# 強化學習概念 Hw06

# Introduction

- 強化學習 (Reinforcement learning) 潛力無窮，能解決許多開發應用上面臨的艱難決策問題，包括產業自動化、自主駕駛、電玩競技遊戲以及機器人等，因此備受矚目。

- 強化學習是機器學習 (Machine learning) 的一種，指的是電腦透過與一個**動態(dynamic) 環境**不斷重複地互動，來學習正確地執行一項任務。這種**嘗試錯誤(trial-and-error) 的學習方法**，使電腦在沒有人類干預、沒有被寫入明確的執行任務程式下，就能夠做出一系列的決策。

- 最著名的強化學習案例就是 **AlphaGo**，它是第一支打敗人類圍棋比賽世界冠軍的電腦程式。

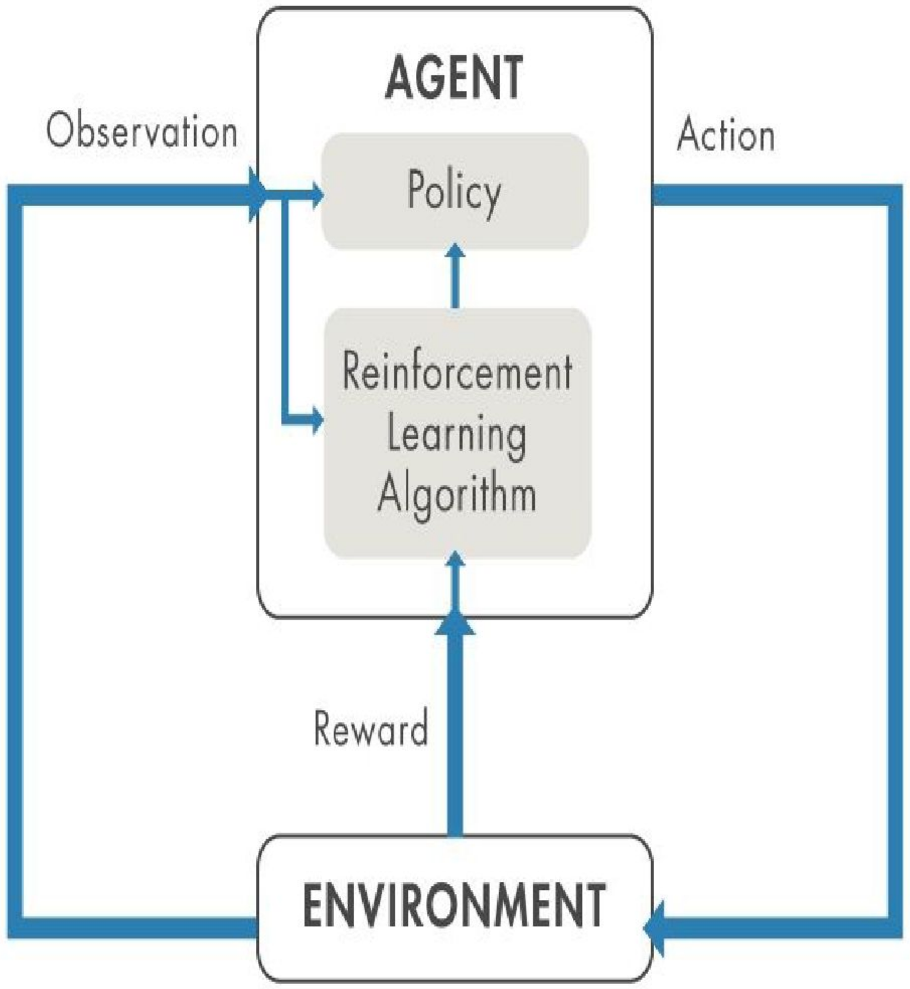
- 強化學習的運作主要是仰賴動態環境中的資料 -- 也就是會隨著外部條件變化而改變的資料。強化學習演算法的目標，即是於找出能夠產生最佳結果的策略。強化學習之所以能達成目標，是藉著軟體當中被稱為主體 (agent) 的部分在環境中進行探索、互動和學習的方法。

# Element

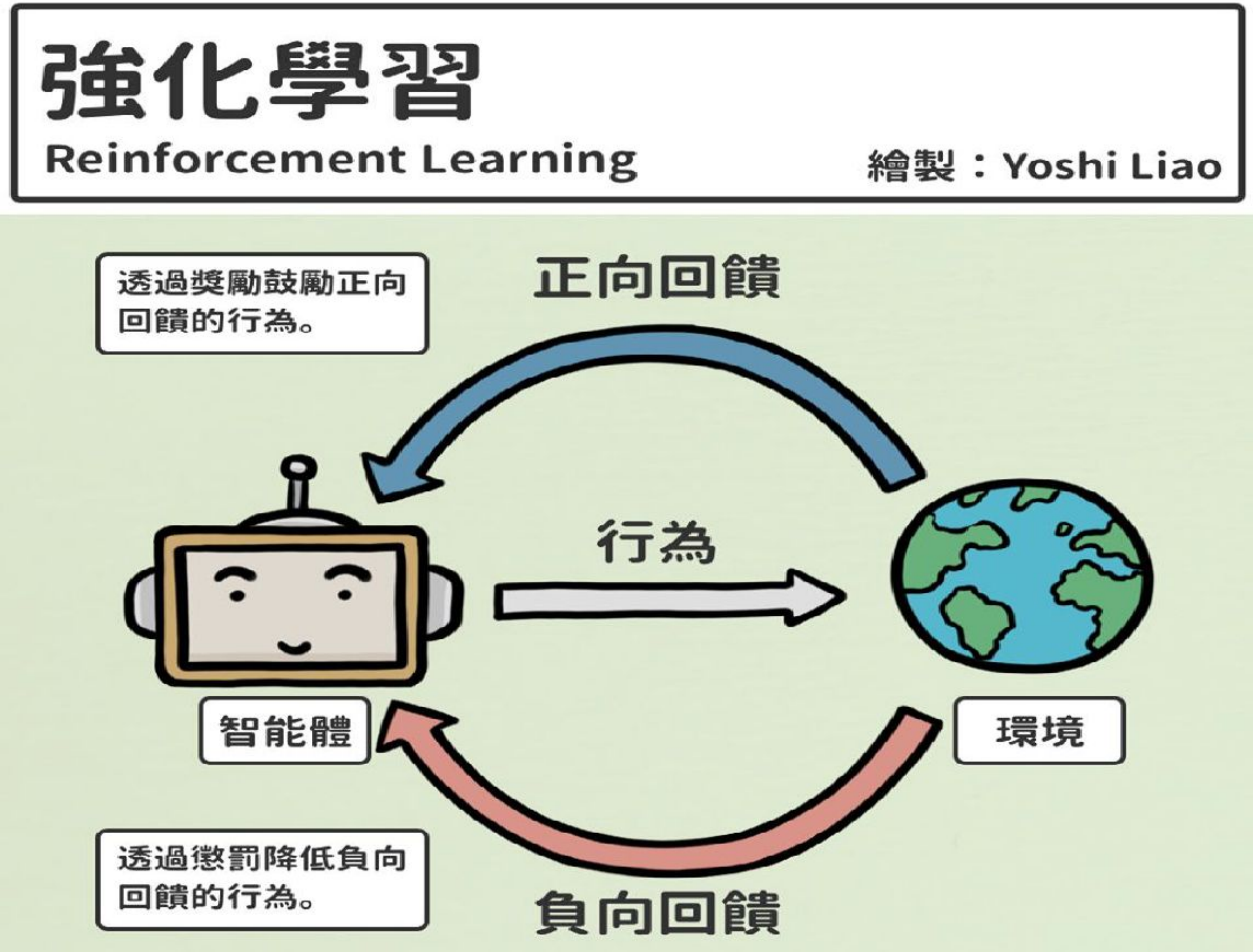
- 自助停車(self-parking)　是自動駕駛功能中極為重要的一環，目標是要讓**車輛中的電腦 (主體，agent)** 能準確地尋找位置並將車輛停入正確的停車格。

- 在以下的範例中，環境(environment)指的是主體之外的所有事物—比如車輛本身的動態、附近的車輛、天候條件等等。訓練過程中，主體使用從各種感測器如攝影機、GPS、光學雷達(LiDAR)以及其他感測器讀取的資料來產生**駕駛、煞車、與加速指令(動作，action)**。

- 為了學習如何從觀察去產生正確的動作(也就是策略調整，policy　tuning)，主體會不斷反覆地嘗試錯誤來試著停車，而正確的動作會得到一個**獎賞(reward)**。



# Reinforcement learning



# Introduction to Q-Learning and Q-Table

## 1. Q-learning :

- Q-learning 是強化學習的一種方法，主要是透過記錄學習過的策略，來告訴 Agent，什麼情況下要對應採取什麼 Action 會得到最大的獎勵 Reward。
- Q-learning 最基本的應用方式，就是將對應行動的獎勵值存在一個 Q-table 中。

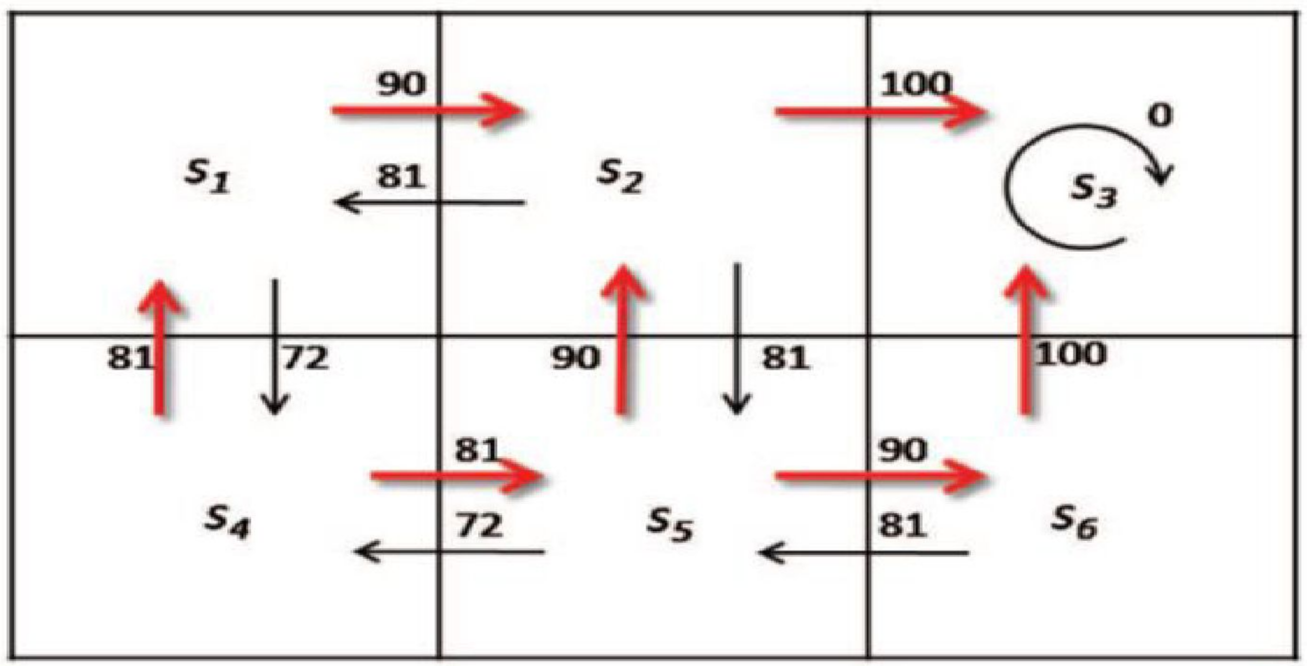
## 2. Q-table :

- 簡單來說就是一個查詢表，用來計算某狀態下做某行為後未來可以期望得到最大的 Reward 為多少，這個表可以引導我們選出每個狀態 state 下，最好的行為 Action。

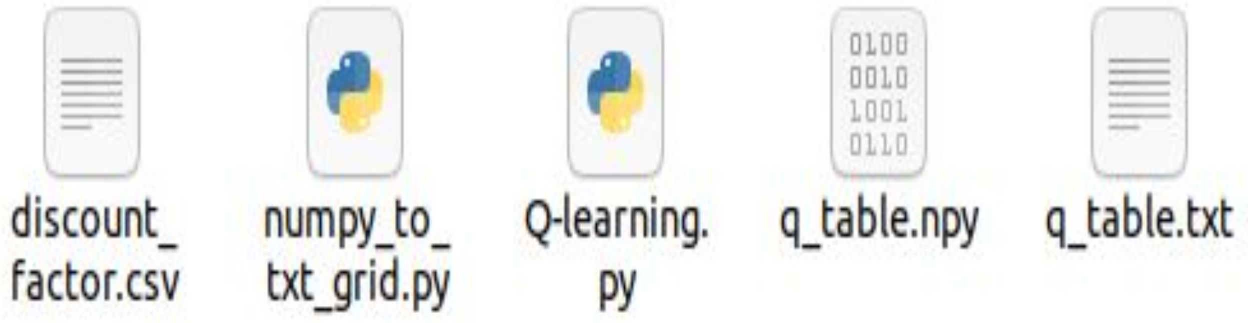
# Q-learning

- $\alpha$ is the learning rate ($0 < \alpha < 1$). $\gamma$ is the discount factor ($0 < \gamma < 1$).
- When the value of $\gamma$ is larger, the long-term rewards obtained in the future will receive more attention; the smaller the value, the more current rewards will be considered.
- $R$ is the reward; it acts on each state and will receive the corresponding reward.

$$Q(s,a)^{new} = (1-\alpha) * Q(s,a)^{old} + \alpha[r + \gamma * max_{a'} Q(s',a')]$$



# Q-learning 檔案




| 檔案名稱 | 類型 |
|---|---|
| discount_factor.csv | Excel 檔案 |
| Q-learning.py | Python 檔案 |
| q_table.npy | Numpy 檔案 |
| numpy_to_txt_grid.py | Python 檔案 |
| q_table.txt | 文字檔案 |

# Q-learning 程式說明

```
# set the rows and columns length
BOARD_ROWS = 4
BOARD_COLS = 6

# initalise start, win and lose states
START = (0, 0)
WIN_STATE = (3, 5)
```

# Q-learning 程式說明

Column

STAR
T

| Row | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | (0, 0) | (0, 1) | (0, 2) | (0, 3) | (0, 4) | (0, 5) |
| 1 | (1, 0) | (1, 1) | (1, 2) | (1, 3) | (1, 4) | (1, 5) |
| 2 | (2, 0) | (2, 1) | (2, 2) | (2, 3) | (2, 4) | (2, 5) |
| 3 | (3, 0) | (3, 1) | (3, 2) | (3, 3) | (3, 4) | (3, 5) |

WIN_STATE

# Q-learning 程式說明

```python
class State:
    def __init__(self, state=START):
        self.state = state
        self.isEnd = False

    def getReward(self):
        if self.state == WIN_STATE:
            return 100
        else:
            return 0

    def isEndFunc(self):
        if self.state == WIN_STATE:
            self.isEnd = True

    def nxtPosition(self, action):
        if action == 0:
            nxtState = (self.state[0] - 1, self.state[1])         # up
        elif action == 1:
            nxtState = (self.state[0] + 1, self.state[1])         # down
        elif action == 2:
            nxtState = (self.state[0], self.state[1] - 1)         # left
        elif action == 3:
            nxtState = (self.state[0], self.state[1] + 1)         # right
        elif action == 4:
            nxtState = (self.state[0] - 1, self.state[1] - 1)     # up-left
        elif action == 5:
            nxtState = (self.state[0] - 1, self.state[1] + 1)     # up-right
        elif action == 6:
            nxtState = (self.state[0] + 1, self.state[1] - 1)     # down-left
        elif action == 7:
            nxtState = (self.state[0] + 1, self.state[1] + 1)     # down-right

        if (nxtState[0] >= 0) and (nxtState[0] < BOARD_ROWS) and (nxtState[1] >= 0) and (nxtState[1] < BOARD_COLS):
            return nxtState

        return self.state
```

# Q-learning 程式說明

```python
# class agent to implement reinforcement learning through grid
class Agent:
    def __init__(self):
        # inialise states and actions
        self.states = []
        self.actions = [0, 1, 2, 3, 4, 5, 6, 7]  # up, down, left, right, up-left, up-right, down-left, down-right
        self.State = State()
        self.alpha = 0.8
        self.epsilon = 0.5
        self.isEnd = self.State.isEnd

        # array to retain reward values for plot
        self.plot_reward = []

        # initalise Q values as a dictionary for current and new
        self.Q = {}
        self.new_Q = {}
        self.rewards = 0

        # initalise all Q values across the board to 0, print these values
        for i in range(BOARD_ROWS):
            for j in range(BOARD_COLS):
                for k in range(len(self.actions)):
                    self.Q[(i, j, k)] = 0
                    self.new_Q[(i, j, k)] = 0
        # print(self.Q)
```

# Q-learning 程式說明

```
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|               (0,0)|               (0,1)|               (0,2)|               (0,3)|               (0,4)|               (0,5)|
|↑ Q(0,0,0) = 0      |↑ Q(0,1,0) = 0      |↑ Q(0,2,0) = 0      |↑ Q(0,3,0) = 0      |↑ Q(0,4,0) = 0      |↑ Q(0,5,0) = 0      |
|↓ Q(0,0,1) = 0      |↓ Q(0,1,1) = 0      |↓ Q(0,2,1) = 0      |↓ Q(0,3,1) = 0      |↓ Q(0,4,1) = 0      |↓ Q(0,5,1) = 0      |
|← Q(0,0,2) = 0      |← Q(0,1,2) = 0      |← Q(0,2,2) = 0      |← Q(0,3,2) = 0      |← Q(0,4,2) = 0      |← Q(0,5,2) = 0      |
|→ Q(0,0,3) = 0      |→ Q(0,1,3) = 0      |→ Q(0,2,3) = 0      |→ Q(0,3,3) = 0      |→ Q(0,4,3) = 0      |→ Q(0,5,3) = 0      |
|↖ Q(0,0,4) = 0      |↖ Q(0,1,4) = 0      |↖ Q(0,2,4) = 0      |↖ Q(0,3,4) = 0      |↖ Q(0,4,4) = 0      |↖ Q(0,5,4) = 0      |
|↗ Q(0,0,5) = 0      |↗ Q(0,1,5) = 0      |↗ Q(0,2,5) = 0      |↗ Q(0,3,5) = 0      |↗ Q(0,4,5) = 0      |↗ Q(0,5,5) = 0      |
|↙ Q(0,0,6) = 0      |↙ Q(0,1,6) = 0      |↙ Q(0,2,6) = 0      |↙ Q(0,3,6) = 0      |↙ Q(0,4,6) = 0      |↙ Q(0,5,6) = 0      |
|↘ Q(0,0,7) = 0      |↘ Q(0,1,7) = 0      |↘ Q(0,2,7) = 0      |↘ Q(0,3,7) = 0      |↘ Q(0,4,7) = 0      |↘ Q(0,5,7) = 0      |
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|               (1,0)|               (1,1)|               (1,2)|               (1,3)|               (1,4)|               (1,5)|
|↑ Q(1,0,0) = 0      |↑ Q(1,1,0) = 0      |↑ Q(1,2,0) = 0      |↑ Q(1,3,0) = 0      |↑ Q(1,4,0) = 0      |↑ Q(1,5,0) = 0      |
|↓ Q(1,0,1) = 0      |↓ Q(1,1,1) = 0      |↓ Q(1,2,1) = 0      |↓ Q(1,3,1) = 0      |↓ Q(1,4,1) = 0      |↓ Q(1,5,1) = 0      |
|← Q(1,0,2) = 0      |← Q(1,1,2) = 0      |← Q(1,2,2) = 0      |← Q(1,3,2) = 0      |← Q(1,4,2) = 0      |← Q(1,5,2) = 0      |
|→ Q(1,0,3) = 0      |→ Q(1,1,3) = 0      |→ Q(1,2,3) = 0      |→ Q(1,3,3) = 0      |→ Q(1,4,3) = 0      |→ Q(1,5,3) = 0      |
|↖ Q(1,0,4) = 0      |↖ Q(1,1,4) = 0      |↖ Q(1,2,4) = 0      |↖ Q(1,3,4) = 0      |↖ Q(1,4,4) = 0      |↖ Q(1,5,4) = 0      |
|↗ Q(1,0,5) = 0      |↗ Q(1,1,5) = 0      |↗ Q(1,2,5) = 0      |↗ Q(1,3,5) = 0      |↗ Q(1,4,5) = 0      |↗ Q(1,5,5) = 0      |
|↙ Q(1,0,6) = 0      |↙ Q(1,1,6) = 0      |↙ Q(1,2,6) = 0      |↙ Q(1,3,6) = 0      |↙ Q(1,4,6) = 0      |↙ Q(1,5,6) = 0      |
|↘ Q(1,0,7) = 0      |↘ Q(1,1,7) = 0      |↘ Q(1,2,7) = 0      |↘ Q(1,3,7) = 0      |↘ Q(1,4,7) = 0      |↘ Q(1,5,7) = 0      |
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|               (2,0)|               (2,1)|               (2,2)|               (2,3)|               (2,4)|               (2,5)|
|↑ Q(2,0,0) = 0      |↑ Q(2,1,0) = 0      |↑ Q(2,2,0) = 0      |↑ Q(2,3,0) = 0      |↑ Q(2,4,0) = 0      |↑ Q(2,5,0) = 0      |
|↓ Q(2,0,1) = 0      |↓ Q(2,1,1) = 0      |↓ Q(2,2,1) = 0      |↓ Q(2,3,1) = 0      |↓ Q(2,4,1) = 0      |↓ Q(2,5,1) = 0      |
|← Q(2,0,2) = 0      |← Q(2,1,2) = 0      |← Q(2,2,2) = 0      |← Q(2,3,2) = 0      |← Q(2,4,2) = 0      |← Q(2,5,2) = 0      |
|→ Q(2,0,3) = 0      |→ Q(2,1,3) = 0      |→ Q(2,2,3) = 0      |→ Q(2,3,3) = 0      |→ Q(2,4,3) = 0      |→ Q(2,5,3) = 0      |
|↖ Q(2,0,4) = 0      |↖ Q(2,1,4) = 0      |↖ Q(2,2,4) = 0      |↖ Q(2,3,4) = 0      |↖ Q(2,4,4) = 0      |↖ Q(2,5,4) = 0      |
|↗ Q(2,0,5) = 0      |↗ Q(2,1,5) = 0      |↗ Q(2,2,5) = 0      |↗ Q(2,3,5) = 0      |↗ Q(2,4,5) = 0      |↗ Q(2,5,5) = 0      |
|↙ Q(2,0,6) = 0      |↙ Q(2,1,6) = 0      |↙ Q(2,2,6) = 0      |↙ Q(2,3,6) = 0      |↙ Q(2,4,6) = 0      |↙ Q(2,5,6) = 0      |
|↘ Q(2,0,7) = 0      |↘ Q(2,1,7) = 0      |↘ Q(2,2,7) = 0      |↘ Q(2,3,7) = 0      |↘ Q(2,4,7) = 0      |↘ Q(2,5,7) = 0      |
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|               (3,0)|               (3,1)|               (3,2)|               (3,3)|               (3,4)|               (3,5)|
|↑ Q(3,0,0) = 0      |↑ Q(3,1,0) = 0      |↑ Q(3,2,0) = 0      |↑ Q(3,3,0) = 0      |↑ Q(3,4,0) = 0      |↑ Q(3,5,0) = 0      |
|↓ Q(3,0,1) = 0      |↓ Q(3,1,1) = 0      |↓ Q(3,2,1) = 0      |↓ Q(3,3,1) = 0      |↓ Q(3,4,1) = 0      |↓ Q(3,5,1) = 0      |
|← Q(3,0,2) = 0      |← Q(3,1,2) = 0      |← Q(3,2,2) = 0      |← Q(3,3,2) = 0      |← Q(3,4,2) = 0      |← Q(3,5,2) = 0      |
|→ Q(3,0,3) = 0      |→ Q(3,1,3) = 0      |→ Q(3,2,3) = 0      |→ Q(3,3,3) = 0      |→ Q(3,4,3) = 0      |→ Q(3,5,3) = 0      |
|↖ Q(3,0,4) = 0      |↖ Q(3,1,4) = 0      |↖ Q(3,2,4) = 0      |↖ Q(3,3,4) = 0      |↖ Q(3,4,4) = 0      |↖ Q(3,5,4) = 0      |
|↗ Q(3,0,5) = 0      |↗ Q(3,1,5) = 0      |↗ Q(3,2,5) = 0      |↗ Q(3,3,5) = 0      |↗ Q(3,4,5) = 0      |↗ Q(3,5,5) = 0      |
|↙ Q(3,0,6) = 0      |↙ Q(3,1,6) = 0      |↙ Q(3,2,6) = 0      |↙ Q(3,3,6) = 0      |↙ Q(3,4,6) = 0      |↙ Q(3,5,6) = 0      |
|↘ Q(3,0,7) = 0      |↘ Q(3,1,7) = 0      |↘ Q(3,2,7) = 0      |↘ Q(3,3,7) = 0      |↘ Q(3,4,7) = 0      |↘ Q(3,5,7) = 0      |
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
```

# Q-learning 程式說明

```python
    def Action(self):
        # random value vs epsilon
        rnd = random.random()
        # set arbitraty low value to compare with Q values to find max
        mx_nxt_reward = -10
        action = None

        # find max Q value over actions
        if rnd > self.epsilon:
            # iterate through actions, find Q-value and choose best
            for k in self.actions:
                i, j = self.State.state
                nxt_reward = self.Q[(i, j, k)]

                if nxt_reward >= mx_nxt_reward:
                    action = k
                    mx_nxt_reward = nxt_reward
        # else choose random action
        else:
            action = np.random.choice(self.actions)

        # select the next state based on action chosen
        position = self.State.nxtPosition(action)
        return position, action
```

# Q-learning 程式說明

```
    # Q-learning Algorithm
    def Q_Learning(self, episodes):
        df_factor_path = "discount_factor.csv"
        df_factor = pd.read_csv(df_factor_path, index_col="Grid 座標")
        x = 0
        # iterate through best path for each episode
        while x < episodes:
            # check if state is end
            if self.isEnd:
                # get current rewrard and add to array for plot
                reward = self.State.getReward()
                self.rewards += reward
                self.plot_reward.append(self.rewards)

                # get state, assign reward to each Q_value in state
                i, j = self.State.state
                for a in self.actions:
                    self.new_Q[(i, j, a)] = round(reward, 3)

                # reset state
                self.State = State()
                self.isEnd = self.State.isEnd

                # set rewards to zero and iterate to next episode
                self.rewards = 0
                x += 1
```

# discount_factor.csv 檔案說明

| Grid 座標 | discount factor |
|---|---|
| 0_0 | 0.3/0.9/0.3/0.69449/0.3/0.3/0.3/0.3 |
| 0_1 | 0.3/0.43638/0.68073/0.55197/0.3/0.55602/0.3/0.32028 |
| 0_2 | 0.3/0.53606/0.55858/0.5093/0.3/0.3/0.3/0.3 |
| 0_3 | 0.3/0.3/0.72106/0.60534/0.3/0.32053/0.3/0.36304 |
| 0_4 | 0.3/0.54984/0.50086/0.61222/0.3/0.3/0.3/0.3 |
| 0_5 | 0.3/0.34667/0.9/0.3/0.3/0.36411/0.3/0.3 |
| 1_0 | 0.41314/0.9/0.3/0.46518/0.3/0.3/0.3/0.39871 |
| 1_1 | 0.40229/0.59559/0.72101/0.58091/0.3/0.53315/0.30577/0.32393 |
| 1_2 | 0.44158/0.51077/0.50963/0.44838/0.3/0.37714/0.3/0.3 |

# Q-learning 程式說明

$$Q(s,a)^{new} = (1-\alpha) * Q(s,a)^{old} + \alpha[r + \gamma * max_{a'}Q(s',a')]$$

```
            else:
                mx_nxt_value = -10
                next_state, action = self.Action()
                i, j = self.State.state
                reward = self.State.getReward()
                # add reward to rewards for plot
                self.rewards += reward

                # iterate through actions to find max Q value for action based on next state action
                for a in self.actions:
                    now_index = str(i) + "_" + str(j)
                    df = df_factor.loc[now_index, "discount factor"]
                    df = df.split("/")

                    nxtStateAction = (next_state[0], next_state[1], a)
                    q_value = (1 - self.alpha) * self.Q[(i, j, action)] + self.alpha * (
                        reward + float(df[a]) * self.Q[nxtStateAction]
                    )

                    if q_value >= mx_nxt_value:
                        mx_nxt_value = q_value

                # next state is now current state, check if end state
                self.State = State(state=next_state)
                self.State.isEndFunc()
                self.isEnd = self.State.isEnd
                self.new_Q[(i, j, action)] = round(mx_nxt_value, 3)
            self.Q = self.new_Q.copy()
        # print(self.Q)
```

# Q-learning 程式說明

| (0,0) | (0,1) | (0,2) | (0,3) | (0,4) | (0,5) |
|---|---|---|---|---|---|
| ↑ Q(0, 0, 0) = 2.286 | ↑ Q(0, 1, 0) = 1.785 | ↑ Q(0, 2, 0) = 3.030 | ↑ Q(0, 3, 0) = 6.154 | ↑ Q(0, 4, 0) = 7.931 | ↑ Q(0, 5, 0) = 6.195 |
| ↓ Q(0, 0, 1) = 2.540 | ↓ Q(0, 1, 1) = 3.086 | ↓ Q(0, 2, 1) = 4.902 | ↓ Q(0, 3, 1) = 10.685 | ↓ Q(0, 4, 1) = 14.424 | ↓ Q(0, 5, 1) = 14.620 |
| ← Q(0, 0, 2) = 2.286 | ← Q(0, 1, 2) = 1.556 | ← Q(0, 2, 2) = 1.654 | ← Q(0, 3, 2) = 3.667 | ← Q(0, 4, 2) = 5.875 | ← Q(0, 5, 2) = 6.957 |
| → Q(0, 0, 3) = 2.777 | → Q(0, 1, 3) = 3.235 | → Q(0, 2, 3) = 5.728 | → Q(0, 3, 3) = 8.418 | → Q(0, 4, 3) = 8.039 | → Q(0, 5, 3) = 6.261 |
| ↖ Q(0, 0, 4) = 2.286 | ↖ Q(0, 1, 4) = 1.785 | ↖ Q(0, 2, 4) = 3.030 | ↖ Q(0, 3, 4) = 6.154 | ↖ Q(0, 4, 4) = 7.931 | ↖ Q(0, 5, 4) = 6.261 |
| ↗ Q(0, 0, 5) = 2.286 | ↗ Q(0, 1, 5) = 1.785 | ↗ Q(0, 2, 5) = 3.030 | ↗ Q(0, 3, 5) = 6.154 | ↗ Q(0, 4, 5) = 7.931 | ↗ Q(0, 5, 5) = 6.261 |
| ↙ Q(0, 0, 6) = 2.286 | ↙ Q(0, 1, 6) = 2.018 | ↙ Q(0, 2, 6) = 2.847 | ↙ Q(0, 3, 6) = 5.826 | ↙ Q(0, 4, 6) = 10.807 | ↙ Q(0, 5, 6) = 14.008 |
| ↘ Q(0, 0, 7) = 3.883 | ↘ Q(0, 1, 7) = 5.313 | ↘ Q(0, 2, 7) = 10.102 | ↘ Q(0, 3, 7) = 16.952 | ↘ Q(0, 4, 7) = 23.189 | ↘ Q(0, 5, 7) = 6.261 |

| (1,0) | (1,1) | (1,2) | (1,3) | (1,4) | (1,5) |
|---|---|---|---|---|---|
| ↑ Q(1, 0, 0) = 2.286 | ↑ Q(1, 1, 0) = 1.879 | ↑ Q(1, 2, 0) = 3.030 | ↑ Q(1, 3, 0) = 6.408 | ↑ Q(1, 4, 0) = 8.117 | ↑ Q(1, 5, 0) = 7.377 |
| ↓ Q(1, 0, 1) = 2.330 | ↓ Q(1, 1, 1) = 4.064 | ↓ Q(1, 2, 1) = 6.705 | ↓ Q(1, 3, 1) = 18.845 | ↓ Q(1, 4, 1) = 23.319 | ↓ Q(1, 5, 1) = 42.174 |
| ← Q(1, 0, 2) = 2.097 | ← Q(1, 1, 2) = 2.124 | ← Q(1, 2, 2) = 2.606 | ← Q(1, 3, 2) = 7.096 | ← Q(1, 4, 2) = 10.528 | ← Q(1, 5, 2) = 14.008 |
| → Q(1, 0, 3) = 3.657 | → Q(1, 1, 3) = 5.591 | → Q(1, 2, 3) = 9.625 | → Q(1, 3, 3) = 17.652 | → Q(1, 4, 3) = 23.561 | → Q(1, 5, 3) = 21.280 |
| ↖ Q(1, 0, 4) = 2.097 | ↖ Q(1, 1, 4) = 1.648 | ↖ Q(1, 2, 4) = 1.594 | ↖ Q(1, 3, 4) = 4.223 | ↖ Q(1, 4, 4) = 5.969 | ↖ Q(1, 5, 4) = 7.278 |
| ↗ Q(1, 0, 5) = 2.777 | ↗ Q(1, 1, 5) = 3.327 | ↗ Q(1, 2, 5) = 5.457 | ↗ Q(1, 3, 5) = 8.766 | ↗ Q(1, 4, 5) = 8.167 | ↗ Q(1, 5, 5) = 21.280 |
| ↙ Q(1, 0, 6) = 2.097 | ↙ Q(1, 1, 6) = 2.910 | ↙ Q(1, 2, 6) = 3.137 | ↙ Q(1, 3, 6) = 11.025 | ↙ Q(1, 4, 6) = 17.450 | ↙ Q(1, 5, 6) = 21.062 |
| ↘ Q(1, 0, 7) = 3.351 | ↘ Q(1, 1, 7) = 8.687 | ↘ Q(1, 2, 7) = 14.955 | ↘ Q(1, 3, 7) = 27.513 | ↘ Q(1, 4, 7) = 46.695 | ↘ Q(1, 5, 7) = 21.280 |

| (2,0) | (2,1) | (2,2) | (2,3) | (2,4) | (2,5) |
|---|---|---|---|---|---|
| ↑ Q(2, 0, 0) = 2.618 | ↑ Q(2, 1, 0) = 2.615 | ↑ Q(2, 2, 0) = 4.616 | ↑ Q(2, 3, 0) = 11.820 | ↑ Q(2, 4, 0) = 17.009 | ↑ Q(2, 5, 0) = 20.916 |
| ↓ Q(2, 0, 1) = 2.539 | ↓ Q(2, 1, 1) = 3.262 | ↓ Q(2, 2, 1) = 8.618 | ↓ Q(2, 3, 1) = 26.452 | ↓ Q(2, 4, 1) = 41.742 | ↓ Q(2, 5, 1) = 83.584 |
| ← Q(2, 0, 2) = 3.586 | ← Q(2, 1, 2) = 3.227 | ← Q(2, 2, 2) = 3.355 | ← Q(2, 3, 2) = 10.014 | ← Q(2, 4, 2) = 18.159 | ← Q(2, 5, 2) = 20.701 |
| → Q(2, 0, 3) = 5.009 | → Q(2, 1, 3) = 6.996 | → Q(2, 2, 3) = 14.955 | → Q(2, 3, 3) = 24.989 | → Q(2, 4, 3) = 37.320 | → Q(2, 5, 3) = 41.453 |
| ↖ Q(2, 0, 4) = 3.586 | ↖ Q(2, 1, 4) = 1.887 | ↖ Q(2, 2, 4) = 2.681 | ↖ Q(2, 3, 4) = 6.445 | ↖ Q(2, 4, 4) = 10.022 | ↖ Q(2, 5, 4) = 14.008 |
| ↗ Q(2, 0, 5) = 4.003 | ↗ Q(2, 1, 5) = 4.502 | ↗ Q(2, 2, 5) = 8.466 | ↗ Q(2, 3, 5) = 15.776 | ↗ Q(2, 4, 5) = 18.831 | ↗ Q(2, 5, 5) = 41.453 |
| ↙ Q(2, 0, 6) = 3.586 | ↙ Q(2, 1, 6) = 2.063 | ↙ Q(2, 2, 6) = 3.345 | ↙ Q(2, 3, 6) = 12.031 | ↙ Q(2, 4, 6) = 22.149 | ↙ Q(2, 5, 6) = 27.489 |
| ↘ Q(2, 0, 7) = 4.993 | ↘ Q(2, 1, 7) = 8.405 | ↘ Q(2, 2, 7) = 18.946 | ↘ Q(2, 3, 7) = 49.852 | ↘ Q(2, 4, 7) = 56.315 | ↘ Q(2, 5, 7) = 41.453 |

| (3,0) | (3,1) | (3,2) | (3,3) | (3,4) | (3,5) |
|---|---|---|---|---|---|
| ↑ Q(3, 0, 0) = 2.547 | ↑ Q(3, 1, 0) = 2.715 | ↑ Q(3, 2, 0) = 6.802 | ↑ Q(3, 3, 0) = 14.955 | ↑ Q(3, 4, 0) = 27.785 | ↑ Q(3, 5, 0) = 100.000 |
| ↓ Q(3, 0, 1) = 2.292 | ↓ Q(3, 1, 1) = 2.707 | ↓ Q(3, 2, 1) = 8.173 | ↓ Q(3, 3, 1) = 20.961 | ↓ Q(3, 4, 1) = 55.428 | ↓ Q(3, 5, 1) = 100.000 |
| ← Q(3, 0, 2) = 2.292 | ← Q(3, 1, 2) = 1.585 | ← Q(3, 2, 2) = 3.172 | ← Q(3, 3, 2) = 9.534 | ← Q(3, 4, 2) = 29.411 | ← Q(3, 5, 2) = 100.000 |
| → Q(3, 0, 3) = 3.547 | → Q(3, 1, 3) = 6.974 | → Q(3, 2, 3) = 17.968 | → Q(3, 3, 3) = 39.504 | → Q(3, 4, 3) = 74.450 | → Q(3, 5, 3) = 100.000 |
| ↖ Q(3, 0, 4) = 2.292 | ↖ Q(3, 1, 4) = 1.959 | ↖ Q(3, 2, 4) = 3.182 | ↖ Q(3, 3, 4) = 7.935 | ↖ Q(3, 4, 4) = 18.604 | ↖ Q(3, 5, 4) = 100.000 |
| ↗ Q(3, 0, 5) = 3.558 | ↗ Q(3, 1, 5) = 5.805 | ↗ Q(3, 2, 5) = 14.955 | ↗ Q(3, 3, 5) = 19.802 | ↗ Q(3, 4, 5) = 30.862 | ↗ Q(3, 5, 5) = 100.000 |
| ↙ Q(3, 0, 6) = 2.292 | ↙ Q(3, 1, 6) = 2.707 | ↙ Q(3, 2, 6) = 8.173 | ↙ Q(3, 3, 6) = 20.961 | ↙ Q(3, 4, 6) = 55.428 | ↙ Q(3, 5, 6) = 100.000 |
| ↘ Q(3, 0, 7) = 2.292 | ↘ Q(3, 1, 7) = 2.707 | ↘ Q(3, 2, 7) = 8.173 | ↘ Q(3, 3, 7) = 20.961 | ↘ Q(3, 4, 7) = 55.428 | ↘ Q(3, 5, 7) = 100.000 |

# Q-learning 程式說明

```python
def showValues(self, arr):
    outArr = [[0 for _ in range(0, BOARD_COLS)] for _ in range(0, BOARD_ROWS)]
    for i in range(0, BOARD_ROWS):
        print("---------------------------------------------------------------------------------------------------")
        out = "| "
        for j in range(0, BOARD_COLS):
            mx_nxt_value = -10
            for a in self.actions:
                nxt_value = self.Q[(i, j, a)]
                if nxt_value >= mx_nxt_value:
                    mx_nxt_value = nxt_value
            out += str(mx_nxt_value).ljust(6) + " | "
            outArr[i][j] = str(mx_nxt_value)
        print(out)
    print("---------------------------------------------------------------------------------------------------")
```

# Q-learning 程式說明

| | | | | | |
|---|---|---|---|---|---|
| 3.883 | 5.313 | 10.102 | 16.952 | 23.189 | 14.62 |
| 3.657 | 8.687 | 14.955 | 27.513 | 46.695 | 42.174 |
| 5.009 | 8.405 | 18.946 | 49.852 | 56.315 | 83.584 |
| 3.558 | 6.974 | 17.968 | 39.504 | 74.45 | 100 |

# Q-learning 程式說明

| | | | | | |
|---|---|---|---|---|---|
| S | | | | | |
| | ↘ | | | | |
| | | ↘ | | | |
| | | | → | → | G |

# Q-learning 程式說明

```
[-1.       3.657 -1.      5.313 -1.      -1.      -1.       8.687]
```

```python
        # up
        if START[0] - 1 < 0:
            arr[0] = -1
        else:
            arr[0] = outArr[START[0] - 1][START[1]]

        # down
        if START[0] + 1 >= BOARD_ROWS:
            arr[1] = -1
        else:
            arr[1] = outArr[START[0] + 1][START[1]]

        # left
        if START[1] - 1 < 0:
            arr[2] = -1
        else:
            arr[2] = outArr[START[0]][START[1] - 1]

        # right
        if START[1] + 1 >= BOARD_COLS:
            arr[3] = -1
        else:
            arr[3] = outArr[START[0]][START[1] + 1]
```

```python
        # up-left
        if START[0] - 1 < 0 or START[1] - 1 < 0:
            arr[4] = -1
        else:
            arr[4] = outArr[START[0] - 1][START[1] - 1]

        # up-right
        if START[0] - 1 < 0 or START[1] + 1 >= BOARD_COLS:
            arr[5] = -1
        else:
            arr[5] = outArr[START[0] - 1][START[1] + 1]

        # down-left
        if START[0] + 1 >= BOARD_ROWS or START[1] - 1 < 0:
            arr[6] = -1
        else:
            arr[6] = outArr[START[0] + 1][START[1] - 1]

        # down-right
        if START[0] + 1 >= BOARD_ROWS or START[1] + 1 >= BOARD_COLS:
            arr[7] = -1
        else:
            arr[7] = outArr[START[0] + 1][START[1] + 1]

        print(arr)
```

# Q-learning 程式說明

```
if __name__ == "__main__":
    # create agent for 15,000 episdoes implementing a Q-learning algorithm plot and show values.
    episodes = 10000
    ag = Agent()

    filename = "q_table"
    q_value = np.zeros((8), dtype=np.float64)

    print(START, WIN_STATE)

    ag.Q_Learning(episodes)
    ag.showValues(q_value)
    np.save(os.path.join(filename), q_value)
```

# numpy_to_txt_grid.py 程式說明

```python
import numpy as np
import os

fileName = 'q_table.txt'

try:
    os.remove(fileName)
except:
    pass

test = np.load('q_table.npy')
fo = open("q_table.txt", "a+")

string = " ".join(map(str, test))
fo.write(string + "\n")

fo.close()
```

# 作業繳交說明

## ✔ 作業要求：

1. 將 row 改為 8、column 改為 10
2. 改變起點、終點
3. 嘗試產生自己的 discount factor 資料 (隨機生成八個方位的 discount factor, 範圍介於 0 到 1 區間, 數值取至小數點後第三位。)
4. 紀錄最佳路徑策略圖
5. 紀錄完整的 Q-table (不單單紀錄起點的 Q-value)
6. 紀錄 ε (epsilon ) 變化 (x 軸 : ε 參數變化, y 軸 : 執行時間)
7. 檔名內容:學號_HW6.zip (包含 Q-learning.py、discount_factor.csv、numpy_to_txt_grid.py、q_table.npy、q_table.txt、學號_HW6.pdf ) □ 補交檔名 : 學號_HW6-2.zip

繳交檔案 :1. 紙本檔案、2. 電子檔案上傳 FTP、3. 檔名為 : 學號_HW6.zip

上傳 : 120.107.172.19 使用者名稱 : 1132VANET 密碼 : 1132student

# 作業說明 - 紀錄完整的 Q-table (不單單紀錄起點的 Q-value)

```
0_0 -1.0 15.041 -1.0 20.294 -1.0 -1.0 -1.0 20.857
0_1 -1.0 20.857 15.629 17.87 -1.0 -1.0 15.041 21.384
0_2 -1.0 21.384 20.294 16.429 -1.0 -1.0 20.857 28.823
0_3 -1.0 28.823 17.87 15.479 -1.0 -1.0 21.384 24.929
0_4 -1.0 24.929 16.429 26.493 -1.0 -1.0 28.823 24.116
0_5 -1.0 24.116 15.479 18.48 -1.0 -1.0 24.929 30.278
0_6 -1.0 30.278 26.493 18.366 -1.0 -1.0 24.116 38.868
0_7 -1.0 38.868 18.48 25.769 -1.0 -1.0 30.278 34.67
0_8 -1.0 34.67 18.366 27.285 -1.0 -1.0 38.868 25.25
0_9 -1.0 25.25 25.769 -1.0 -1.0 -1.0 34.67 -1.0
1_0 15.629 20.946 -1.0 20.857 -1.0 20.294 -1.0 25.712
1_1 20.294 25.712 15.041 21.384 15.629 17.87 20.946 27.151
1_2 17.87 27.151 20.857 28.823 20.294 16.429 25.712 32.608
1_3 16.429 32.608 21.384 24.929 17.87 15.479 27.151 36.393
1_4 15.479 36.393 28.823 24.116 16.429 26.493 32.608 36.546
1_5 26.493 36.546 24.929 30.278 15.479 18.48 36.393 39.661
1_6 18.48 39.661 24.116 38.868 26.493 18.366 36.546 30.996
1_7 18.366 30.996 30.278 34.67 18.48 25.769 39.661 31.238
1_8 25.769 31.238 38.868 25.25 18.366 27.285 30.996 29.889
1_9 27.285 29.889 34.67 -1.0 25.769 -1.0 31.238 -1.0
2_0 15.041 23.164 -1.0 25.712 -1.0 20.857 -1.0 23.331
2_1 20.857 23.331 20.946 27.151 15.041 21.384 23.164 16.95
2_2 21.384 16.95 25.712 32.608 20.857 28.823 23.331 22.081
2_3 28.823 22.081 27.151 36.393 21.384 24.929 16.95 26.916
2_4 24.929 26.916 32.608 36.546 28.823 24.116 22.081 43.171
2_5 24.116 43.171 36.393 39.661 24.929 30.278 26.916 45.172
2_6 30.278 45.172 36.546 30.996 24.116 38.868 43.171 37.925
2_7 38.868 37.925 39.661 31.238 30.278 34.67 45.172 39.79
2_8 34.67 39.79 30.996 29.889 38.868 25.25 37.925 29.869
2_9 25.25 29.869 31.238 -1.0 34.67 -1.0 39.79 -1.0
3_0 20.946 30.545 -1.0 23.331 -1.0 25.712 -1.0 31.95
3_1 25.712 31.95 23.164 16.95 20.946 27.151 30.545 34.775
3_2 27.151 34.775 23.331 22.081 25.712 32.608 31.95 39.862
3_3 32.608 39.862 16.95 26.916 27.151 36.393 34.775 34.346
3_4 36.393 34.346 22.081 43.171 32.608 36.546 39.862 45.491
3_5 36.546 45.491 26.916 45.172 36.393 39.661 34.346 32.237
3_6 39.661 32.237 43.171 37.925 36.546 30.996 45.491 37.678
3_7 30.996 37.678 45.172 39.79 39.661 31.238 32.237 53.734
3_8 31.238 53.734 37.925 29.869 30.996 29.889 37.678 55.279
3_9 29.889 55.279 39.79 -1.0 31.238 -1.0 53.734 -1.0
```

# 作業說明 - 紀錄完整的 Q-table (不單單紀錄起點的 Q-value)

```
4_0 23.164 30.817 -1.0 31.95 -1.0 23.331 -1.0 35.053
4_1 23.331 35.053 30.545 34.775 23.164 16.95 30.817 31.502
4_2 16.95 31.502 31.95 39.862 23.331 22.081 35.053 38.812
4_3 22.081 38.812 34.775 34.346 16.95 26.916 31.502 40.428
4_4 26.916 40.428 39.862 45.491 22.081 43.171 38.812 57.657
4_5 43.171 57.657 34.346 32.237 26.916 45.172 40.428 39.246
4_6 45.172 39.246 45.491 37.678 43.171 37.925 57.657 49.288
4_7 37.925 49.288 32.237 53.734 45.172 39.79 39.246 55.866
4_8 39.79 55.866 37.678 55.279 37.925 29.869 49.288 59.44
4_9 29.869 59.44 53.734 -1.0 39.79 -1.0 55.866 -1.0
5_0 30.545 38.184 -1.0 35.053 -1.0 31.95 -1.0 41.777
5_1 31.95 41.777 30.817 31.502 30.545 34.775 38.184 44.825
5_2 34.775 44.825 35.053 38.812 31.95 39.862 41.777 44.87
5_3 39.862 44.87 31.502 40.428 34.775 34.346 44.825 41.492
5_4 34.346 41.492 38.812 57.657 39.862 45.491 44.87 56.42
5_5 45.491 56.42 40.428 39.246 34.346 32.237 41.492 72.8
5_6 32.237 72.8 57.657 49.288 45.491 37.678 56.42 79.429
5_7 37.678 79.429 39.246 55.866 32.237 53.734 72.8 70.8
5_8 53.734 70.8 49.288 59.44 37.678 55.279 79.429 77.7
5_9 55.279 77.7 55.866 -1.0 53.734 -1.0 70.8 -1.0
6_0 30.817 33.594 -1.0 41.777 -1.0 35.053 -1.0 31.049
6_1 35.053 31.049 38.184 44.825 30.817 31.502 33.594 32.842
6_2 31.502 32.842 41.777 44.87 35.053 38.812 31.049 35.684
6_3 38.812 35.684 44.825 41.492 31.502 40.428 32.842 55.102
6_4 40.428 55.102 44.87 56.42 38.812 57.657 35.684 63.918
6_5 57.657 63.918 41.492 72.8 40.428 39.246 55.102 62.531
6_6 39.246 62.531 56.42 79.429 57.657 49.288 63.918 78.873
6_7 49.288 78.873 72.8 70.8 39.246 55.866 62.531 92.9
6_8 55.866 92.9 79.429 77.7 49.288 59.44 78.873 100.0
6_9 59.44 100.0 70.8 -1.0 55.866 -1.0 92.9 -1.0
7_0 38.184 -1.0 -1.0 31.049 -1.0 41.777 -1.0 -1.0
7_1 41.777 -1.0 33.594 32.842 38.184 44.825 -1.0 -1.0
7_2 44.825 -1.0 31.049 35.684 41.777 44.87 -1.0 -1.0
7_3 44.87 -1.0 32.842 55.102 44.825 41.492 -1.0 -1.0
7_4 41.492 -1.0 35.684 63.918 44.87 56.42 -1.0 -1.0
7_5 56.42 -1.0 55.102 62.531 41.492 72.8 -1.0 -1.0
7_6 72.8 -1.0 63.918 78.873 56.42 79.429 -1.0 -1.0
7_7 79.429 -1.0 62.531 92.9 72.8 70.8 -1.0 -1.0
7_8 70.8 -1.0 78.873 100.0 79.429 77.7 -1.0 -1.0
7_9 77.7 -1.0 92.9 -1.0 70.8 -1.0 -1.0 -1.0
```