Deep Reinforcement Learning
Homework #1

一、Random Policy Evaluation

1. Implementation

(1) Data format

- Matrix (to_state, from_state,方向)
  - 方向：0=上、1=左、2=下、3=右
  - Ex. Matrix(1,2,3)：從 state2 向左走到 state1

```
Matrix[1][2]
[0. 1. 0. 0.]
```

(2) Process_Initialization

- traverse 整個 grid world，得到 Matrix(16*16*4)

```python
# traverse all grid world
for row in range(4):
    for col in range(4):
        now_state = row * 4 + col
        ## 0 == move up
        Matrix[:, now_state, 0] = traverse_state(now_state, 0).flatten()
        ## 1 == move left
        Matrix[:, now_state, 1] = traverse_state(now_state, 1).flatten()
        ## 2 == move down
        Matrix[:, now_state, 2] = traverse_state(now_state, 2).flatten()
        ## 3 == move right
        Matrix[:, now_state, 3] = traverse_state(now_state, 3).flatten()
```

Example

```
Matrix[:,1,:]:
[[0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Matrix[:, 1, :] 的 Column 依序為上左下右

- 從 state1 往上走會撞到牆壁，所以[0,1]是 1
- 從 state1 往左走到 state0
- 從 state1 往下走到 state5
- 從 state1 往右走到 state2

```
Matrix[1,:,:]:
[[0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Matrix[1, :, :] 的 Column 依序為上左下右

- 從 state1 往上走會撞到牆壁，所以[0,1] 是 1
- 從 state5 往上走到 state1
- 從 state2 往左走到 state1
- 從 state0 往右走到 state1

- Function traverse_state 的設計細節

  依照 spec 的規定，如果 agent 撞到牆壁，會停留在原地到下個 timestamp，故先將 map 從 4*4 padding 成 6*6，所有值初始為 0，接著根據移動方向(上下左右)來更新 map 的值，最後更新至 Matrix。

  Example：如果方向是上(dir=1)

```python
if dir == 0:
    Map[Map_row-1, Map_col] = 1
elif dir == 1:
    Map[Map_row, Map_col-1] = 1
elif dir == 2:
    Map[Map_row+1, Map_col] = 1
elif dir == 3:
    Map[Map_row, Map_col+1] = 1
else:
    print(str(dir) + "is not a valid direction.")
```

```
** dir == 0 **
now_state: 1
Map:
[[0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]]
```

要特別處理撞到牆壁的情況，Example：

```python
## if the agent hits the wall
if np.max(Map[:,0]) > 0:
    idx = np.argmax(Map[:,0])
    Map[idx,1] += Map[idx,0]

if np.max(Map[:,5]) > 0:
    idx = np.argmax(Map[:,4+1])
    Map[idx,4] += Map[idx,4+1]

if np.max(Map[0,:]) > 0:
    idx = np.argmax(Map[0,:])
    Map[1,idx] += Map[0,idx]

if np.max(Map[5,:]) > 0:
    idx = np.argmax(Map[4+1,:])
    Map[4,idx] += Map[4+1,idx]
```

```
Map[:,0]: [0. 1. 0. 0. 0. 0.]
idx: 1
Map[idx,0]: 1.0
Before Map[idx,1]: 0.0
After Map[idx,1]: 1.0
```

另外，要將 Matrix 中 from_state 為 terminal state (state0 和 state15) 設為 1。

```python
## state 0 and 15 is terminal, set the value as 1
state0 = np.zeros([16,4])
state0[0,:] = 1
Matrix[:,0,:] = state0

state15 = np.zeros([16,4])
state15[15,:] = 1
Matrix[:,15,:] = state15
```

```
Matrix[:,state_0,:]
[[1. 1. 1. 1.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
Matrix[:,state_15,:]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [1. 1. 1. 1.]]
```

(3) Process_Iteration

- Iterative Policy Evaluation

按照 pseudo code 的步驟更新 value function，直到 converge

Iterative policy evaluation

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

```python
# iterate policy evaluation
while delta > theta:
    func_value_now = func_value.copy()
    for state in range(1,15):
        ## p(s',r|s,a)
        prob_next_state = prob * Matrix[:, state, :]
        ## r + gamma * V(s')
        future_reward = func_reward + gamma * func_value_now
        ## sum( prob_next_state * future_reward)
        func_value[state] = np.sum(np.matmul(np.transpose(prob_next_state), future_reward))
    delta = np.max(np.abs(func_value - func_value_now))
```

- 每跑一次 while loop 更新一次 state value，直到收斂

下圖以 gamma = 0.9 為例，印出 Iter1-5 和 Iter17-21 的 state value

```
Iter 1
[[ 0.   -0.75 -1.   -1.  ]
 [-0.75 -1.   -1.   -1.  ]
 [-1.   -1.   -1.   -0.75]
 [-1.   -1.   -0.75  0.  ]]
-----------------------------
Iter 2
[[ 0.   -1.37 -1.84 -1.9 ]
 [-1.37 -1.79 -1.9  -1.84]
 [-1.84 -1.9  -1.79 -1.37]
 [-1.9  -1.84 -1.37  0.  ]]
-----------------------------
Iter 3
[[ 0.   -1.88 -2.58 -2.68]
 [-1.88 -2.47 -2.63 -2.58]
 [-2.58 -2.63 -2.47 -1.88]
 [-2.68 -2.58 -1.88  0.  ]]
-----------------------------
```

```
Iter 19
[[ 0.   -4.53 -6.48 -7.02]
 [-4.53 -5.94 -6.54 -6.48]
 [-6.48 -6.54 -5.94 -4.53]
 [-7.02 -6.48 -4.53  0.  ]]
-----------------------------
Iter 20
[[ 0.   -4.56 -6.53 -7.08]
 [-4.56 -5.98 -6.59 -6.53]
 [-6.53 -6.59 -5.98 -4.56]
 [-7.08 -6.53 -4.56  0.  ]]
-----------------------------
Iter 21
[[ 0.   -4.59 -6.57 -7.12]
 [-4.59 -6.02 -6.63 -6.57]
 [-6.57 -6.63 -6.02 -4.59]
 [-7.12 -6.57 -4.59  0.  ]]
-----------------------------
```

2. Result

(1) Output the learned (converged) Vπ for state 1-14

- gamma = 0.9

```
106070038_hw1_1_data_gamma_0.9.txt
-4.59 -6.57 -7.12 -4.59 -6.02 -6.63 -6.57 -6.57 -6.63 -6.02 -4.59 -7.12 -6.57 -4.59
```

- gamma = 0.1

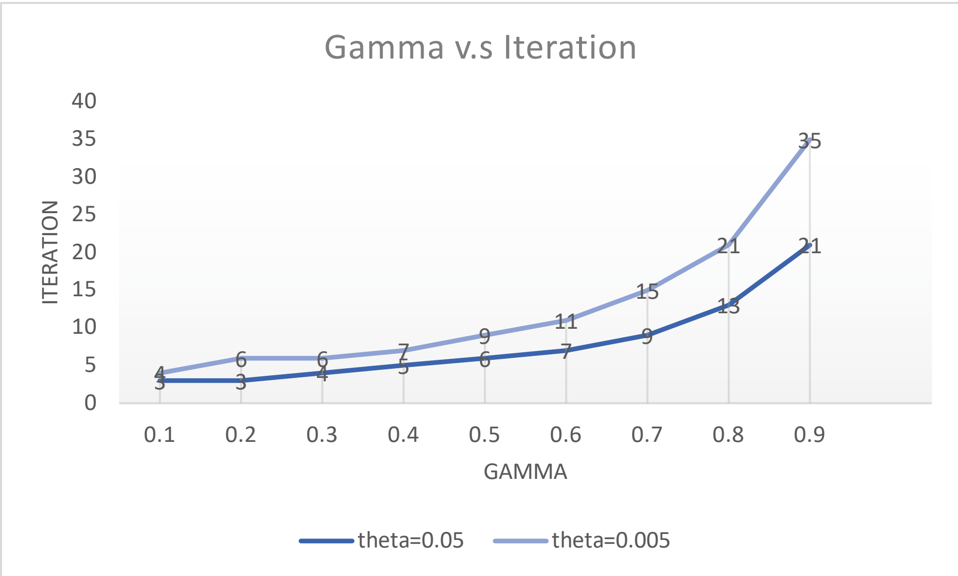
```
106070038_hw1_1_data_gamma_0.1.txt
-0.82 -1.10 -1.11 -0.82 -1.10 -1.11 -1.10 -1.10 -1.11 -1.10 -0.82 -1.11 -1.10 -0.82
```

3. How does the iteration stop?

當 delta 小於或等於 theta (設為 0.05) 的時候結束 while 迴圈

4. How does the discount factor gamma affect the results?

當 discount factor (gamma) 越大，需要越多個 iteration 才能 converge，實驗了兩個不同的 theta 值皆有同樣的趨勢(如下圖所示)。



二、Compare Q-Learning with SARSA

1. Environment design

(1) Environment：

		Small Bomb			Small Bomb			Small Bomb			
Start	Big Bomb										Goal

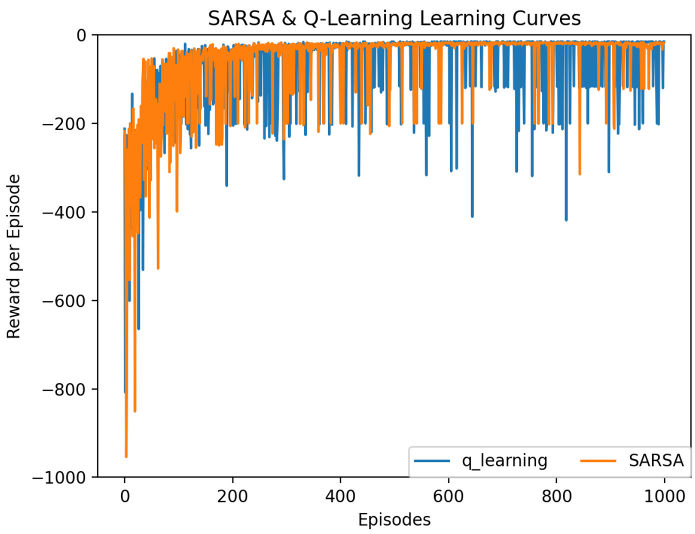
(2) Reward：

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-100	-100	-1	-100	-100	-1	-100	-100	-1	-1
Start	-200	-200	-200	-200	-200	-200	-200	-200	-200	-200	Goal

(3) Action：上下左右

2. Behaviors of these algorithms

(1) Learning Rate：嘗試 1000 個 episodes，下圖的計算方式為把當回合的所有 reward 加總，初期兩個方法皆有蠻多極端值，到中後期 SARSA 逐漸趨近於穩定，Q-Learning 仍有一些回合得到負值較大的回饋。此圖也反映 SARSA 較保守、考慮未來的 action、避免負值較大的回饋，Q-Learning 較敢於嘗試、尋找最佳路線，



(2) Q-Learning 路徑

註：* 為走過的路徑、X 是炸彈、0 是沒有經過但可以走的路徑、S 是起點、G 是終點

可觀察到 Q-Learning 在第 100 個 episode 仍在尋找最佳解，但到第 1000 個 episode 時，已經接近找到最 optimal 的路徑了。

```
Q-learning ep 96
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | * | 0 | * | * | * | * | * | * | * | * | * |
| * | 0 | X | X | * | X | X | * | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Q-learning ep 97
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | 0 | 0 |
| * | * | * | 0 | * | * | * | * | * | * | * | 0 |
| * | * | X | X | 0 | X | X | * | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Q-learning ep 98
-------------------------------------------------
| * | * | * | * | * | 0 | 0 | 0 | * | * | * | * |
| * | * | * | * | * | * | * | * | * | 0 | 0 | * |
| * | * | X | X | * | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Q-learning ep 99
-------------------------------------------------
| 0 | 0 | 0 | * | * | * | * | * | * | 0 | 0 | 0 |
| 0 | * | * | * | * | * | * | * | * | * | * | * |
| * | * | X | X | * | X | X | * | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Q-learning ep 100
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | * | X | X | 0 | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
```

```
Q-learning ep 995
-------------------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | * | * | * | * | * | * | * | * | * | * | 0 |
| * | * | X | X | 0 | X | X | 0 | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Q-learning ep 996
-------------------------------------------------
| 0 | * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | 0 | X | X | 0 | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Q-learning ep 997
-------------------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * | 0 | 0 |
| 0 | * | * | * | * | * | * | * | * | * | * | 0 |
| * | * | X | X | 0 | X | X | 0 | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Q-learning ep 998
-------------------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | 0 | X | X | 0 | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Q-learning ep 999
-------------------------------------------------
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | * | * | * | * | * | * | * | * | * | * | 0 |
| * | * | X | X | 0 | X | X | 0 | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
```

(3) SARSA 路徑：

註：＊為走過的路徑、X 是炸彈、0 是沒有經過但可以走的路徑、S 是起點、G 是終點

可觀察到 SARSA 在第 100 或到第 1000 個 episode 都盡量走距離炸彈最遠的路線，避免負值較大的 reward。另外，有嘗試跑 5000 個 episodes 結果還是類似，無法像 Q-Learning 找到最佳路線。

```
Sarsa ep 96
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | * | * | * | * | * | * | 0 | 0 | * | * | 0 |
| * | * | X | X | * | X | X | 0 | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Sarsa ep 97
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | 0 | * | * |
| * | * | * | 0 | 0 | * | * | * | * | * | * | * |
| * | * | X | X | 0 | X | X | 0 | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Sarsa ep 98
-------------------------------------------------
| 0 | * | * | * | * | * | * | * | * | * | * | * |
| * | * | 0 | * | * | * | * | * | * | * | * | * |
| * | * | X | X | 0 | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Sarsa ep 99
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | 0 | 0 |
| * | * | * | * | * | * | * | * | 0 | * | * | 0 |
| * | * | X | X | 0 | X | X | 0 | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Sarsa ep 100
-------------------------------------------------
| 0 | 0 | * | * | * | * | * | * | * | * | * | * |
| * | * | * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * | * |
| * | * | X | X | 0 | X | X | 0 | X | X | * | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
```

```
Sarsa ep 995
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * |
| * | 0 | X | X | 0 | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Sarsa ep 996
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * |
| * | 0 | X | X | 0 | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Sarsa ep 997
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * |
| * | 0 | X | X | 0 | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Sarsa ep 998
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * |
| * | 0 | X | X | 0 | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
Sarsa ep 999
-------------------------------------------------
| * | * | * | * | * | * | * | * | * | * | * | * |
| * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | * |
| * | 0 | X | X | 0 | X | X | 0 | X | X | 0 | * |
| S | X | X | X | X | X | X | X | X | X | X | G |
```

3. Implementation

```
# Q(S, A) ← Q(S, A) + α [R + γ max Q(S', A) − Q(S, A)]
if method == 'qlearning':
    update_value = alpha * (reward + gamma * np.max(action_value[next_state,:]) - now_value)
# Q(S, A) ← Q(S, A) + α [R + γ Q(S', A') − Q(S, A)]
elif method == 'sarsa':
    next_action = record[4]
    update_value = alpha * (reward + gamma * action_value[next_state, next_action] - now_value)
```

三、Tic-tac-toe (3x3)

1. Implementation: Q-learning training procedure

(1) 先定義 MDP 問題的各個元素

- Action: 下一步的位置
- States: 目前盤面狀態
- Reward:
  - 贏：R(s, a) = 1
  - 輸：R(s, a) = -1
  - 平手：R(s, a) = 0
- Goal: 最大化 Reward

(2) 可調整參數

- Learning rate (α): 更新值的程度
- Discount factor (γ):
- Exploration probability (ε): Explore(選擇 random action)和 Exploit(根據 Q(s,a)決定 action)的比例，一開始設為 1 會逐漸遞減，並設定下界。
- Iteration: 跑幾個 episode

(3) Q-learning 實作

對每個 episode 中的每一步

- $Q(S, A) \leftarrow Q(S, A) + \alpha\,[R + \gamma \max Q(S', A) - Q(S, A)]$

```
newQ = currentQ + self.learning_parameter * (self.agent.rewardFunction(state, action)
+ self.discount_factor * max_nextQ - currentQ)
```

2. Experiment

(1) Iteration：嘗試改變 Iteration 為 500,000、1,000,000，如下表淺藍色網底的實驗，1,000,000 個 Iter 和 500,000 個 Iter 相比， 500,000 個 Iter 的勝率較高。

註：表格中的數值為每 100 個 episode 中，# col win, # col lose, # tie

Iteration	500,000	1,000,000
500,000	37, 38, 26	0, 0, 0
1,000,000	**49**, 0, 51	36, 36, 28

(2) Learning rate (α) ：實驗 $\alpha = 0.1$、0.5，可觀察到在下表淺藍色網底的實驗中，learning rate 設為 0.1 較的勝率比 0.5 高一倍，故推測 learning rate 設為 0.1 較恰當。

註：表格中的數值為每 100 個 episode 中，# col win, # col lose, # tie

learning_rate	0.1	0.5
0.1	37, 38, 26	25, **50**, 25
0.5	0, 0, 100	35, 36, 29

(3) Discount factor (γ)：改變 γ 為 0.9、0.5，可觀察到在下表淺綠色網底的實驗中，當 $\gamma = 0.9$ 與 0.5 對戰時，0.9 的勝率比較高。

註：表格中的數值為每 100 個 episode 中，# col win, # col lose, # tie

discount factor	0.5	0.9
0.5	39, 38, 23	**25**, 0, 75
0.9	0, 1, 99	37, 38, 26

(4) Exploration probability (ε)：ε 的初始值皆為 1.0，每個 episode 依序遞減 1/Iter，直到設定的 ε_lower_bound，依序使用 ε_lower_bound＝0.2、0.5 做訓練，訓練好後相互對戰，觀察勝率變化。結果如下表，可觀察到相同 lower bound 的 agent 對戰勝率皆為 1:1，若不同 lower bound 的 agent 相比，ε lower_bound = 0.5 的勝率比較高。

註：表格中的數值為每 100 個 episode 中# col win, # col lose, # tie

ε lower_bound	0.2	0.5
0.2	37, 38, 25	**50,** 25, 25
0.5	0, 0, 100	41, 42, 17

以上實驗其他參數 default 為 Iteration = 500,000、α = 0.1、γ = 0.9、ε lower_bound = 0.2

3. Implementation: environment

(1) Class Agent：

```
symbol: 1
current_state: [[-1  0  0]
 [ 0  0  0]
 [ 0  0  0]]
actions: [[1 2]
 [2 0]]
action_history: [array([0, 0]), array([2, 1]), array([2, 2]), array([1, 1])]
```

- symbol(1 或-1，代表 agent 是 X 或 O)、current_state、actions(空的位置)、action_history(agent 當局下過的位置)
- Function: getPossibleActions、updatePossibleActions、performAction、performRandomAction、revertLastAction、getActionHash、getActionHashFromState、rewardFunction、assignState、getBestAction(選擇 reward 最高的)、saveTrainer、LoadTrainer
- performAction()：更新 Q(實作 Q learning，詳細寫在三 1.) → 執行 action → 把 action 記在 action_history → 更新空的位置(available_pos)

```python
def performAction(self, action, state = None, updateQ = False):
    if action.shape != (2,):
        print("Wrong shape " + str(action))

    if state == None:
        state = self.current_state

    # Read action
    x = action[0]
    y = action[1]

    # Update Q as part of Q-learning in the Trainer class
    if updateQ is True:
        self.trainer.updateQ(state, action)

    # Make move
    state.setPosition(x, y, self.symbol)
    self.action_history.append(action)

    # Update possible actions
    self.updatePossibleActions()
```

- getBestAction()：找到 expected return 最大的 action。

```
def getBestAction(self):
    self.updatePossibleActions()

    # Get hash key for state and actions
    state_hash, actions_hash = self.getActionHashFromState()

    # Return best move (if all are equally good, then it picks one at random)
    return self.trainer.getBestAction(state_hash, actions_hash, self.actions)
```

(2) Class Board：模擬圈圈叉叉 3x3 的棋盤

```
state:
[[-1  1 -1]
 [-1 -1  1]
 [ 1 -1  1]]
rows: 3
cols: 3
win_threshold: 3
```

- state(紀錄盤面)、rows=3、cols=3、win_threshold=3
- getState、getPosition、setPosition、getAvailablePos(尋找空的位置)、getStateHash、checkWinner(判斷是否有玩家獲勝)、checkGameEnded(判斷是否有空位置，若無則平手、遊戲結束)、checkWinPossible、resetGame、getInvertedState
- checkWinner()：判斷現在的盤面是否有贏家，依序檢查 row、col、對角線。

```
def checkWinner(self):
    symbols = np.unique(self.state)
    symbols = list(symbols[np.nonzero(symbols)])

    for sym in symbols:
        # Check rows
        row= np.any((np.all(self.state == sym, axis=1)))

        # Check columns
        col = np.any((np.all(self.state == sym, axis=0)))

        # Check diagonals
        diag1 = np.array([self.state[0,0], self.state[1,1], self.state[2,2]])
        diag1 = np.all(diag1 == sym)

        diag2 = np.array([self.state[0,2], self.state[1,1], self.state[2,0]])
        diag2 = np.all(diag2 == sym)

        # Check if state has winner and return winner in that case
        if row or col or diag1 or diag2:
            return sym

    return 0  # No winner found
```

(3) Class Trainer：

learning_parameter 和 discount_factor 參數的相關實驗在上頁，Q 是用 dictionary 的資料型別實作，儲存訓練過成中儲存的 Q function

- agent、learning_parameter、discount_factor、Q
- getStatePairKey、getValueQ、setValueQ、getMaxQ、getBestAction、updateQ
- getBestAction()：在所有可行的 action 中，找 Q 最大的。

```python
def getBestAction(self, state_hash, list_action_hash, list_actions):
    # Pick a random action at first
    random_idx = np.random.choice(list_actions.shape[0])
    best_action = list_actions[random_idx]

    # Find action that given largest Q in given state
    maxQ = 0
    for a_hash, action in zip(list_action_hash, list_actions):
        tmpQ = self.getValueQ(state_hash, a_hash)
        if maxQ < tmpQ:
            maxQ = tmpQ
            best_action = action

    return best_action
```

(4) function simulate()：

因為 Q-learning 訓練的方式為一部分的機率(exploration_probability)為 random、其他為使用從過去的經驗(Q function)找最佳解，exploration_probability 逐 episode 遞減 1/Iter，實作方式如下：

```python
#Explore
if explore_only is True or random.random() < exploration_probability:
    a.performRandomAction(updateQ=True)
# Exploit
else:
    best_action = a.getBestAction()
    a.performAction(best_action, updateQ=(True))

# Reduce probability to explore during training
# Do not remove completely
exploration_probability_lower_bound = 0.2
if exploration_probability > exploration_probability_lower_bound:
    exploration_probability -= 1/iterations
```

(5) training 時執行順序：simulate() → 初始化 Board 和兩個 agent → for loop{checkGameEnded() → updatePossibleActions → 根據 exploration_probability 選擇 explore-performRandomAction()或 exploit-getBestAction()-performAction() → 更新 exploration_probability → checkWinner() → 換另一個 agent}

4. Implement: Testing procedure

(1) 實作找 Q 最大、決定 action 的部分和訓練差不多，惟不用再更新 Q 值。

(2) 讀入 input 檔，用 for loop 依照行讀取，放入 board 內，用訓練好的 agent 尋找 Q 最大的 Action，最後寫入 output 檔。

```python
class Play():
    def __init__(self):
        trained_agent = "hw1_3_data"

        for line in input_file.readlines():
            input_state = line.rstrip("\n").split(" ")

            playerX = Board.playerX
            playerO = Board.playerO

            if len(input_state[1:10]) == 9:
                state = input_state[1:10]
                # print("state:"+str(state))
            else:
                print("Error State!")

            # Start game
            self.board = Board(rows=3, cols=3, win_threshold=3)
            index = 0
            for i in range(BOARD_ROWS):
                for j in range(BOARD_COLS):
                    if int(state[index]) == 1:
                        self.board.setPosition(i, j, 1)
                    elif int(state[index]) == -1:
                        self.board.setPosition(i, j, -1)
                    index+=1
            if input_state[0] == '1':
                self.current_player = playerX
            else:
                self.current_player = playerO
            # print("self.current_player: "+str(self.current_player))

            self.agent = Agent(self.current_player, self.board, load_trainer = trained_agent)
            self.agent_symbol = self.agent.symbol

            if self.current_player == self.agent_symbol:
                self.agentMove()

        input_file.close()
        output_file.close()
        # print("Finish!")

    def playMove(self, x, y):
        # print("Output move - x:"+str(x)+", y:"+str(y))
        print(str(x)+" "+str(y))
        output_file.write("%d " % x)
        output_file.write("%d\n" % y)

    def agentMove(self):
        move = self.agent.getBestAction()
        self.playMove(move[1], move[0])
```

四、Tic-tac-toe (4x4x4)

1. Implementation: MCTS training procedure

   Selection → Expansion → Simulation → Backpropagation

   (1) Selection：從 Root 開始，遞歸選擇最優的子節點，直到到達葉節點

   (2) Expansion：創建一個或者更多的字子節點

   (3) Simulation：運行一個模擬的輸出，直到博弈遊戲結束

   (4) Backpropagation：用模擬的結果輸出更新當前行動序列

```
(base) sd204175:~ nkust$ python 106070038_hw1_4_train.py
100%|████████████████████████████████████████| 50000/50000 [33:39<00:00, 24.76it/s]
695 704 1399
Win percentage: Agent 1 49.68%, Agent 2 50.32%.
Saved trainer of agent 2 to hw1_4_data
```

2. Implementation: 4x4x4 testing procedure

   類似上題 3x3 的實作方式，但要修改維度，讀入 input 檔，用 for loop 依照行讀取，放入 board 內，用訓練好的 agent 尋找 Q 最大的 Action，最後寫入 output 檔。

```python
class Play():
    def __init__(self):
        trained_agent = "hw1_4_data"

        for line in input_file.readlines():
            input_state = line.rstrip("\n").split(" ")

            playerX = Board.playerX
            playerO = Board.playerO

            if len(input_state[1:65]) == 64:
                state = input_state[1:65]
            else:
                print("Error State!")

            # Start game
            self.board = Board(rows=4, cols=4, heights=4, win_threshold=4)
            index = 0
            for i in range(BOARD_ROWS):
                for j in range(BOARD_COLS):
                    for k in range(BOARD_HEIGHTS):
                        if int(state[index]) == 1:
                            self.board.setPosition(i, j, k, 1)
                        elif int(state[index]) == -1:
                            self.board.setPosition(i, j, k, -1)
                        index+=1
```

```python
            if input_state[0] == '1':
                self.current_player = playerX
            else:
                self.current_player = playerO

            self.agent = Agent(self.current_player, self.board, load_trainer = trained_agent)
            self.agent_symbol = self.agent.symbol

            if self.current_player == self.agent_symbol:
                self.agentMove()

        input_file.close()
        output_file.close()
```

3. Implementation: Environment

   類似上題 3x3 的實作方式，但因為維度增加一維，所以在檢查是否有贏家時，先檢查 col、row、height，再考慮對角線是否有四點連線，對角線共有 24 種情況(8x3)。