CAP 5636 – Fall 2020 - Homework 2

Step 1: (10 pts)
Solve problems Q1 to Q4 from the Berkeley AI reinforcement learning project:

Q1. <u>Value Iteration</u>
Solution: For this value iteration, we need to implement QValue formula which is as below:

Start with V0(s) = 0: no time steps left means an expected reward sum of zero.
Given vector of Vk(s) values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

Now as to implement this through coding we need to edit below code in **valueIterationAgents.py:**

Users > pritijagtap > Downloads > reinforcement 2 > valueIterationAgents.py > ...

```
        """
        self.mdp = mdp
        self.discount = discount
        self.iterations = iterations
        self.values = util.Counter() # A Counter is a dict with default 0

        # Write value iteration code here
        "*** YOUR CODE HERE ***"


    def getValue(self, state):
        """
          Return the value of the state (computed in __init__).
        """
        return self.values[state]


    def computeQValueFromValues(self, state, action):
        """
          Compute the Q-value of action in state from the
          value function stored in self.values.
        """
        "*** YOUR CODE HERE ***"
        util.raiseNotDefined()

    def computeActionFromValues(self, state):
        """
          The policy is the best action in the given state
          according to the values currently stored in self.values.

          You may break ties any way you see fit.  Note that if
          there are no legal actions, which is the case at the
          terminal state, you should return None.
        """
        "*** YOUR CODE HERE ***"
        util.raiseNotDefined()
```

Original code.jpg

After creating the learning code:

```python
        """
        self.mdp = mdp
        self.discount = discount
        self.iterations = iterations
        self.values = util.Counter() # A Counter is a dict with default 0

        # Write value iteration code here
        "*** YOUR CODE HERE ***"
        for _ in range(self.iterations):
            tempvalues = util.Counter()

            for state in self.mdp.getStates():
                if self.mdp.isTerminal(state):
                    continue
                actions = self.mdp.getPossibleActions(state)
                value = [self.getQValue(state, action) for action in actions]
                tempvalues[state] = max(value) if value else 0
            self.values = tempvalues
```

```python
    def computeQValueFromValues(self, state, action):
        """
          Compute the Q-value of action in state from the
          value function stored in self.values.
        """
        "*** YOUR CODE HERE ***"
        Qvalue= 0
        for next_state_value, transition_prob in self.mdp.getTransitionStatesAndProbs(state, action):
            reward = self.mdp.getReward(state, action, next_state_value)
            future_reward_value = self.discount * self.values[next_state_value]
            Qvalue += transition_prob * (reward + future_reward_value)

        return Qvalue
        # util.raiseNotDefined()
```

```python
def computeActionFromValues(self, state):
    """
      The policy is the best action in the given state
      according to the values currently stored in self.values.

      You may break ties any way you see fit.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return None.
    """
    "*** YOUR CODE HERE ***"
    if self.mdp.isTerminal(state):
        return None
    policy = util.Counter()
    for action in self.mdp.getPossibleActions(state):
        policy[action] = self.computeQValueFromValues(state, action)
    return policy.argMax()
    # util.raiseNotDefined()
```

Below is command which we use to run the program:

python gridworld.py -a value -i 5

After writing code below is the final output:

Output on grid.jpg

Below is Output on terminal:

```
pritijagtap@Pritis-MacBook-Air reinforcement % python gridworld.py -a value -i 5
DEPRECATION WARNING: The system version of Tk is deprecated and may be removed in a future release. Please don't rely on it. Set TK_SILENCE_DEPRECATION=1 to suppress th
is warning.

RUNNING 1 EPISODES

BEGINNING EPISODE: 1

Started in state: (0, 0)
Took action: north
Ended in state: (0, 1)
Got reward: 0.0

Started in state: (0, 1)
Took action: north
Ended in state: (0, 2)
Got reward: 0.0

Started in state: (0, 2)
Took action: east
Ended in state: (1, 2)
Got reward: 0.0

Started in state: (1, 2)
Took action: east
Ended in state: (2, 2)
Got reward: 0.0

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 1 COMPLETE: RETURN WAS 0.59049


AVERAGE RETURNS FROM START STATE: 0.59049
```

If we run the autograder command get the below output:

```
pritijagtap@Pritis-MacBook-Air reinforcement % python autograder.py -q q1
Starting on 11-13 at 23:33:08

Question q1
===========

*** PASS: test_cases/q1/1-tinygrid.test
*** PASS: test_cases/q1/2-tinygrid-noisy.test
*** PASS: test_cases/q1/3-bridge.test
*** PASS: test_cases/q1/4-discountgrid.test

### Question q1: 6/6 ###


Finished at 23:33:08

Provisional grades
==================
Question q1: 6/6
------------------
Total: 6/6

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

For this Question we got below values against each iteration and the graph is plotted in Step 2.

Iteration	Values
1	0
2	0
3	0
4	0.37
5	0.5
6	0.58
7	0.61
8	0.63
9	0.64
10	0.64

Q2. <u>Bridge Crossing Analysis</u>

For this question we have to change analysis.py. Change the values for answerDiscount = 0.9 and answerNoise = 0.2.

Original code:

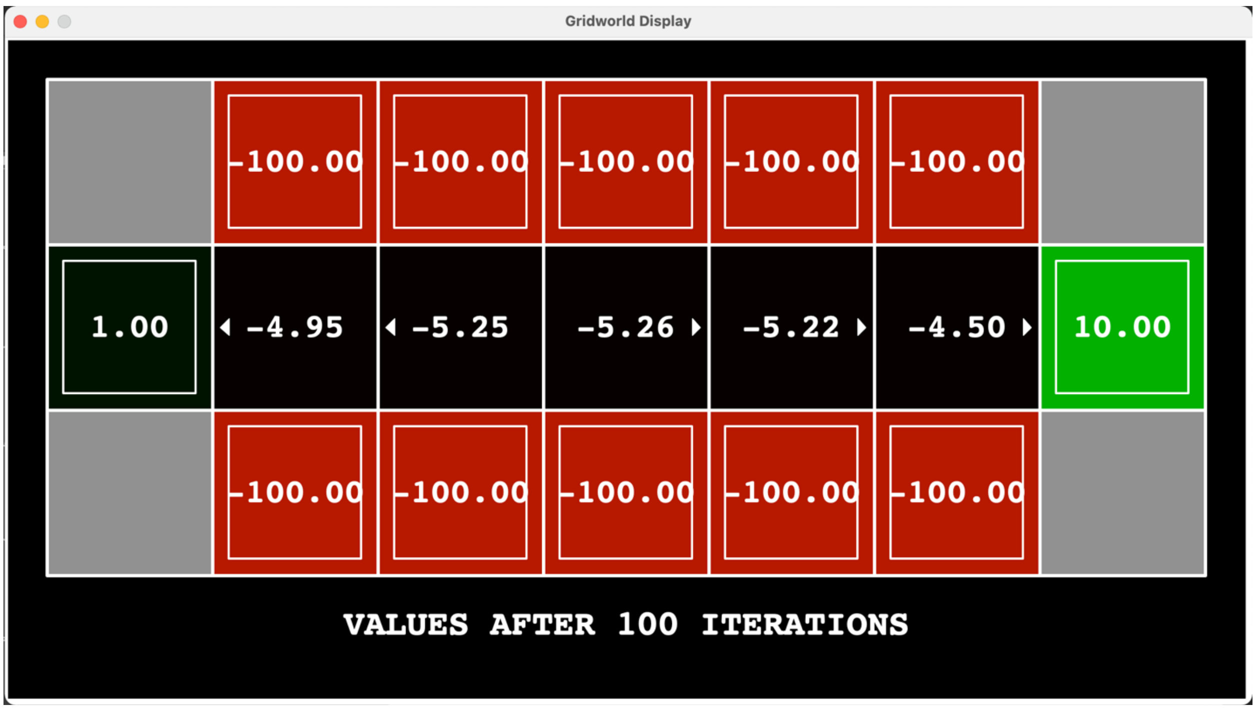
```
######################
# ANALYSIS QUESTIONS #
######################

# Set the given parameters to obtain the specified policies through
# value iteration.

def question2():
    answerDiscount = 0.9
    answerNoise = 0.2
    return answerDiscount, answerNoise
```

We tried lot of combinations for answerDiscount and answerNoise values. Like answerDiscount = 0.1 and answerNoise = 0.5. For mentioned values below is output and we used this command: python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.1 --noise 0.5

```python
def question2():
    answerDiscount = 0.1
    answerNoise = 0.5
    return answerDiscount, answerNoise
```



After creating the learning code:

```python
######################
# ANALYSIS QUESTIONS #
######################

# Set the given parameters to obtain the specified policies through
# value iteration.

def question2():
    answerDiscount = 0.9
    answerNoise = 0.0
    return answerDiscount, answerNoise
```

We are using below command to get the output:

python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0.0

This is the final output for the program:

Q3: Policies

In this question we need to use the optimal policy types for below. We are editing **analysis.py**, in the file we going to edit **answerDiscount, answerNoise and answerLivingReward** paramters in the file.

a. Prefer the close exit (+1), risking the cliff (-10)
   For this we are editing def question3a:

   Original code:

```
def question3a():
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

After creating the learning code:

```
def question3a():
    answerDiscount = 0.2
    answerNoise = 0.2
    answerLivingReward = -7
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

b. Prefer the close exit (+1), but avoiding the cliff (-10)
For this we are editing def question3b:

Original code:

```
def question3b():
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

After creating the learning code:

```
def question3b():
    answerDiscount = 0.2
    answerNoise = 0.2
    answerLivingReward = -0.4
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

c. Prefer the distant exit (+10), risking the cliff (-10)
For this we are editing def question3c:

Original code:

```python
def question3c():
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

After creating the learning code:

```python
def question3c():
    answerDiscount = 0.9
    answerNoise = 0.0
    answerLivingReward = 0.0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

d. Prefer the distant exit (+10), avoiding the cliff (-10)
For this we are editing def question3d:

Original code:

```python
def question3d():
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

After creating the learning code:

```python
def question3d():
    answerDiscount = 0.8
    answerNoise = 0.3
    answerLivingReward = 0.3
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

e. Avoid both exits and the cliff (so an episode should never terminate)
For this we are editing def question3e:

Original code:

```
def question3e():
    answerDiscount = None
    answerNoise = None
    answerLivingReward = None
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

After creating the learning code:

```
def question3e():
    answerDiscount = 0.9
    answerNoise = 0.0
    answerLivingReward = 1.0
    return answerDiscount, answerNoise, answerLivingReward
    # If not possible, return 'NOT POSSIBLE'
```

Once done editing the analysis.py we are going to run code using below command:

python autograder.py -q q3

Below is the output for the program:

```
pritijagtap@Pritis-MacBook-Air reinforcement % python autograder.py -q q3
Starting on 11-15 at 22:00:53

Question q3
===========

*** PASS: test_cases/q3/1-question-3.1.test
*** PASS: test_cases/q3/2-question-3.2.test
*** PASS: test_cases/q3/3-question-3.3.test
*** PASS: test_cases/q3/4-question-3.4.test
*** PASS: test_cases/q3/5-question-3.5.test

### Question q3: 5/5 ###


Finished at 22:00:54

Provisional grades
==================
Question q3: 5/5
------------------
Total: 5/5

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

Q4: <u>Q-Learning</u>

For this question we have edit 4 functions as mentioned in the instructions from file qlearningAgents.py and functions are update, computeValueFromQValues, getQValue, and computeActionFromQValues:

Original code:

```python
def __init__(self, **args):
    "You can initialize Q-values here..."
    ReinforcementAgent.__init__(self, **args)

    "*** YOUR CODE HERE ***"
```

```python
def getQValue(self, state, action):
    """
      Returns Q(state,action)
      Should return 0.0 if we have never seen a state
      or the Q node value otherwise
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()
```

```python
def computeValueFromQValues(self, state):
    """
      Returns max_action Q(state,action)
      where the max is over legal actions.  Note that if
      there are no legal actions, which is the case at the
      terminal state, you should return a value of 0.0.
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()
```

```python
def computeActionFromQValues(self, state):
    """
      Compute the best action to take in a state.  Note that if there
      are no legal actions, which is the case at the terminal state,
      you should return None.
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()
```

```python
def update(self, state, action, nextState, reward):
    """
      The parent class calls this to observe a
      state = action => nextState and reward transition.
      You should do your Q-Value update here

      NOTE: You should never call this function,
      it will be called on your behalf
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()
```

After creating the learning code:

```python
def __init__(self, **args):
    "You can initialize Q-values here..."
    ReinforcementAgent.__init__(self, **args)

    "*** YOUR CODE HERE ***"
    self.qvalue = util.Counter()
```

```python
def getQValue(self, state, action):
    """
      Returns Q(state,action)
      Should return 0.0 if we have never seen a state
      or the Q node value otherwise
    """
    "*** YOUR CODE HERE ***"
    return self.qvalue[(state, action)]
    # util.raiseNotDefined()
```

```python
def computeValueFromQValues(self, state):
    """
    Returns max_action Q(state,action)
    where the max is over legal actions.  Note that if
    there are no legal actions, which is the case at the
    terminal state, you should return a value of 0.0.
    """
    "*** YOUR CODE HERE ***"
    if len(self.getLegalActions(state)) == 0:
        return 0.0
    maximum_qvalue = float('-inf')
    for actions in self.getLegalActions(state):
        maximum_qvalue = max(maximum_qvalue, self.getQValue(state, actions))
    return maximum_qvalue
    # util.raiseNotDefined()
```

```python
def computeActionFromQValues(self, state):
    """
    Compute the best action to take in a state.  Note that if there
    are no legal actions, which is the case at the terminal state,
    you should return None.
    """
    "*** YOUR CODE HERE ***"
    if len(self.getLegalActions(state)) == 0:
        return None
    bestaction_in_state = []
    best_qvalue = self.computeValueFromQValues(state)
    for actions in self.getLegalActions(state):
        if best_qvalue == self.getQValue(state, actions):
            bestaction_in_state.append(actions)

    return random.choice(bestaction_in_state)
    # util.raiseNotDefined()
```

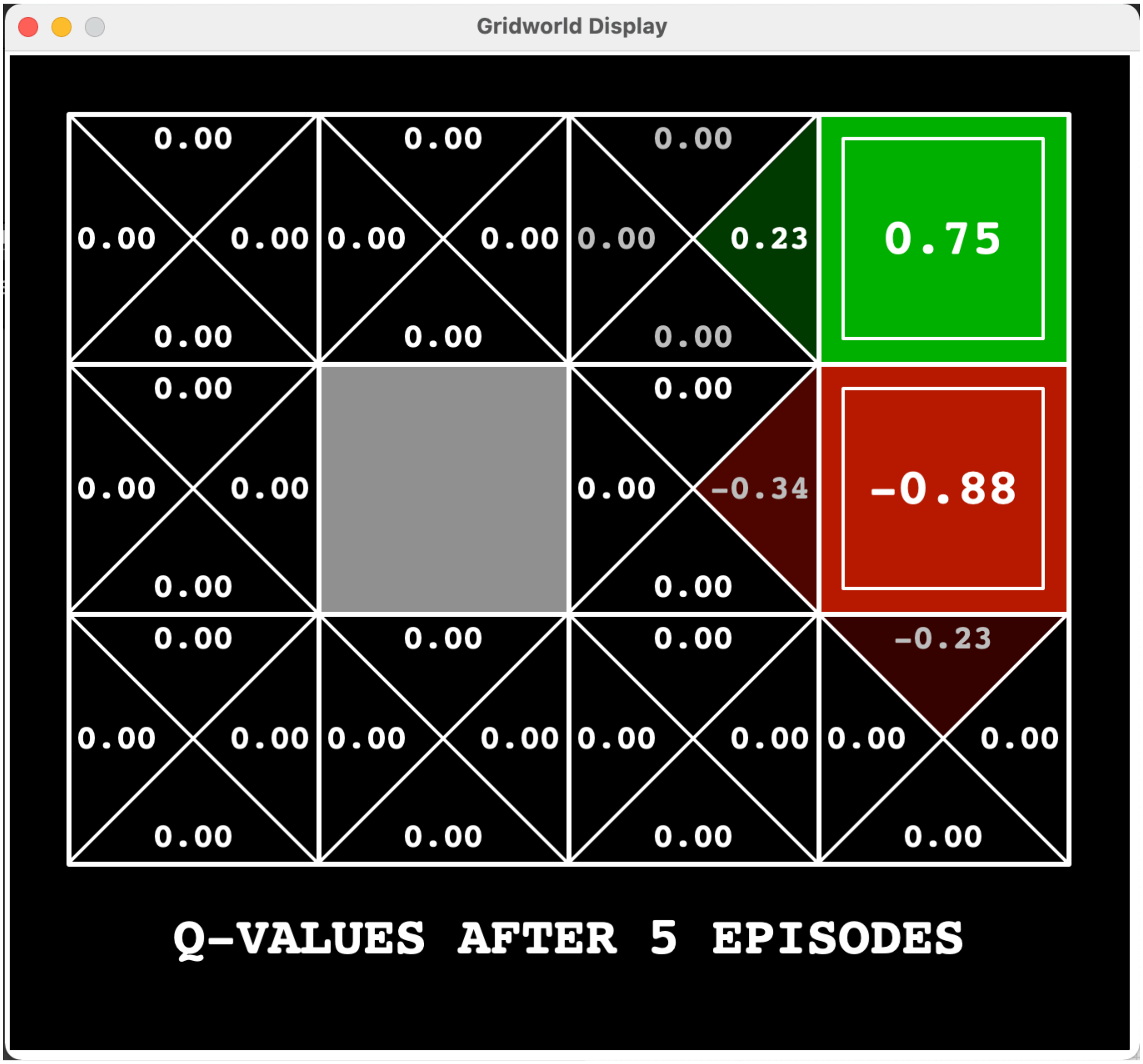
```python
def update(self, state, action, nextState, reward):
    """
      The parent class calls this to observe a
      state = action => nextState and reward transition.
      You should do your Q-Value update here

      NOTE: You should never call this function,
      it will be called on your behalf
    """
    "*** YOUR CODE HERE ***"
    update_q_value = self.qvalue[(state, action)]
    values = reward + (self.discount * self.computeValueFromQValues(nextState))
    self.qvalue[(state, action)] = (1 - self.alpha) * update_q_value + self.alpha * values
    # util.raiseNotDefined()
```

Final output on gridworld:

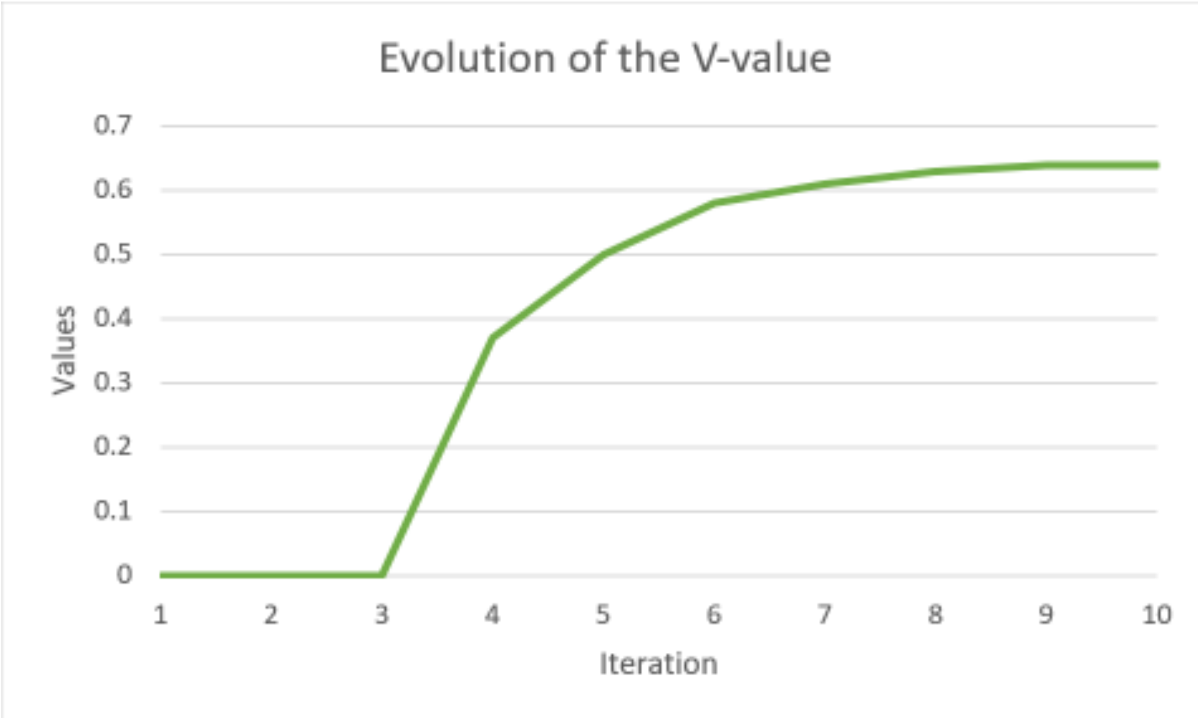
After running program for 100 iterations, we got below output:

We got different values through the 5 iterations for each episode and those values are stored in one csv file.


Gridworld Display
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.23
0.00
0.75
0.00
0.00
0.00
0.00
0.00
0.00
-0.34
0.00
-0.88
0.00
0.00
0.00
0.00
-0.23
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
0.00
Q-VALUES AFTER 5 EPISODES

Iteration	North	East	West	South
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0.056429
10	0	0	0	0.056429
11	0	0	0	0.056429
12	0	0	0	0.056429
13	0	0	0	0.056429
14	0	0	0	0.056429
15	0	0	0	0.056429
16	0	0	0	0.056429
17	0	0	0	0.056429
18	0	0	0	0.056429
19	0	0	0	0.056429
20	0	0	0	0.056429
21	0	0	0	0.056429
22	0	0	0	0.056429
23	0	0	0	0.056429
24	0	0	0	0.056429

Step 2: Visualize the evolution of the V-value (4 pts)



Step 3: Visualize the evolution of the Q-value (4 pts)