# 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`.

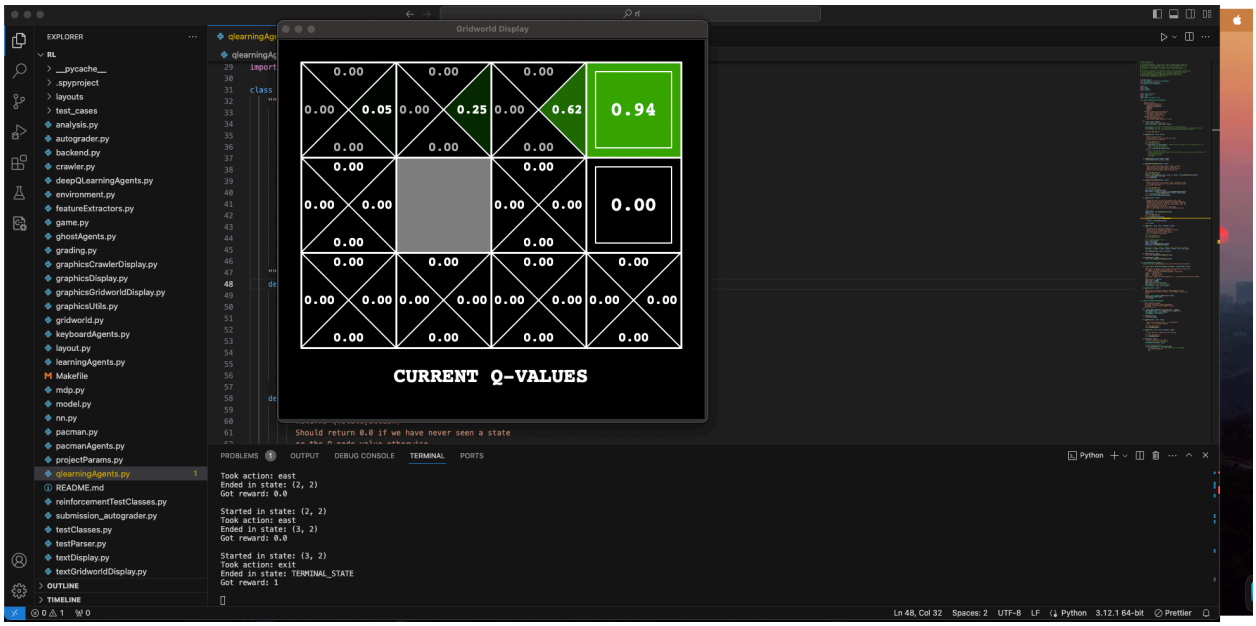
python

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

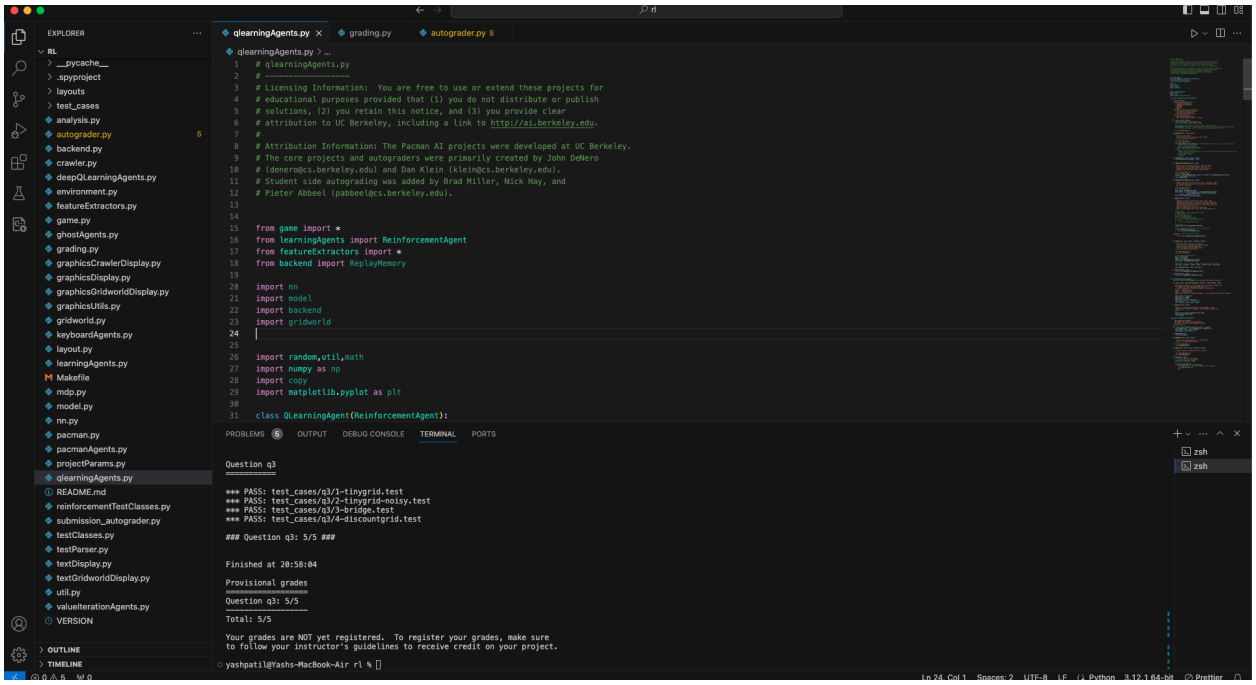
These implementations demonstrate how the Q-learning algorithm updates Q-values and selects actions based on those values while exploring the environment and exploiting learned knowledge.

**Task A**

Experiment 1

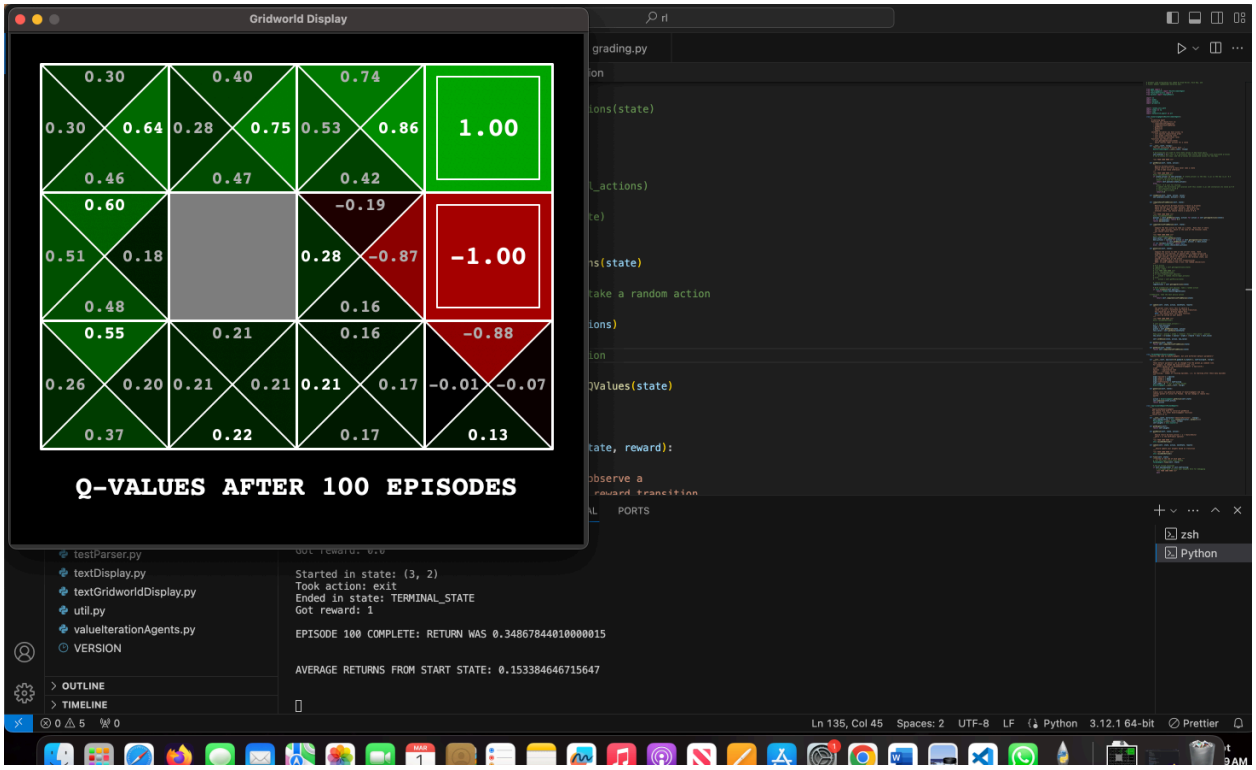


Experiment 2

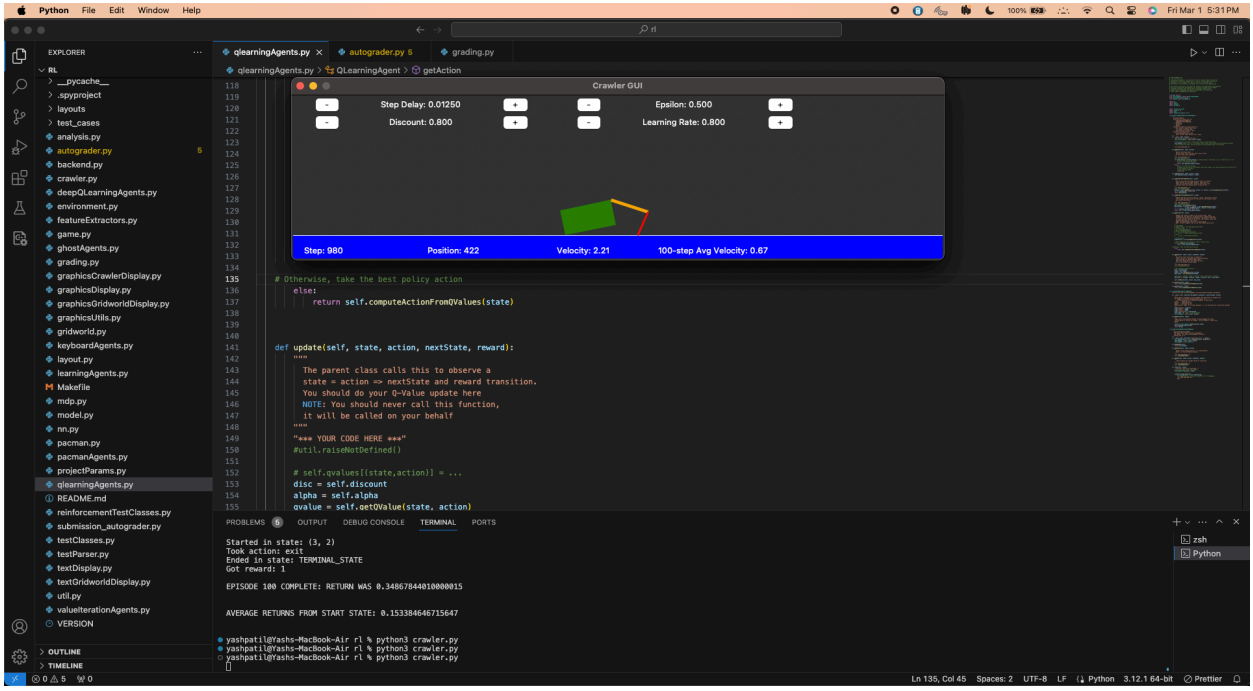



## Task B

Experiment 1



## Experiment 2




## Experiment 3