

## Part 1

### ■ Minimax Search

```
""" YOUR CODE HERE """
# Begin your code
def minValue(state, agentIndex, depth):
    # for agent's actions
    legalActions = state.getLegalActions(agentIndex)
    # 已經沒有可以走的
    if not legalActions:
        return self.evaluationFunction(state)
    minV = float('inf')
    for action in legalActions:
        # 如果鬼走完了下一輪就是pacman
        if agentIndex == state.getNumAgents() - 1:
            value = maxValue(state.getNextState(agentIndex, action), depth)
        else:
            value = minValue(state.getNextState(agentIndex, action), agentIndex + 1, depth)
        if value is not None and value < minV:
            minV = value
    return minV

def maxValue(state, depth):
    # for pacman's actions
    legalActions = state.getLegalActions(0)
    # 深度有限制或是已經沒有可以走的
    if not legalActions or depth == self.depth:
        return self.evaluationFunction(state)
    maxV = -float('inf')
    for action in legalActions:
        value = minValue(state.getNextState(0, action), 0 + 1, depth + 1)
        if value is not None and value > maxV:
            maxV = value
    return maxV

bestAction = max(gameState.getLegalActions(0),
                  key=lambda action: minValue(gameState.getNextState(0, action), 1, 1))
return bestAction
# End your code
```

運用兩個函式 `minValue` 和 `maxValue` 來實作 minimax search

`minValue`: (for agent)

Agent 的目標是找出讓 pacman 分數最低的移動方法。

首先先把所有 agent' s legalAction 找出，若沒有的話便 return 現在的分數。

再來用遞迴為 agent 來找最會讓 pacman 分數最低的移動 action (找分數 min)。

若 `agentIndex == state.getNumAgents() - 1` 則代表此輪 agent 會全部走完，下一個便要換 pacman 移動；反之就是讓下一個 agent 移動繼續算 min 值。

`maxValue`: (for pacman)

Pacman 的目標是找到能夠得最多分的移動方法。

首先先把所有 pacman' s legalAction 找出，若沒有 action 或達到深度的限制便 return 現在的分數。

同樣的找出 pacman 所有的 legalAction，沒有的話就 return 現在的分數。

之後用遞迴的方式為 pacman 找出會讓他能得到最高 score 的 action (找 max)，

當他動後代表就換 agent 動，因此呼叫 `minValue` 做下一個深度的 minimax search。

bestAction:

找出 pacman 所有可行的下一步，全部做完後到最深 depth，其所算出的擁有 max score 者就是當下的 bestAction。

## ■ 1-2 Expectimax Search

```
""" YOUR CODE HERE """
# Begin your code
def maxValue(state, depth):
    # for pacman's actions
    legalActions = state.getLegalActions(0)
    # 深度有限制或是已經沒有可以走的
    if not legalActions or depth == self.depth:
        return self.evaluationFunction(state)
    maxV = -float('inf')
    for action in legalActions:
        value = expValue(state.getNextState(0, action), 0 + 1, depth + 1)
        if value is not None and value > maxV:
            maxV = value
    return maxV

def expValue(state, agentIndex, depth):
    # for agent's actions
    legalActions = state.getLegalActions(agentIndex)
    # 已經沒有可以走的
    if not legalActions:
        return self.evaluationFunction(state)

    numAction = len(legalActions)
    value = 0
    # 找所有怪可以走的路
    for action in legalActions:
        newState = state.getNextState(agentIndex, action)
        # 如果鬼走完了下一輪就是pacman
        if agentIndex == state.getNumAgents() - 1:
            value += maxValue(newState, depth)
        else:
            value += expValue(newState, agentIndex + 1, depth)
    return value / float(numAction)

legalActions = gameState.getLegalActions()
bestAction = max(legalActions, key=lambda action: expValue(gameState.getNextState(0, action), 1, 1))
return bestAction
```

用 maxValue 和 expValue 來實作 Expectimax search

maxValue: (for pacman)

Pacman 的目標同樣為找出能得最高分數的移動方法

這邊的做法跟 minimax search 中的一樣，只是遞迴呼叫的是 expValue。

expValue: (計算期望值)

首先先把所有 agent' s legalAction 找出，若沒有的話便 return 現在的分數。

用輪巡每一個 agent 可能的行動，來計算 expectation 值。

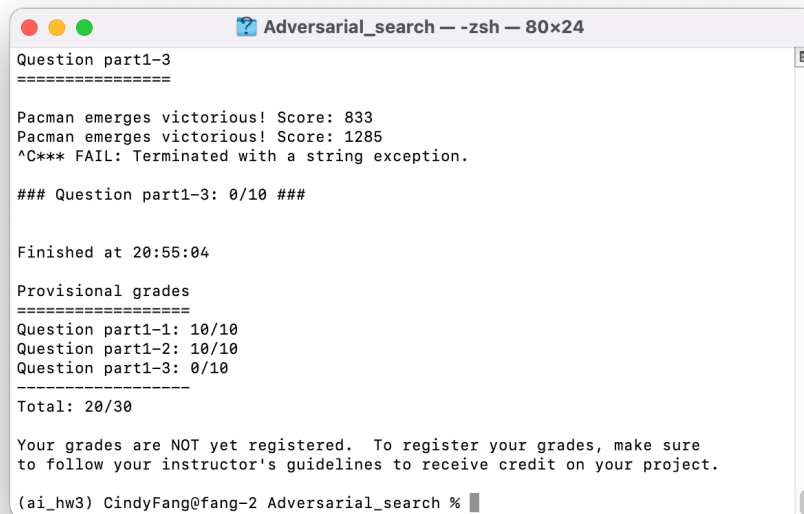
若 agentIndex == state.getNumAgents() - 1 則代表此輪 agent 會全部走完，下一個便要換 pacman 移動，所以便呼叫 maxValue；反之就是讓下一個 agent 行動，計算此次行動的期望值。

最後把算出的期望值求平均回傳。

bestAction:

找出 pacman 所有可行的下一步，全部做完後到最深 depth，其所算出的擁有最高的 expctation 值者就是當下的 bestAction。

## Part1 result (不看 part1-3)



```
Adversarial_search - zsh - 80x24
Question part1-3
=====
Pacman emerges victorious! Score: 833
Pacman emerges victorious! Score: 1285
^C*** FAIL: Terminated with a string exception.

### Question part1-3: 0/10 ###

Finished at 20:55:04

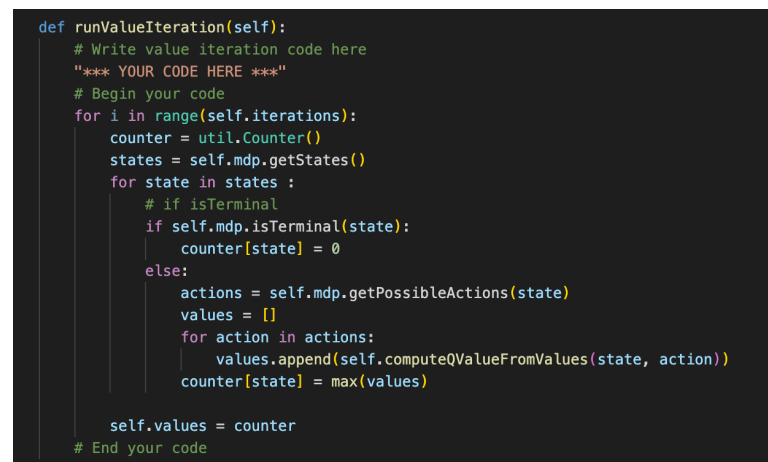
Provisional grades
=====
Question part1-1: 10/10
Question part1-2: 10/10
Question part1-3: 0/10
-----
Total: 20/30

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

(ai_hw3) CindyFang@fang-2 Adversarial_search %
```

## Part 2

### ■ 2-1 Value Iteration



```
def runValueIteration(self):
    # Write value iteration code here
    """** YOUR CODE HERE """
    # Begin your code
    for i in range(self.iterations):
        counter = util.Counter()
        states = self.mdp.getStates()
        for state in states:
            # if isTerminal
            if self.mdp.isTerminal(state):
                counter[state] = 0
            else:
                actions = self.mdp.getPossibleActions(state)
                values = []
                for action in actions:
                    values.append(self.computeQValueFromValues(state, action))
                counter[state] = max(values)
        self.values = counter
    # End your code
```

runValueIteration:

初始化 counter 後用 self.mdp.getStates() 找出所有的 possible states，在每個 state 中用 self.mdp.getPossibleActions 找出他當下所有的 legal actions，最後找出當下所有 action 中最高的 q\_value 存到 counter 中，保存於 self.values 中。

另外在每個 state 裡，如果 isTerminal 是 true 時，其 value 為 0。

```

def computeQValueFromValues(self, state, action):
    """
    Compute the Q-value of action in state from the
    value function stored in self.values.
    """
    """ YOUR CODE HERE """
    # Begin your code
    state_Probs = self.mdp.getTransitionStatesAndProbs(state, action)
    total = 0
    for nextState, prob in state_Probs:
        reward = self.mdp.getReward(state, action, nextState)
        total += prob * (reward + self.discount * self.getValue(nextState))
    return total
    # End your code

```

computeQValueFromValues:

用 self.mdp. getTransitionStatesAndProbs 取出他 nextState 和 prob，並依照 Q 值公式寫出 total。

```

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 """
    # Begin your code

    # terminal
    if self.mdp.isTerminal(state):
        return None

    actions = self.mdp.getPossibleActions(state)
    # 初始
    maxV = -float('inf')
    bestAction = None

    for action in actions:
        q = self.computeQValueFromValues(state, action)
        # update maxV and bestAction
        if q is not None and q > maxV:
            maxV = q
            bestAction = action

    return bestAction
    # End your code

```

computeActionFromValues:

先 check 是否為 terminal state，若為 true 則代表沒有任何 legal actions 並回傳 none。

將當下 state 的所有 legal actions 找出來並用 computeQValueFromValues 找出 actions 中有最大 q\_value 的，其對應的 action 便是當下最好的 bestAction。

## ■ 2-2 & 2-3 Q-learning & epsilon-greedy action selection

### QLearningAgent

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

    "*** YOUR CODE HERE ***"
    # Begin your code
    # A Counter is a dict with default 0
    self.values = util.Counter()
    # End your code
```

初始化 self.values

```
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 ***"
    # Begin your code
    return self.values[state, action]
    # End your code
```

getQValue:

回傳存到 dict 裡面的 values

```
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 ***"
    # Begin your code
    legalActions = self.getLegalActions(state)
    # if not legal => return 0.0
    if not legalActions:
        return 0.0

    return self.getQValue(state, self.getPolicy(state))
    # End your code
```

computeValueFromQValues:

先確定是否有能移動的 action，沒有則 return 0.0，若有的話則運用 getQValue 來取得當下 status,action 的值。

```

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 ***"
    # Begin your code
    legalActions = self.getLegalActions(state)
    if not legalActions:
        return None

    maxV = float('-inf')
    bestAction = None
    for action in legalActions:
        q = self.getQValue(state, action)
        if q is not None and q > maxV:
            maxV = q
            bestAction = action
    return bestAction

    # End your code

```

computeActionFromQValues:

找出當下 state 最好的 action，同樣也是先確認是否有能移動的 actions，若無則回傳 none，若有則找出他最大的 q\_value，其對應的 action 便為當下的 best action。

```

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
    legalActions = self.getLegalActions(state)
    action = None
    "*** YOUR CODE HERE ***"
    # Begin your code
    prob = util.flipCoin(self.epsilon)

    if prob :
        # 隨機選
        action = random.choice(legalActions)
    else:
        action = self.getPolicy(state)

    return action
    # End your code

```

getAction:

先找出所有能夠用的 legal actions，若沒有的話 action 為 none。再用 util.flipCoin() 來決定選擇的 action 是否為隨機選擇，或是其 action 為 best policy。

```

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 """
    # Begin your code
    old_q = self.getQValue(state, action)
    a = self.alpha
    if nextState:
        new_q = (1 - a) * old_q + a * (reward + self.discount * self.getValue(nextState))
    else:
        new_q = (1 - a) * old_q + a * reward

    self.values[state, action] = new_q
    # End your code

```

Update:

先用 getQValue 呼叫出當下的 old\_q，依照是否有 nextState 來分別依照 Q value 公式來更新他的 q\_value。

## ■ 2-4 Approximate Q-learning

### ApproximateQAgent

```

def getQValue(self, state, action):
    """
    Should return Q(state,action) = w * featureVector
    where * is the dotProduct operator
    """
    """ YOUR CODE HERE """
    # Begin your code
    # get weights and feature
    w = self.getWeights()
    featureVector = self.featsExtractor.getFeatures(state, action)
    return w * featureVector
    # End your code

```

getQValue:

取得 weights 之後跟自己的 features 相乘後回傳

```

def update(self, state, action, nextState, reward):
    """
    Should update your weights based on transition
    """
    """ YOUR CODE HERE """
    # Begin your code
    diff = reward + self.discount * self.getValue(nextState) - self.getQValue(state, action)
    features = self.featsExtractor.getFeatures(state, action)
    for feature in features :
        w = self.alpha * diff * features[feature]
        self.weights[feature] += w

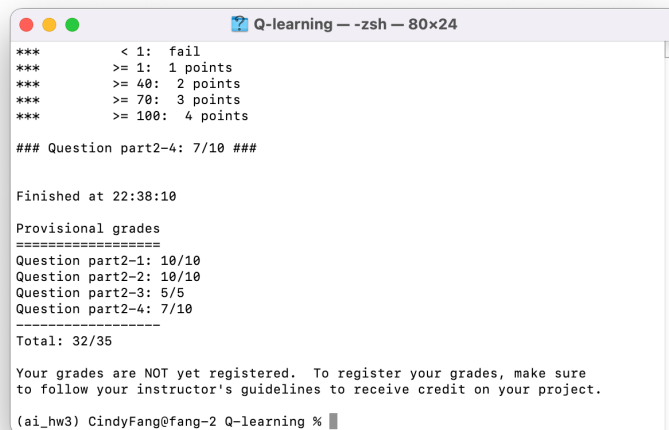
    # End your code

```

update:

先算出他們之間的 diff，在將 alpha\*diff\*feature 一一更新不同的權重。

## Part2 result

A terminal window titled "Q-learning -- zsh -- 80x24" displays the following text:

```
***      < 1:  fail
***      >= 1:  1 points
***      >= 40:  2 points
***      >= 70:  3 points
***      >= 100: 4 points

### Question part2-4: 7/10 ###

Finished at 22:38:10

Provisional grades
=====
Question part2-1: 10/10
Question part2-2: 10/10
Question part2-3: 5/5
Question part2-4: 7/10
-----
Total: 32/35

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

(ai_hw3) CindyFang@fang-2 Q-learning %
```

## Part 3

- What is the difference between On-policy and Off-policy  
On-policy 會將目標 policy 跟行為 policy 用同一種方法去學習  
Off-policy 會將行為策略跟目標策略分開
- Briefly explain value-based, policy-based and Actor-Critic. Also, describe the value function  $V \pi(S)$   
Value-based: 依照輸入選取 action，在訓練時使用 epsilon-greedy，並測試時是選取其 Q 值的 max。  
Policy-based: 依照輸入選取 action，訓練時根據機率去選取動作，而好的動作會有更高的選取機率，更容易被選中。  
Actor-Critic: 結合 policy and value based 兩者，用 policy network 的 actor 來選擇 action，用 critic 來評估 actor 好壞，預測 actor 的 reward 來更新整體模型。  
 $V \pi(S)$ : 遵循  $\pi$  在此 state  $s$  長期累積下來的 expectation 值。
- What is the difference between Monte-Carlo (MC) based approach and Temporal-difference (TD) approach for estimating  $V \pi(S)$ .  
MC 的變異數會比較大，MC 是運用回歸估計 reward，累積的 reward 是每一步的和，但每一步都會有隨機性所以也會被累計上去。  
TD 變異數較小，是用遞迴來推估出來的，可以即時推估下一步的 reward。



- Describe State-action value function  $Q \pi(s,a)$  and the relationship between  $V \pi(S)$  in Q-learning.

$Q \pi(s,a)$ : 在  $s$  狀態中，根據  $a$  動作的累積期望值。

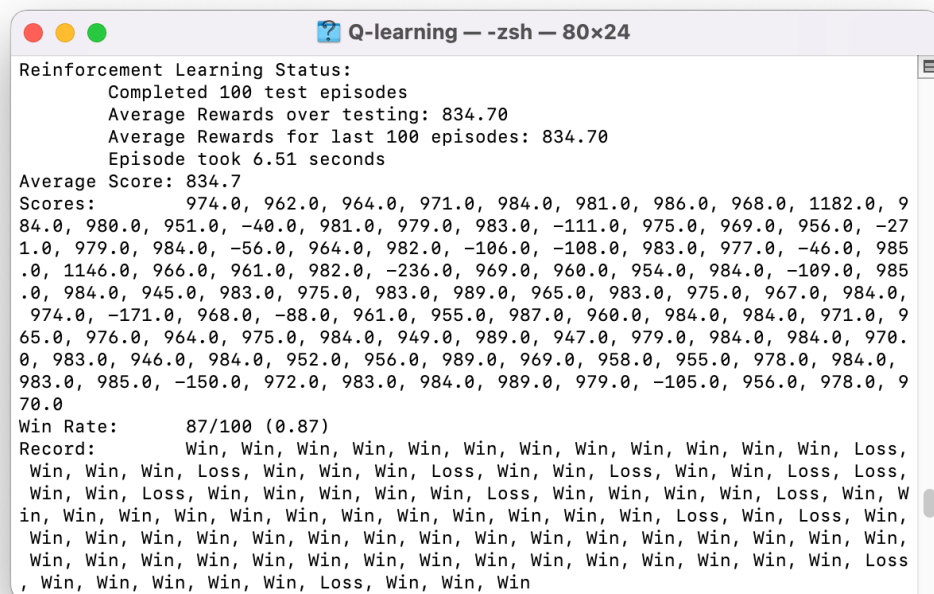
relationship between  $V \pi(S)$ :  $V \pi(S)$  為當時期望值的 max 值，我們可以透過策略更新找到更好的  $Q \pi'(s,a)$  來使  $Q \pi'(s,a) > V \pi(S)$ 。

- Explain what is different between DQN and Q-learning.

Q-learning 運用輸出的  $Q$  值表來找到 max 值去做決策。

DQN 運用神經網路來計算  $Q$  值，並多了 experience replay 和 fixed target 來幫助收斂。

Result between Q-learning and DQN



```

Reinforcement Learning Status:
  Completed 100 test episodes
  Average Rewards over testing: 834.70
  Average Rewards for last 100 episodes: 834.70
  Episode took 6.51 seconds
Average Score: 834.7
Scores:      974.0, 962.0, 964.0, 971.0, 984.0, 981.0, 986.0, 968.0, 1182.0, 9
84.0, 980.0, 951.0, -40.0, 981.0, 979.0, 983.0, -111.0, 975.0, 969.0, 956.0, -27
1.0, 979.0, 984.0, -56.0, 964.0, 982.0, -106.0, -108.0, 983.0, 977.0, -46.0, 985
.0, 1146.0, 966.0, 961.0, 982.0, -236.0, 969.0, 960.0, 954.0, 984.0, -109.0, 985
.0, 984.0, 945.0, 983.0, 975.0, 983.0, 989.0, 965.0, 983.0, 975.0, 967.0, 984.0,
974.0, -171.0, 968.0, -88.0, 961.0, 955.0, 987.0, 960.0, 984.0, 984.0, 971.0, 9
65.0, 976.0, 964.0, 975.0, 984.0, 949.0, 989.0, 947.0, 979.0, 984.0, 984.0, 970.
0, 983.0, 946.0, 984.0, 952.0, 956.0, 989.0, 969.0, 958.0, 955.0, 978.0, 984.0,
983.0, 985.0, -150.0, 972.0, 983.0, 984.0, 989.0, 979.0, -105.0, 956.0, 978.0, 9
70.0
Win Rate:      87/100 (0.87)
Record:      Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss,
Win, Win, Win, Loss, Win, Win, Win, Loss, Win, Win, Loss, Win, Win, Loss, Loss,
Win, Win, Loss, Win, Win, Win, Win, Win, Loss, Win, Win, Win, Win, Loss, Win, W
in, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Loss, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win,
Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Win, Loss
, Win, Win, Win, Win, Win, Loss, Win, Win, Win

```

```
DQN --zsh-- 80x24
Episode no = 19; won: False; Q(s,a) = 233.6131091095629; reward = 149.0; and epsilon = 0.0
Episode no = 20; won: True; Q(s,a) = 238.78422786274464; reward = 720.0; and epsilon = 0.0
Pacman emerges victorious! Score: 1568
Episode no = 21; won: True; Q(s,a) = 222.24648030736552; reward = 765.0; and epsilon = 0.0
Pacman emerges victorious! Score: 1776
Episode no = 22; won: True; Q(s,a) = 227.66284286840983; reward = 813.0; and epsilon = 0.0
Pacman emerges victorious! Score: 1345
Episode no = 23; won: True; Q(s,a) = 216.51490663684746; reward = 691.0; and epsilon = 0.0
Pacman emerges victorious! Score: 1747
Episode no = 24; won: True; Q(s,a) = 248.50393196214662; reward = 751.0; and epsilon = 0.0
Pacman died! Score: -258
Episode no = 25; won: False; Q(s,a) = 235.08663144213637; reward = -524.0; and epsilon = 0.0
Average Score: 1235.6
Scores:      1568.0, 1776.0, 1345.0, 1747.0, -258.0
Win Rate:    4/5 (0.80)
Record:      Win, Win, Win, Win, Loss
(ai_hw3) CindyFang@fang-2 DQN %
```

DQN 的 average score 比起 Q-learning 的好很多