

CSEN 266 Artificial Intelligence

Group Homework 3

Name: Ashutosh Uday Talwalkar ID: 00001650053 (50%)

Name: Yash Nandkumar Patil ID: 07700006459 (50%)

getQValue(state, action):

- This function returns the Q-value associated with a state-action pair. If the state-action pair has not been encountered before, it returns 0.0.
- Implementation: The Q-values are stored in a dictionary `self.qvalues` where keys are state-action pairs and values are the corresponding Q-values. If the state-action pair exists in the dictionary, its associated Q-value is returned. Otherwise, it returns 0.0.

python

```
def getQValue(self, state, action):  
    if (state, action) in self.qvalues:  
        return self.qvalues[(state, action)]  
    else:  
        return 0.0
```

computeValueFromQValues(state):

- This function computes the maximum Q-value over all legal actions in a given state, representing the value of that state.
- Implementation: It iterates over all legal actions in the state, retrieves their corresponding Q-values using `getQValue` function, and returns the maximum value.

python

```
def computeValueFromQValues(self, state):  
    qvalues = [self.getQValue(state, action) for action in  
self.getLegalActions(state)]  
    if not len(qvalues):  
        return 0.0  
    return max(qvalues)
```

computeActionFromQValues(state):

- This function computes the best action to take in a given state based on the maximum Q-value.
- Implementation: It first retrieves the maximum Q-value using `computeValueFromQValues`, then finds all actions with Q-values equal to the maximum value, and finally randomly selects one of those actions.

python

```
def computeActionFromQValues(self, state):
    best_value = self.getValue(state)
    best_actions = [action for action in self.getLegalActions(state)
if self.getQValue(state, action) == best_value]
    if not len(best_actions):
        return None
    else:
        return random.choice(best_actions)
```

update(state, action, nextState, reward):

- This function updates the Q-value associated with a state-action pair based on the observed transition and received reward.
- Implementation: It computes the new Q-value using the Q-learning update rule and updates the `self.qvalues` dictionary with the new value.

python

```
def update(self, state, action, nextState, reward):
    disc = self.discount
    alpha = self.alpha
    qvalue = self.getQValue(state, action)
    next_value = self.getValue(nextState)
    new_value = (1-alpha) * qvalue + alpha * (reward + disc *
next_value)
    self.setQValue(state, action, new_value)
```

getAction(state):

- This function computes the action to take in the current state using an epsilon-greedy strategy.
- Implementation: It randomly selects a legal action with probability `self.epsilon` or selects the best action based on Q-values with probability `1 - self.epsilon`.

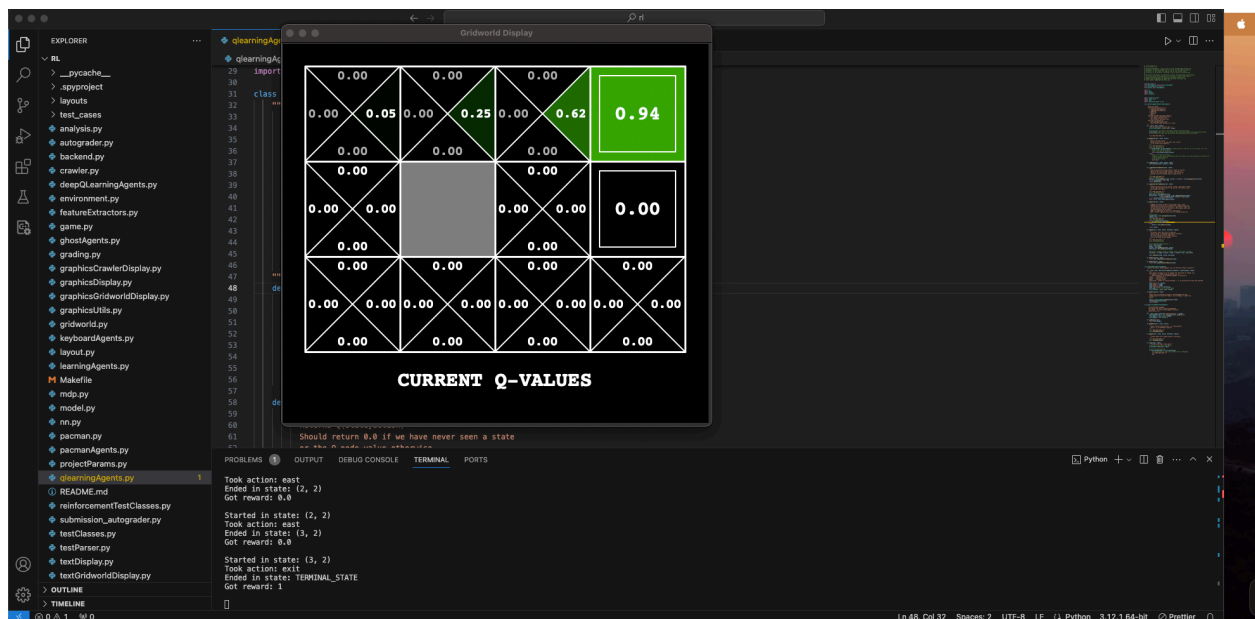
python

```
def getAction(self, state):  
    legalActions = self.getLegalActions(state)  
    if util.flipCoin(self.epsilon):  
        return random.choice(legalActions)  
    else:  
        return self.computeActionFromQValues(state)
```

These implementations demonstrate how the Q-learning algorithm updates Q-values and selects actions based on those values while exploring the environment and exploiting learned knowledge.

Task A

Experiment 1



Experiment 2

```
1 # qlearningAgents.py
2 #
3 # Licensing Information: You are free to use or extend these projects for
4 # educational purposes provided that (1) you do not distribute or publish
5 # solutions, (2) you retain this notice, and (3) you provide clear
6 # attribution to UC Berkeley, including a link to http://ai.berkeley.edu.
7 #
8 # Attribution Information: The Pacman AI projects were developed at UC Berkeley.
9 # The core projects and autograders were primarily created by John DeNero
10 # (denereq@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
11 # Student side autograding was added by Brad Miller, Nick May, and
12 # Pieter Abbeel (pabbeel@cs.berkeley.edu).
13
14
15 from game import *
16 from learningAgents import ReinforcementAgent
17 from featureExtractors import *
18 from backend import ReplayMemory
19
20 import nn
21 import model
22 import backend
23 import gridworld
24
25
26 import random, util, math
27 import numpy as np
28 import copy
29 import matplotlib.pyplot as plt
30
31 class QLearningAgent(ReinforcementAgent):
32
33     def __init__(self, gamma=0.9, epsilon=0.05):
34         super().__init__(gamma, epsilon)
35
36     def getQValue(self, state, action):
37         return self.qvalues[state][action]
38
39     def chooseAction(self, state):
40         if self.epsilon > random.random():
41             return random.choice(self.getLegalActions(state))
42         else:
43             return self.argmax(self.getQValues(state))
44
45     def update(self, state, action, nextState, reward):
46         oldQ = self.getQValue(state, action)
47         newQ = reward + gamma * self.getQValue(nextState, self.chooseAction(nextState))
48         self.qvalues[state][action] = (1 - self.alpha) * oldQ + self.alpha * newQ
49
50     def save(self, filename):
51         util.dump(self.qvalues, filename)
52
53     def load(self, filename):
54         self.qvalues = util.load(filename)
55
56     def __str__(self):
57         return 'QLearningAgent'
58
59     def __repr__(self):
60         return 'QLearningAgent'
61
62     def __hash__(self):
63         return hash('QLearningAgent')
```

Question q3

```
*** PASS: test_cases/q3/1-1inygrid.test
*** PASS: test_cases/q3/2-1inygrid-noisy.test
*** PASS: test_cases/q3/3-bridge.test
*** PASS: test_cases/q3/4-discountgrid.test

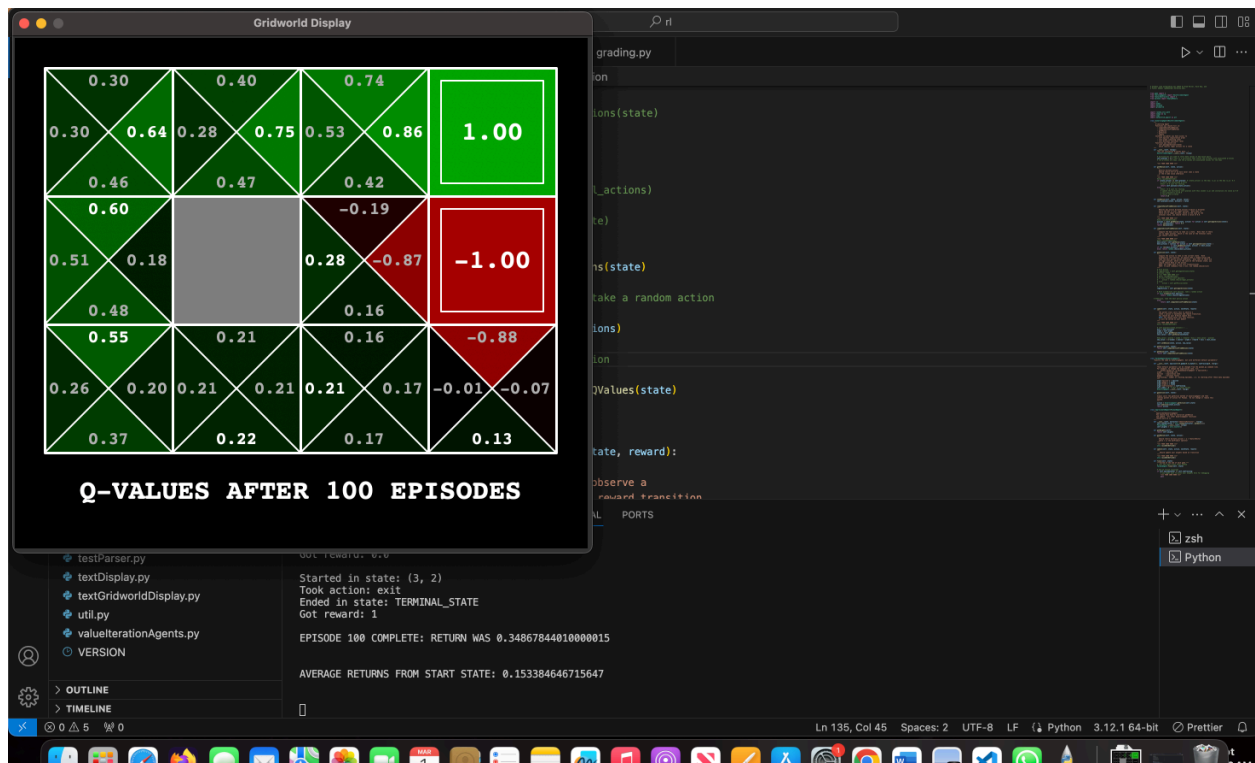
*** Question q3: 5/5 ***

Finished at 20:58:04
Provisional grades
Question q3: 5/5
Total: 5/5

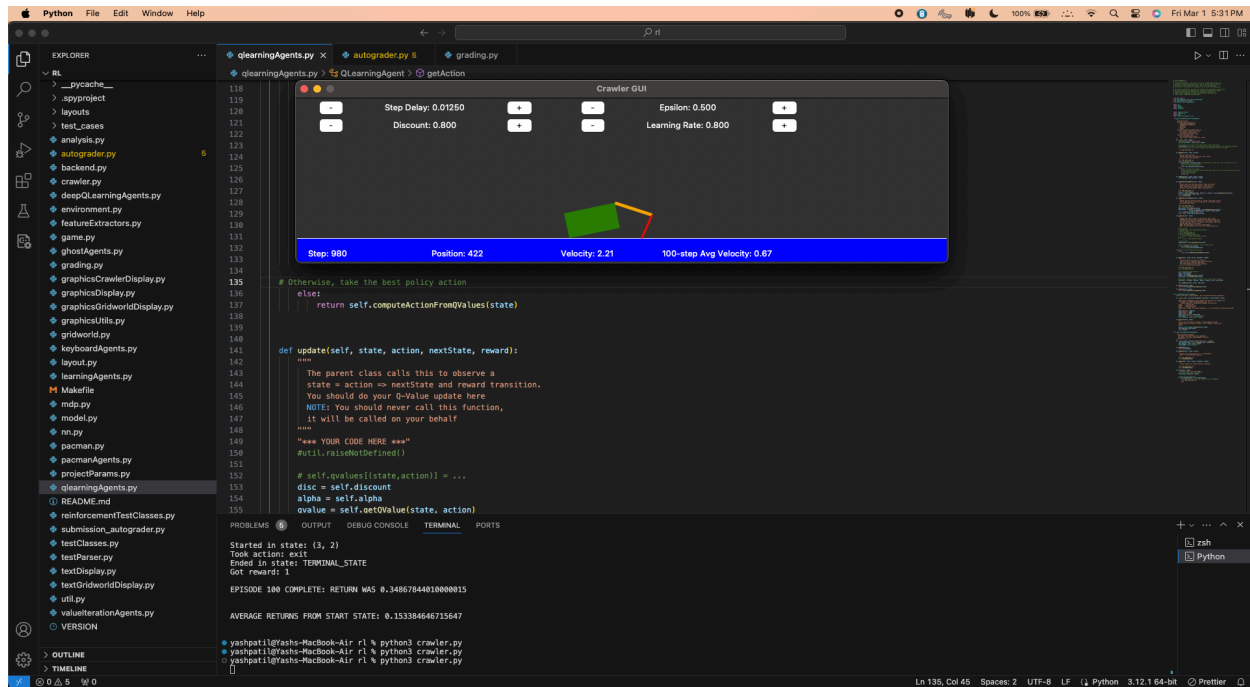
Your grades are NOT yet registered. To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

Task B

Experiment 1



Experiment 2



Experiment 3

