpha * difference *
                        features[key]
```

After running it:

```
(base) C:\Users\Administrator\Desktop\800\assignment-support-code-master\assignment-3-support-code-master>python pacman.
py -p ApproximateQAgent -x 2000 -n 2010 -l smallGrid
Beginning 2000 episodes of Training
Reinforcement Learning Status:
        Completed 100 out of 2000 training episodes
        Average Rewards over all training: -510.40
        Average Rewards for last 100 episodes: -510.40
        Episode took 1.13 seconds
Reinforcement Learning Status:
        Completed 200 out of 2000 training episodes
        Average Rewards over all training: -511.65
        Average Rewards for last 100 episodes: -512.91
        Episode took 1.60 seconds
```

It'll show that:


```
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 503
Pacman emerges victorious! Score: 495
Pacman emerges victorious! Score: 499
Average Score: 501.0
Scores:        503, 503, 503, 495, 503, 503, 503, 503, 495, 499
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

We can see it's better than the other method and it can be abstracted as:

$$w_i \leftarrow w_i + \alpha[correction] f_i(s,a)$$
$$correction = \big(R(s,a) + \gamma V(s')\big) - Q(s,a)$$

And now the status is where Pacman is and if there's fruit and if it's the answer.