NCU AI & ML HW3 RL

資管二 A 109403019 鄒翔宇

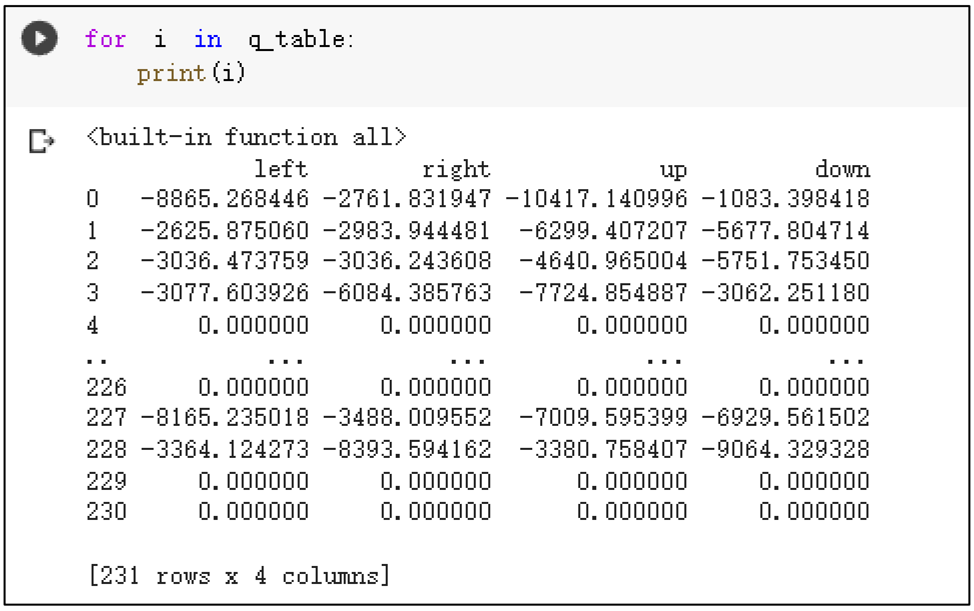
目錄

A. HW COLAB LINKS

HW3_109403019: click me

B. SCREENSHOTS

a. 訓練完的 Q-table

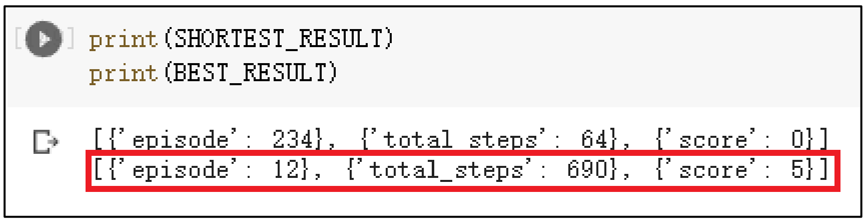


```
for i in q_table:
    print(i)
```

```
<built-in function all>
           left        right          up         down
0   -8865.268446 -2761.831947 -10417.140996 -1083.398418
1   -2625.875060 -2983.944481  -6299.407207 -5677.804714
2   -3036.473759 -3036.243608  -4640.965004 -5751.753450
3   -3077.603926 -6084.385763  -7724.854887 -3062.251180
4       0.000000     0.000000      0.000000     0.000000
..           ...          ...           ...          ...
226     0.000000     0.000000      0.000000     0.000000
227 -8165.235018 -3488.009552  -7009.595399 -6929.561502
228 -3364.124273 -8393.594162  -3380.758407 -9064.329328
229     0.000000     0.000000      0.000000     0.000000
230     0.000000     0.000000      0.000000     0.000000

[231 rows x 4 columns]
```

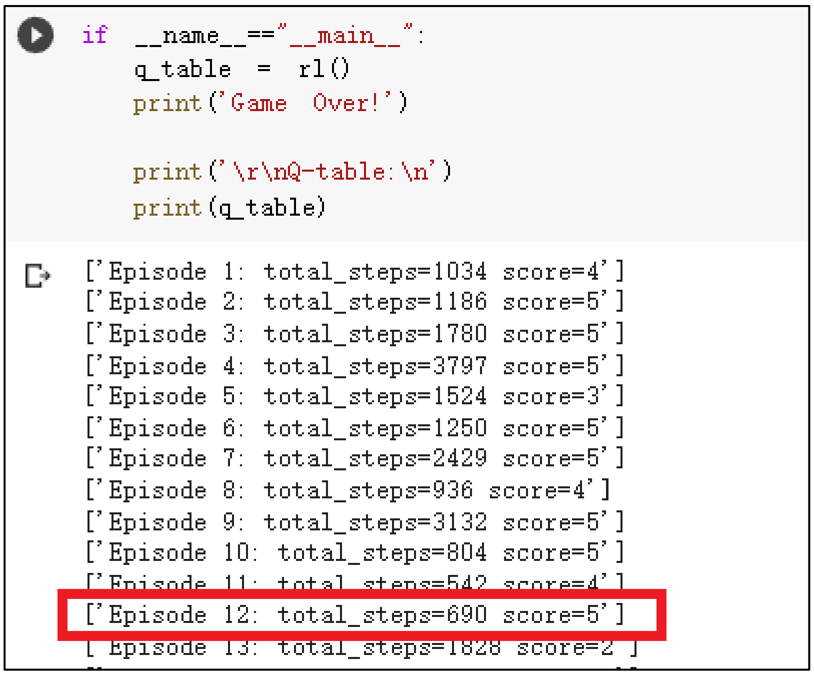
b. 最佳結果 (步數最少且寶藏最多)

total_steps: 690, score: 5



```
print(SHORTEST_RESULT)
print(BEST_RESULT)
```

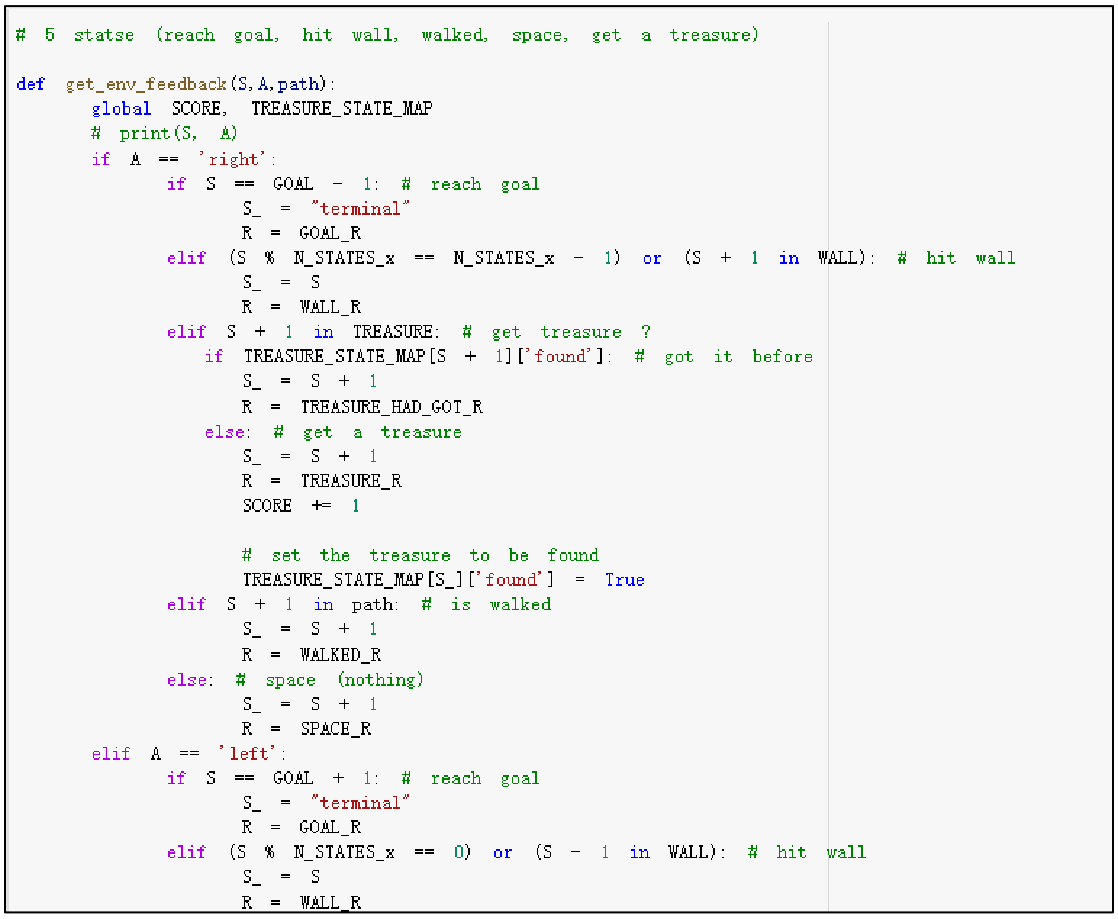
```
[{'episode': 234}, {'total_steps': 64}, {'score': 0}]
[{'episode': 12}, {'total_steps': 690}, {'score': 5}]
```



```
if __name__=="__main__":
    q_table = rl()
    print('Game Over!')

    print('\r\nQ-table:\n')
    print(q_table)
```

```
['Episode 1: total_steps=1034 score=4']
['Episode 2: total_steps=1186 score=5']
['Episode 3: total_steps=1780 score=5']
['Episode 4: total_steps=3797 score=5']
['Episode 5: total_steps=1524 score=3']
['Episode 6: total_steps=1250 score=5']
['Episode 7: total_steps=2429 score=5']
['Episode 8: total_steps=936 score=4']
['Episode 9: total_steps=3132 score=5']
['Episode 10: total_steps=804 score=5']
['Episode 11: total_steps=542 score=4']
['Episode 12: total_steps=690 score=5']
['Episode 13: total_steps=1828 score=2']
```

c. Reward 設定



```
# 5 statse (reach goal, hit wall, walked, space, get a treasure)

def get_env_feedback(S,A,path):
        global SCORE, TREASURE_STATE_MAP
        # print(S, A)
        if A == 'right':
                if S == GOAL - 1: # reach goal
                        S_ = "terminal"
                        R = GOAL_R
                elif (S % N_STATES_x == N_STATES_x - 1) or (S + 1 in WALL): # hit wall
                        S_ = S
                        R = WALL_R
                elif S + 1 in TREASURE: # get treasure ?
                    if TREASURE_STATE_MAP[S + 1]['found']: # got it before
                        S_ = S + 1
                        R = TREASURE_HAD_GOT_R
                    else: # get a treasure
                        S_ = S + 1
                        R = TREASURE_R
                        SCORE += 1

                        # set the treasure to be found
                        TREASURE_STATE_MAP[S_]['found'] = True
                elif S + 1 in path: # is walked
                        S_ = S + 1
                        R = WALKED_R
                else: # space (nothing)
                        S_ = S + 1
                        R = SPACE_R
        elif A == 'left':
                if S == GOAL + 1: # reach goal
                        S_ = "terminal"
                        R = GOAL_R
                elif (S % N_STATES_x == 0) or (S - 1 in WALL): # hit wall
                        S_ = S
                        R = WALL_R
```

```
            elif S - 1 in TREASURE:  # get treasure ?
                if TREASURE_STATE_MAP[S - 1]['found']:  # got it before
                    S_ = S - 1
                    R = TREASURE_HAD_GOT_R
                else:  # get a treasure
                    S_ = S - 1
                    R = TREASURE_R
                    SCORE += 1

                    # set the treasure to be found
                    TREASURE_STATE_MAP[S_]['found'] = True
            elif S - 1 in path:  # is walked
                    S_ = S - 1
                    R = WALKED_R
            else:  # space (nothing)
                    S_ = S - 1
                    R = SPACE_R
    elif A == 'up':
            if S == GOAL + 21:  # reach goal
                    S_ = "terminal"
                    R = GOAL_R
            elif (S < N_STATES_x) or (S - 21 in WALL):  # hit wall
                    S_ = S
                    R = WALL_R
            elif S - 21 in TREASURE:  # get treasure ?
                if TREASURE_STATE_MAP[S - 21]['found']:  # got it before
                    S_ = S - 21
                    R = TREASURE_HAD_GOT_R
                else:  # get a treasure
                    S_ = S - 21
                    R = TREASURE_R
                    SCORE += 1
```

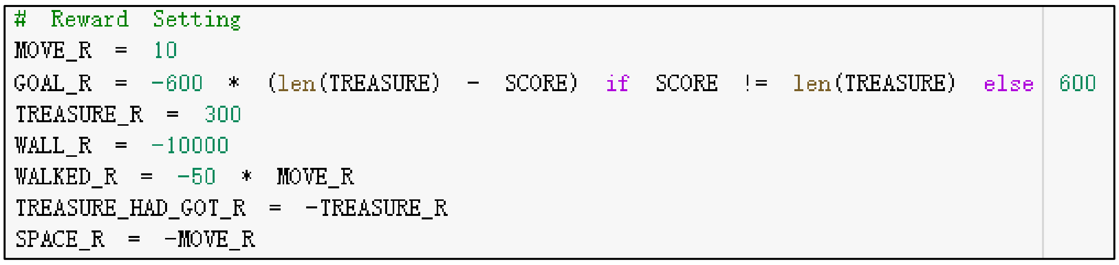
```
                    # set the treasure to be found
                    TREASURE_STATE_MAP[S_]['found'] = True
            elif S - 21 in path:  # is walked
                    S_ = S - 21
                    R = WALKED_R
            else:  # space (nothing)
                    S_ = S - 21
                    R = SPACE_R
    elif A == 'down':
            if S == GOAL - 21:  # reach goal
                    S_ = "terminal"
                    R = GOAL_R
            elif (S >= (N_STATES_y-1) * (N_STATES_x)) or (S + 21 in WALL):  # hit wall
                    S_ = S
                    R = WALL_R
            elif S + 21 in TREASURE:  # get treasure ?
                if TREASURE_STATE_MAP[S + 21]['found']:  # got it before
                    S_ = S + 21
                    R = TREASURE_HAD_GOT_R
                else:  # get a treasure
                    S_ = S + 21
                    R = TREASURE_R
                    SCORE += 1

                    # set the treasure to be found
                    TREASURE_STATE_MAP[S_]['found'] = True
            elif S + 21 in path:  # is walked
                    S_ = S + 21
                    R = WALKED_R
            else:  # space (nothing)
                    S_ = S + 21
                    R = SPACE_R

    return S_, R
```

d. Reward 設定說明

將上下左補齊，其中可能的狀況為 (1)抵達終點 (2)撞牆 (3)拿到寶藏，其中又分成是已拿過的寶藏或尚未拿過的寶藏 (4)走過的地方 (5)空格，甚麼都沒有。並依照 action 對於 state 值的改變去作計算以及判別條件。

雖然計算結果來作設定可能有點失去訓練的意義，可是我真的很想拿寶藏 (~~很想拿高分~~)，因此我去看地圖並計算了最小步數。score = 0 時最小步數為 60 步；而 score = 5 時最小步數為 99，多走步數為 39 (其中重複步數為 20)。
以每步 MOVE_R 為 10 作計算，重複步數 WALKED_R 設為 -50。

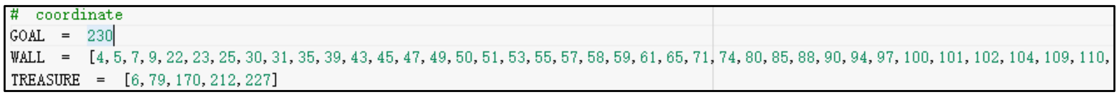
- **GOAL_R:**
  若不拿寶藏最少為 60 步，因此 GOAL_R 至少為 -MOVE_R * 60 = 600。有設定若寶藏沒有全拿會有 penalty。
- **TREASURE_R:**
  平均取得每個寶藏的多走步數約為 8, 重複步數為 4，因此獲得寶藏的 TREASURE_R 設為至少 MOVE_R * 8 - WALKED_R * 4 = 280。
- **TREASURE_HAD_GOT_R:**
  為避免重複繞遠路前往拿寶藏的地方，如果到達已拿過的寶藏位置，TREASURE_HAD_GOT_R 設為至少 -8 * MOVE_R(多走步數) + 4 * WALKED_R (重複步數) = -280 。基本上就是 - TREASURE_R。
- **WALL_R**
  完全不想撞到牆，所以撞到就扣爆。
- **SPACE_R:**
  踩空的話設為負值，避免一直走、繞遠路。



```
# Reward Setting
MOVE_R = 10
GOAL_R = -600 * (len(TREASURE) - SCORE) if SCORE != len(TREASURE) else 600
TREASURE_R = 300
WALL_R = -10000
WALKED_R = -50 * MOVE_R
TREASURE_HAD_GOT_R = -TREASURE_R
SPACE_R = -MOVE_R
```

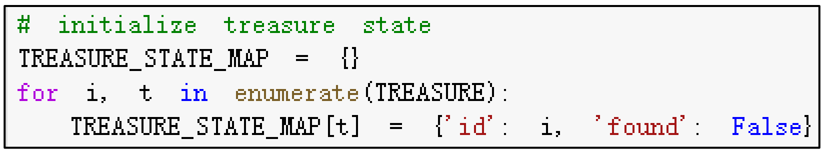
C. 心得

這次的作業是要利用 Q-learning 逃離迷宮，過去就一直想作迷宮類的程式，這次終於等到了。一開始先將地圖以及超參數等等設定好，迷宮的部分使用數值的方式計算，因為我自己覺得用陣列計算在程式碼設定上較為麻煩。



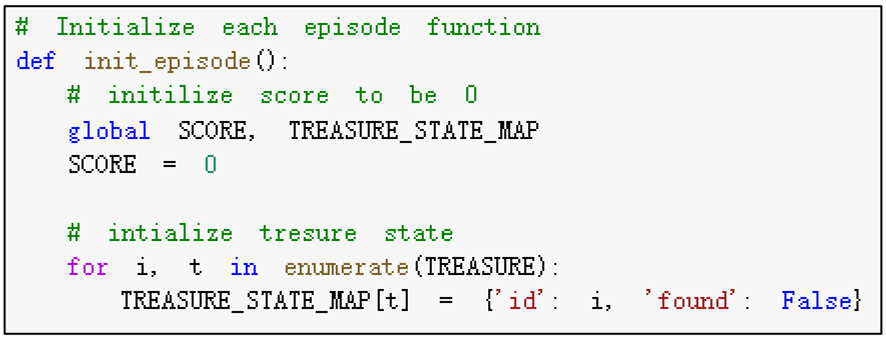
```
# coordinate
GOAL = 230
WALL = [4,5,7,9,22,23,25,30,31,35,39,43,45,47,49,50,51,53,55,57,58,59,61,65,71,74,80,85,88,90,94,97,100,101,102,104,109,110,
TREASURE = [6,79,170,212,227]
```

拿寶藏可以加分，但要注意寶藏是否已經拿過，因此使用 map 設定各寶藏的狀態。



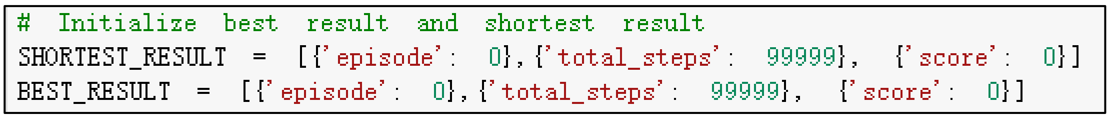
```
# initialize treasure state
TREASURE_STATE_MAP = {}
for i, t in enumerate(TREASURE):
    TREASURE_STATE_MAP[t] = {'id': i, 'found': False}
```

後面訓練本來想說為甚麼 score 拿到 5 以後就沒在更新過，發現原來是沒有初始化好每一輪，因此寫了一個初始化每一輪的 function，會將寶藏狀態與 score 重設。



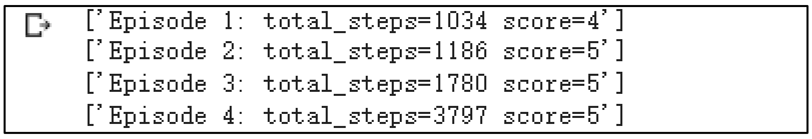
```
# Initialize each episode function
def init_episode():
    # initilize score to be 0
    global SCORE, TREASURE_STATE_MAP
    SCORE = 0

    # intialize tresure state
    for i, t in enumerate(TREASURE):
        TREASURE_STATE_MAP[t] = {'id': i, 'found': False}
```

因為跑的次數可能很多，要手動去找最佳結果勢必有些沒效率還可能找錯，因此宣告並初始化變數用來存放最佳結果。除了最多寶藏最少步數走出迷宮外，還加上最少步數走出迷宮的結果 (不考慮寶藏)。



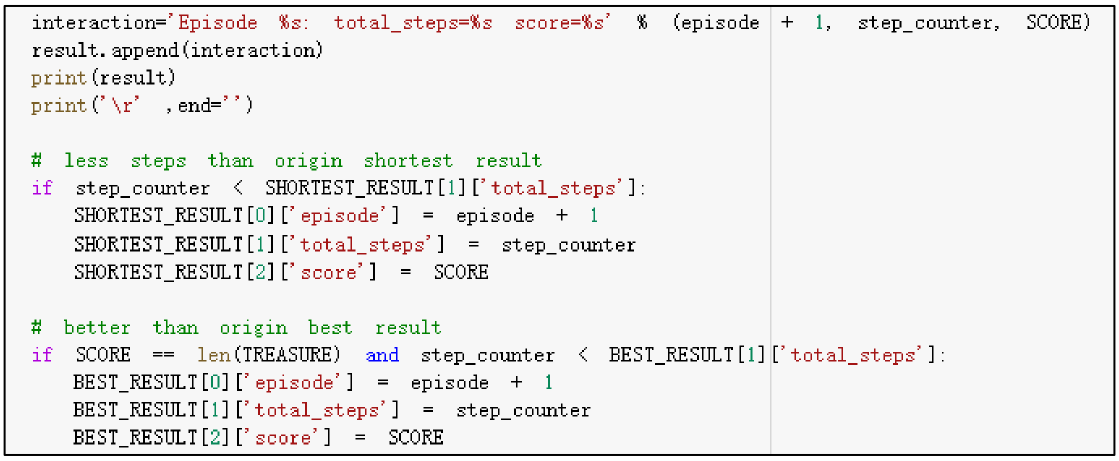
```
# Initialize best result and shortest result
SHORTEST_RESULT = [{'episode': 0},{'total_steps': 99999}, {'score': 0}]
BEST_RESULT = [{'episode': 0},{'total_steps': 99999}, {'score': 0}]
```

一開始訓練時跑了五六分鐘都沒有輸出結果，想說感覺不會這麼久，於是就在就將每一步的 (state, action) 印出來，發現怎麼一直卡在 44，看了一下圖，44 居然是在四周環牆的空格內。那肯定是 reward 設定上寫錯了。修正過後開始有每一輪的輸出結果。



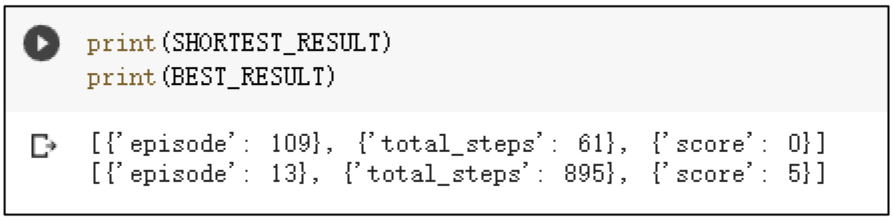
```
['Episode 1: total_steps=1034 score=4']
['Episode 2: total_steps=1186 score=5']
['Episode 3: total_steps=1780 score=5']
['Episode 4: total_steps=3797 score=5']
```

將要列印的輸出結果加上 score 欄位外，也加入判斷式去更新最佳結果。



```
interaction='Episode %s: total_steps=%s score=%s' % (episode + 1, step_counter, SCORE)
result.append(interaction)
print(result)
print('\r' ,end='')

# less steps than origin shortest result
if step_counter < SHORTEST_RESULT[1]['total_steps']:
    SHORTEST_RESULT[0]['episode'] = episode + 1
    SHORTEST_RESULT[1]['total_steps'] = step_counter
    SHORTEST_RESULT[2]['score'] = SCORE

# better than origin best result
if SCORE == len(TREASURE) and step_counter < BEST_RESULT[1]['total_steps']:
    BEST_RESULT[0]['episode'] = episode + 1
    BEST_RESULT[1]['total_steps'] = step_counter
    BEST_RESULT[2]['score'] = SCORE
```

有一次跑出 61 部就走完迷宮，比最短 60 還多一步而已。



```
print(SHORTEST_RESULT)
print(BEST_RESULT)
```

```
[{'episode': 109}, {'total_steps': 61}, {'score': 0}]
[{'episode': 13}, {'total_steps': 895}, {'score': 5}]
```

本來想說拿五個寶藏也花太多步了，比我算出最佳路徑的 99 步多將近十倍。還在煩惱如何設定使步數壓得更低，煩惱到睡著。但起床後，看到助教發的通知，只要步數小於 1000 就滿分，就帶著愉悅的心情迅速整理要繳交的程式碼以及完成心得！讚嘆助教！

本次作業的精隨大概就是 reward 的設定上了，好的 reward 會有更好的結果，一開始的 reward 都是隨便亂輸入的，只想說寶藏跟抵達終點的 reward 要很大。直到一次又一次的慘淡測試結果告訴我沒有這麼簡單，我才選擇去看地圖算步數來推出 reward 值的設定，並將撞牆、重複步數等 reward 調整一頓。輸出結果以及訓練過程相對我亂設的好很多，但我有兩個尚未獲得答案的疑惑。第一個是為甚麼同樣的程式碼同樣的超參數、reward 設定每次跑出來的結果都不盡相同，最佳結果可以差到好幾百步。第二個問題是為甚麼我的最佳解幾乎都在 15 回內下降完畢，接著就幾乎不會在拿到 5 個寶藏，一直在跑不拿寶藏的最短路徑，希望等到期末結束能找到這兩項問題的答案！

D. REFERENCES

1. What is Q Learning (Reinforcement Learning) - Morvan
2. A Beginners Guide to Q-Learning