COEN 166 Artificial Intelligence
Lab Assignment #5 - QLearning
Name: Eva Stenberg & Kian Malakooti ID: 00001576722 & 00001575676

Task: Minimax Agent
paste your code:

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

  def setQValue(self, state, action, value):
    self.qvalues[(state, action)] = value
```

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

Function: computeActionFromQValues
```
  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.

      """
      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)
```

Function: getAction

```
   def getAction(self, state):
      """
        Compute the action to take in the current state.  With
        probability self.epsilon, we should take a random action and
        take the best policy action otherwise.  Note that if there are
        no legal actions, which is the case at the terminal state, you
        should choose None as the action.
        HINT: You might want to use util.flipCoin(prob)
        HINT: To pick randomly from a list, use random.choice(list)
      """
      # Pick Action
      legal_actions = self.getLegalActions(state) #gets available legal actions from the current
state
      action = None #sets actions to none


      if util.flipCoin(self.epsilon): #if random probability function to take random action
         action = random.choice(legal_actions)
      else:
         action = self.getPolicy(state) #takes best policy action otherwise

      return action
```

Function: update

```
   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

  next_value = max[a'] Q(s', a')

    - Q(s, a) = (1-alpha) * Q(s, a) + alpha * (R(s,a,s') + disc * max{a'}[Q(s',a')])

"""
disc = self.discount
alpha = self.alpha
qvalue = self.getQValue(state, action)
next_value = self.getValue(nextState)

#new_value = qvalue + alpha * (reward + disc * next_value - qvalue)
new_value = (1-alpha) * qvalue + alpha * (reward + disc * next_value)

self.setQValue(state, action, new_value)
```

Comment:

Lab 4 Contributions of each group member (in percentage):
Kian Malakooti 50% Eva Stenberg 50%

Explain how you implemented the following functions in your qlearningAgents.py code:

• getQValue

This function returns Q(state,action) by checking if the state has been seen before in self.qvalues. If it has not, it returns 0.0.

• computeValueFromQValues

This function returns max_action Q(state,action) which checks if the max action is over legal actions. It does this by getting the Qvalue from legal actions. If there are no legal actions, meaning it is at the terminal state, then it returns 0.0

• computeActionFromQValues

This function first sets the best value to the current state's value and checks for each legal action for the current state. It then checks the getQValue function to see if the state and action is the best value we got earlier and if it is, sets it to best action. If there are no best actions, it returns None otherwise it will select a random choice from the best action list.

- **getAction**

This function gets the legal actions and sets action to None. After, it takes a random probability to take a random action. If not, it takes the best policy action otherwise and returns the action.

- **update**

Simply calculates the new_value value with the equation, new_value = (1-alpha) * qvalue + alpha * (reward + disc * next_value).