HW 4-2

基本的網路架構與test model的設計都與4-1相同

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from Gridworld import Gridworld
from IPython.display import clear_output
import random
from matplotlib import pylab as plt

L1 = 64 #輸入層的寬度
L2 = 150 #第一隱藏層的寬度
L3 = 100 #第二隱藏層的寬度
L4 = 4 #輸出層的寬度

model = torch.nn.Sequential(
    torch.nn.Linear(L1, L2), #第一隱藏層的shape
    torch.nn.ReLU(),
    torch.nn.Linear(L2, L3), #第二隱藏層的shape
    torch.nn.ReLU(),
    torch.nn.Linear(L3,L4) #輸出層的shape
).to(device)
loss_fn = torch.nn.MSELoss() #指定損失函數為MSE（均方誤差）
learning_rate = 1e-3  #設定學習率
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate) #指定優化

gamma = 0.9 #折扣因子
epsilon = 1.0
```

```
def test_model(model, mode='static', display=True):
    i = 0
    test_game = Gridworld(size=4, mode=mode) #產生一場測試遊戲
    state_ = test_game.board.render_np().reshape(1,64) + np.random.rand(1,64
```

```
  state = torch.from_numpy(state_).float().to(device)
  if display:
    print("Initial State:")
    print(test_game.display())
  status = 1
  while(status == 1): #遊戲仍在進行
    qval = model(state)
    qval_ = qval.cpu().data.numpy()
    action_ = np.argmax(qval_)
    action = action_set[action_]
    if display:
      print('Move #: %s; Taking action: %s' % (i, action))
    test_game.makeMove(action)
    state_ = test_game.board.render_np().reshape(1,64) + np.random.rand(1,6
    state = torch.from_numpy(state_).float().to(device)
    if display:
      print(test_game.display())
    reward = test_game.reward()
    if reward != -1: #代表勝利（抵達終點）或落敗（掉入陷阱）
      if reward > 0: #reward>0，代表成功抵達終點
        status = 2 #將狀態設為2，跳出迴圈
        if display:
          print("Game won! Reward: %s" %reward)
        else: #掉入陷阱
          status = 0 #將狀態設為0，跳出迴圈
          if display:
            print("Game LOST. Reward: %s" %reward)
    i += 1 #每移動一步，i就加1
    if (i > 10): #若移動了10步，仍未取出勝利，則一樣視為落敗
      if display:
        print("Game lost; too many moves.")
      break
  win = True if status == 2 else False
  print(win)
  return win
```

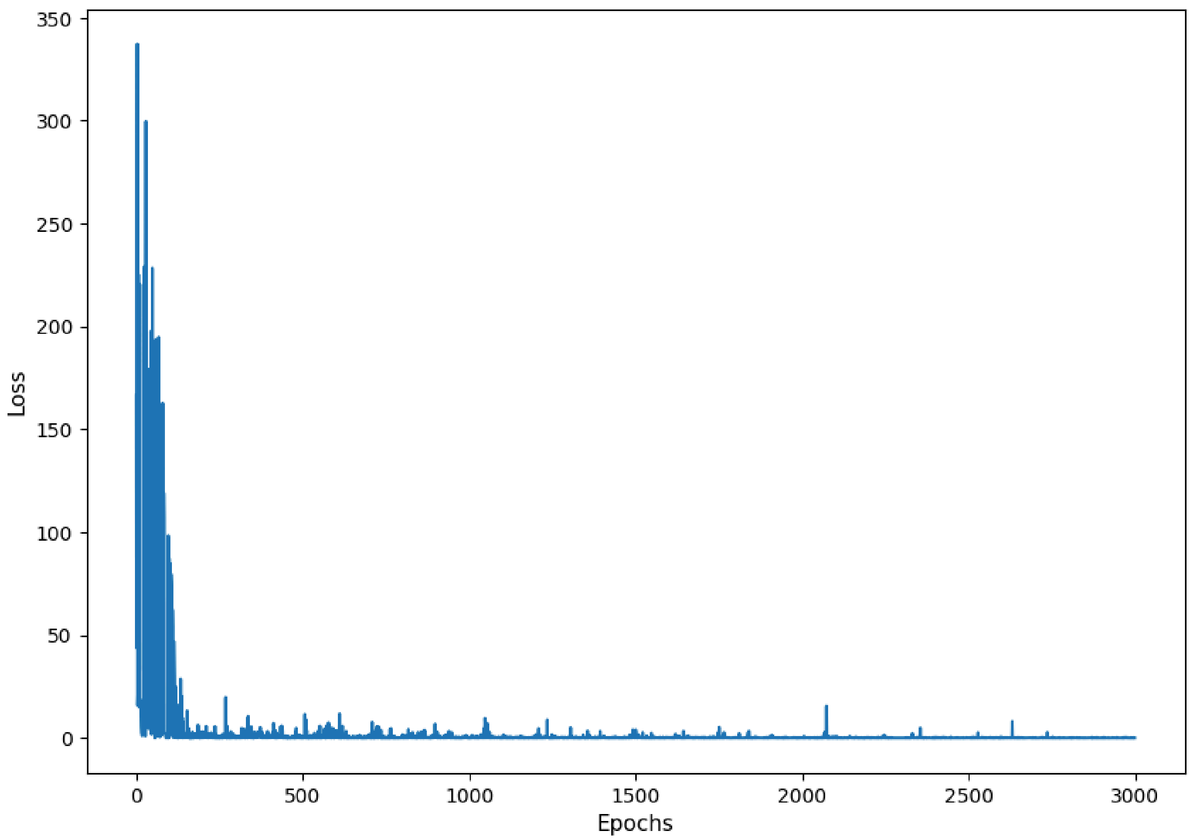
接下來的流程會是

- Basic DQN 分別使用static, player, random模式進行訓練，並且分別在3種模式下跑test model
- Double DQN 分別使用static, player, random模式進行訓練，並且分別在3種模式下跑test model
- Dueling DQN 分別使用static, player, random模式進行訓練，並且分別在3種模式下跑test model

Basic DQN for static mode

```
epochs = 3000
losses = [] #使用串列將每一次的loss記錄下來，方便之後將loss的變化趨勢畫成圖
for i in range(epochs):
  game = Gridworld(size=4, mode='static')
  state_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/10.0
  state1 = torch.from_numpy(state_).float().to(device) #將NumPy陣列轉換成PyT
  status = 1 #用來追蹤遊戲是否仍在繼續（『1』代表仍在繼續）
  while(status == 1):
    qval = model(state1) #執行Q網路，取得所有動作的預測Q值
    qval_ = qval.cpu().data.numpy() #將qval轉換成NumPy陣列
    if (random.random() < epsilon):
      action_ = np.random.randint(0,4) #隨機選擇一個動作（探索）
    else:
      action_ = np.argmax(qval_) #選擇Q值最大的動作（探索）
    action = action_set[action_] #將代表某動作的數字對應到makeMove()的英文字
    game.makeMove(action) #執行之前ε—貪婪策略所選出的動作
    state2_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/10
    state2 = torch.from_numpy(state2_).float().to(device) #動作執行完畢，取得遊
    reward = game.reward()
    with torch.no_grad():
      newQ = model(state2.reshape(1,64))
    maxQ = torch.max(newQ) #將新狀態下所輸出的Q值向量中的最大值給記錄下來
    if reward == -1:
      Y = reward + (gamma * maxQ) #計算訓練所用的目標Q值
    else: #若reward不等於-1，代表遊戲已經結束，也就沒有下一個狀態了，因此目標
      Y = reward
    Y = torch.tensor([Y], dtype=torch.float32).detach().to(device)
    X = qval.squeeze()[action_] #將演算法對執行的動作所預測的Q值存進X，並使
```

```
    loss = loss_fn(X, Y) #計算目標Q值與預測Q值之間的誤差
    if i%100 == 0:
      print(i, loss.item())
      clear_output(wait=True)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    state1 = state2
    if abs(reward) == 10:
      status = 0 # 若 reward 的絕對值為10，代表遊戲已經分出勝負，所以設status
  losses.append(loss.item())
  if epsilon > 0.1:
    epsilon -= (1/epochs) #讓ε的值隨著訓練的進行而慢慢下降，直到0.1（還是要保
plt.figure(figsize=(10,7))
plt.plot(losses)
plt.xlabel("Epochs",fontsize=11)
plt.ylabel("Loss",fontsize=11)
```



- test in static mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' 'P']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' 'P']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in player mode

```
Initial State:
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' 'P']]
Move #: 0; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' 'P' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' 'P' ' ' ' ']]
Move #: 2; Taking action: u
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

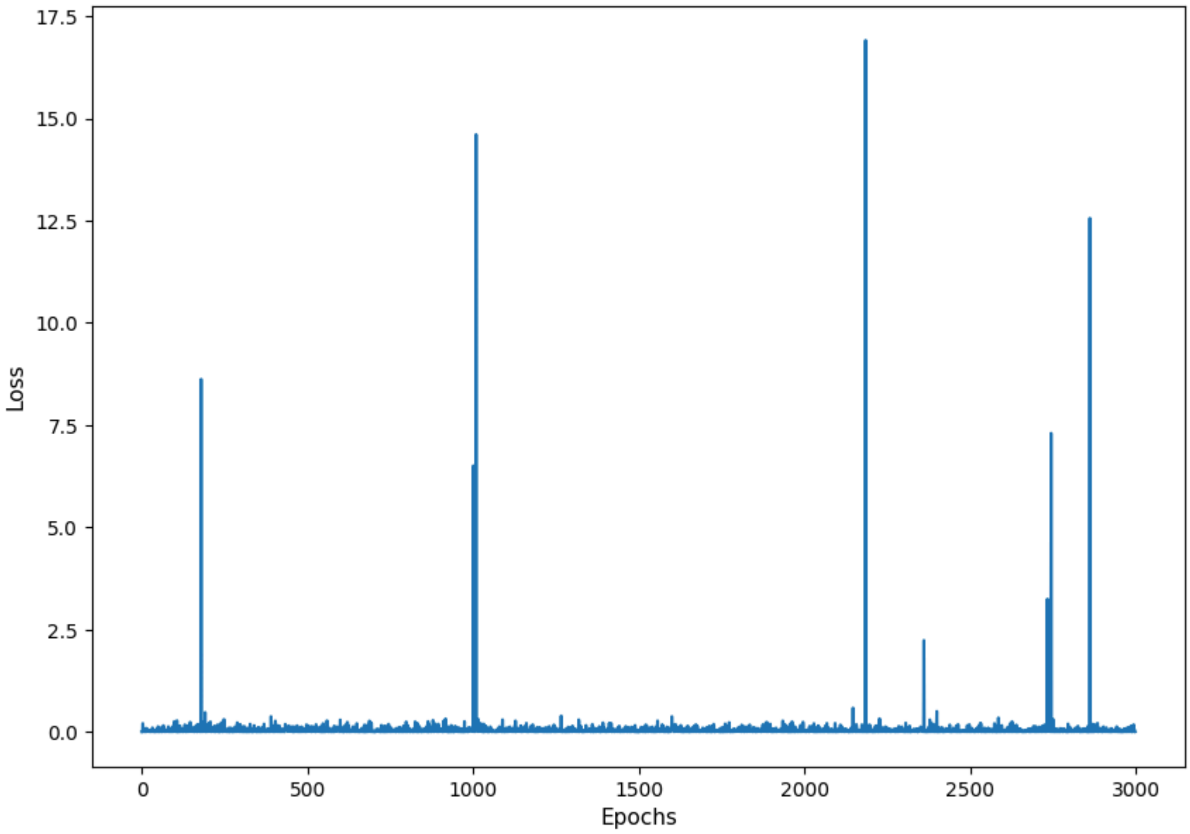
- test in random mode

```
Initial State:
[[' ' ' ' 'W' ' ']
 ['-' '+' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: l
[[' ' ' ' 'W' ' ']
 ['-' '+' ' ' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: u
[[' ' ' ' 'W' ' ']
 ['-' '+' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: u
[['P' ' ' 'W' ' ']
 ['-' '+' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: d
[[' ' ' ' 'W' ' ']
 ['-' '+' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 201
Win percentage: 20.1%
```

Basic mode for player mode

```
epochs = 3000
losses = [] #使用串列將每一次的loss記錄下來，方便之後將loss的變化趨勢畫成圖
for i in range(epochs):
  game = Gridworld(size=4, mode='player')
  state_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/10.0
  state1 = torch.from_numpy(state_).float().to(device) #將NumPy陣列轉換成PyT
  status = 1 #用來追蹤遊戲是否仍在繼續（『1』代表仍在繼續）
  while(status == 1):
    qval = model(state1) #執行Q網路，取得所有動作的預測Q值
    qval_ = qval.cpu().data.numpy() #將qval轉換成NumPy陣列
    if (random.random() < epsilon):
```

```
      action_ = np.random.randint(0,4) #隨機選擇一個動作（探索）
    else:
      action_ = np.argmax(qval_) #選擇Q值最大的動作（探索）
    action = action_set[action_] #將代表某動作的數字對應到makeMove()的英文字
    game.makeMove(action) #執行之前ε—貪婪策略所選出的動作
    state2_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/10
    state2 = torch.from_numpy(state2_).float().to(device) #動作執行完畢，取得遊
    reward = game.reward()
    with torch.no_grad():
      newQ = model(state2.reshape(1,64))
    maxQ = torch.max(newQ) #將新狀態下所輸出的Q值向量中的最大值給記錄下來
    if reward == -1:
      Y = reward + (gamma * maxQ) #計算訓練所用的目標Q值
    else: #若reward不等於-1，代表遊戲已經結束，也就沒有下一個狀態了，因此目標
      Y = reward
    Y = torch.tensor([Y], dtype=torch.float32).detach().to(device)
    X = qval.squeeze()[action_] #將演算法對執行的動作所預測的Q值存進X，並使
    loss = loss_fn(X, Y) #計算目標Q值與預測Q值之間的誤差
    if i%100 == 0:
      print(i, loss.item())
      clear_output(wait=True)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    state1 = state2
    if abs(reward) == 10:
      status = 0 # 若 reward 的絕對值為10，代表遊戲已經分出勝負，所以設status為
  losses.append(loss.item())
  if epsilon > 0.1:
    epsilon -= (1/epochs) #讓ε的值隨著訓練的進行而慢慢下降，直到0.1（還是要保
plt.figure(figsize=(10,7))
plt.plot(losses)
plt.xlabel("Epochs",fontsize=11)
plt.ylabel("Loss",fontsize=11)
```



- test in static mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' 'P']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' 'P']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in player mode

```
Initial State:
[['+' '-' ' ' ' ']
 [' ' 'W' 'P' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: u
[['+' '-' ' ' ' ']
 ['P' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

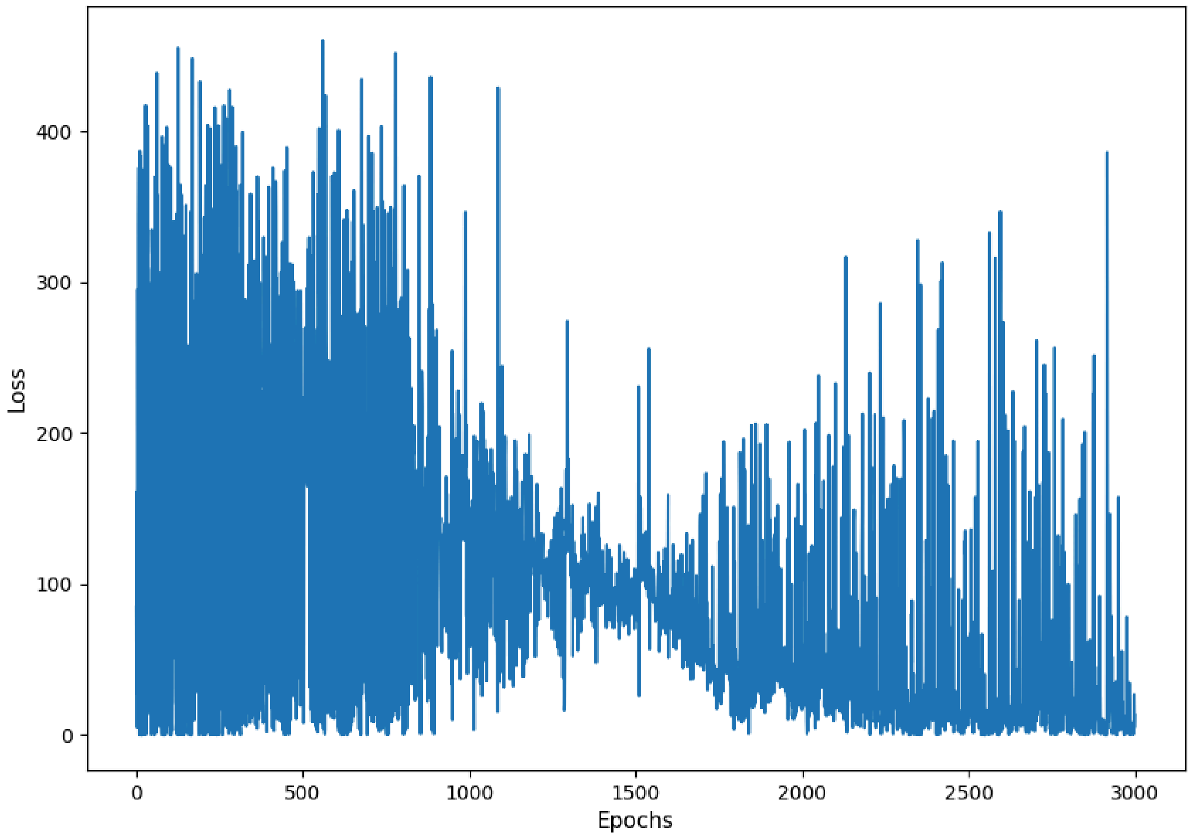
- test in random mode

```
Initial State:
[[' ' ' ' 'P' ' ']
 [' ' '-' 'W' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' '+']]
Move #: 0; Taking action: d
[[' ' ' ' 'P' ' ']
 [' ' '-' 'W' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' '+']]
Move #: 1; Taking action: d
[[' ' ' ' 'P' ' ']
 [' ' '-' 'W' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' '+']]
Move #: 2; Taking action: d
[[' ' ' ' 'P' ' ']
 [' ' '-' 'W' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' '+']]
Move #: 3; Taking action: d
[[' ' ' ' 'P' ' ']
 [' ' '-' 'W' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' '+']]
...
Game lost; too many moves.
False
Games played: 1000, # of wins: 226
Win percentage: 22.6%
```

Basic DQN for random mode

```
epochs = 3000
losses = [] #使用串列將每一次的loss記錄下來，方便之後將loss的變化趨勢畫成圖
for i in range(epochs):
  game = Gridworld(size=4, mode='random')
  state_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/10.0
  state1 = torch.from_numpy(state_).float().to(device) #將NumPy陣列轉換成PyT
  status = 1 #用來追蹤遊戲是否仍在繼續（『1』代表仍在繼續）
  while(status == 1):
    qval = model(state1) #執行Q網路，取得所有動作的預測Q值
    qval_ = qval.cpu().data.numpy() #將qval轉換成NumPy陣列
    if (random.random() < epsilon):
```

```
      action_ = np.random.randint(0,4) #隨機選擇一個動作（探索）
    else:
      action_ = np.argmax(qval_) #選擇Q值最大的動作（探索）
    action = action_set[action_] #將代表某動作的數字對應到makeMove()的英文字
    game.makeMove(action) #執行之前ε—貪婪策略所選出的動作
    state2_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/10
    state2 = torch.from_numpy(state2_).float().to(device) #動作執行完畢，取得遊
    reward = game.reward()
    with torch.no_grad():
      newQ = model(state2.reshape(1,64))
    maxQ = torch.max(newQ) #將新狀態下所輸出的Q值向量中的最大值給記錄下來
    if reward == -1:
      Y = reward + (gamma * maxQ)  #計算訓練所用的目標Q值
    else: #若reward不等於-1，代表遊戲已經結束，也就沒有下一個狀態了，因此目標
      Y = reward
    Y = torch.tensor([Y], dtype=torch.float32).detach().to(device)
    X = qval.squeeze()[action_] #將演算法對執行的動作所預測的Q值存進X，並使用
    loss = loss_fn(X, Y) #計算目標Q值與預測Q值之間的誤差
    if i%100 == 0:
      print(i, loss.item())
      clear_output(wait=True)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    state1 = state2
    if abs(reward) == 10:
      status = 0 # 若 reward 的絕對值為10，代表遊戲已經分出勝負，所以設status為
  losses.append(loss.item())
  if epsilon > 0.1:
    epsilon -= (1/epochs) #讓ε的值隨著訓練的進行而慢慢下降，直到0.1（還是要保
plt.figure(figsize=(10,7))
plt.plot(losses)
plt.xlabel("Epochs",fontsize=11)
plt.ylabel("Loss",fontsize=11)
```



- test in static mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: l
[['+' '-' 'P' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Game won! Reward: 10
True
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in player mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: l
[['+' '-' 'P' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Game won! Reward: 10
True
Initial State:
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 836
Win percentage: 83.6%
```

- test in random mode

```
Initial State:
[[' ' ' ' ' ' '+']
 [' ' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']]
Move #: 0; Taking action: r
[[' ' ' ' ' ' '+']
 [' ' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']]
Move #: 1; Taking action: u
[[' ' ' ' ' ' '+']
 [' ' '-' ' ' ' ']
 [' ' 'W' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: u
[[' ' ' ' ' ' '+']
 [' ' '-' 'P' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: u
[[' ' ' ' 'P' '+']
 [' ' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 912
Win percentage: 91.2%
```

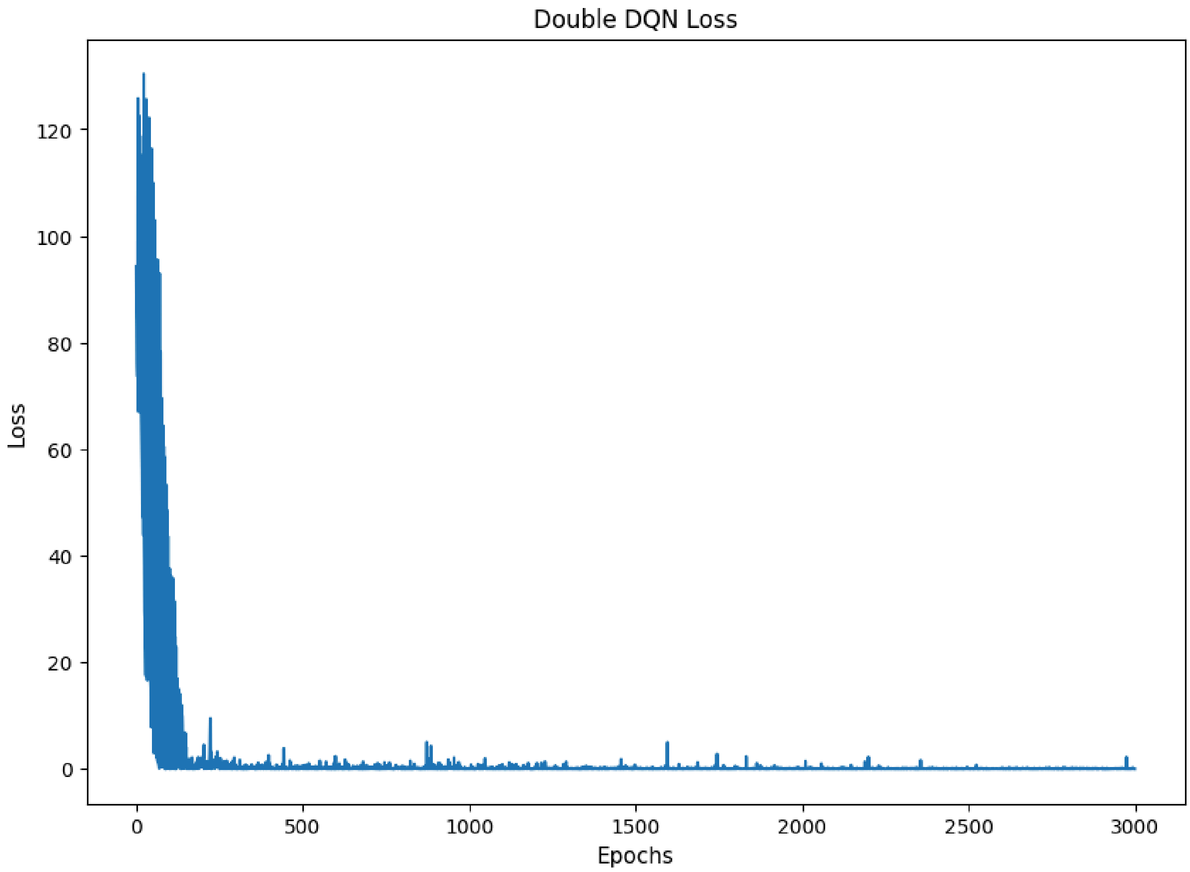
Double DQN for static mode

```
# 定義模型架構
class QNet(nn.Module):
    def __init__(self):
        super(QNet, self).__init__()
        self.fc1 = nn.Linear(64, 128)
        self.fc2 = nn.Linear(128, 4)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

```
# 初始化主網路與目標網路
model = QNet().to(device)
target_model = QNet().to(device)
target_model.load_state_dict(model.state_dict())  # 初始與 model 同權重
target_model.eval()  # 推論用，不需要計算梯度

loss_fn = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
gamma = 0.9
epsilon = 1.0

epochs = 3000
losses = []

# 每隔多少回合同步一次 target_model
sync_interval = 20

for i in range(epochs):
    game = Gridworld(size=4, mode='static')
    state_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/10.0
    state1 = torch.from_numpy(state_).float().to(device)
    status = 1

    while status == 1:
        qval = model(state1)
        qval_ = qval.cpu().data.numpy()

        if random.random() < epsilon:
            action_ = np.random.randint(0, 4)
        else:
            action_ = np.argmax(qval_)

        action = action_set[action_]
        game.makeMove(action)
        state2_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/
        state2 = torch.from_numpy(state2_).float().to(device)
        reward = game.reward()
```

```
        with torch.no_grad():
            # Double DQN 核心邏輯：
            # 主網路選動作
            next_qvals = model(state2)
            next_action = torch.argmax(next_qvals).item()
            # 目標網路估算該動作的 Q 值
            target_qvals = target_model(state2)
            maxQ = target_qvals[0][next_action]

        if reward == -1:
            Y = reward + gamma * maxQ
        else:
            Y = reward

        Y = torch.tensor([Y], dtype=torch.float32).to(device)
        X = qval[0][action_]
        loss = loss_fn(X, Y)

        if i % 100 == 0:
            print(f"Epoch {i}, Loss: {loss.item():.4f}")
            clear_output(wait=True)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        state1 = state2
        if abs(reward) == 10:
            status = 0

    losses.append(loss.item())

    # 更新 ε
    if epsilon > 0.1:
        epsilon -= (1/epochs)

    # 每隔 sync_interval 回合，同步一次 target_model
    if i % sync_interval == 0:
```

```
    target_model.load_state_dict(model.state_dict())

# 畫出 Loss 曲線
plt.figure(figsize=(10, 7))
plt.plot(losses)
plt.xlabel("Epochs", fontsize=11)
plt.ylabel("Loss", fontsize=11)
plt.title("Double DQN Loss")
plt.show()
```



- test in static mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' 'P']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' 'P']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in player mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' 'P']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' 'P']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in random mode

```
Initial State:
[[' ' ' ' ' ' 'W']
 [' ' ' ' ' ' ' ']
 [' ' 'P' ' ' '+']
 [' ' ' ' ' ' '-']]
Move #: 0; Taking action: l
[[' ' ' ' ' ' 'W']
 [' ' ' ' ' ' ' ']
 ['P' ' ' ' ' '+']
 [' ' ' ' ' ' '-']]
Move #: 1; Taking action: u
[[' ' ' ' ' ' 'W']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' '+']
 [' ' ' ' ' ' '-']]
Move #: 2; Taking action: u
[['P' ' ' ' ' 'W']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' '+']
 [' ' ' ' ' ' '-']]
Move #: 3; Taking action: d
[[' ' ' ' ' ' 'W']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' '+']
 [' ' ' ' ' ' '-']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 237
Win percentage: 23.7%
```

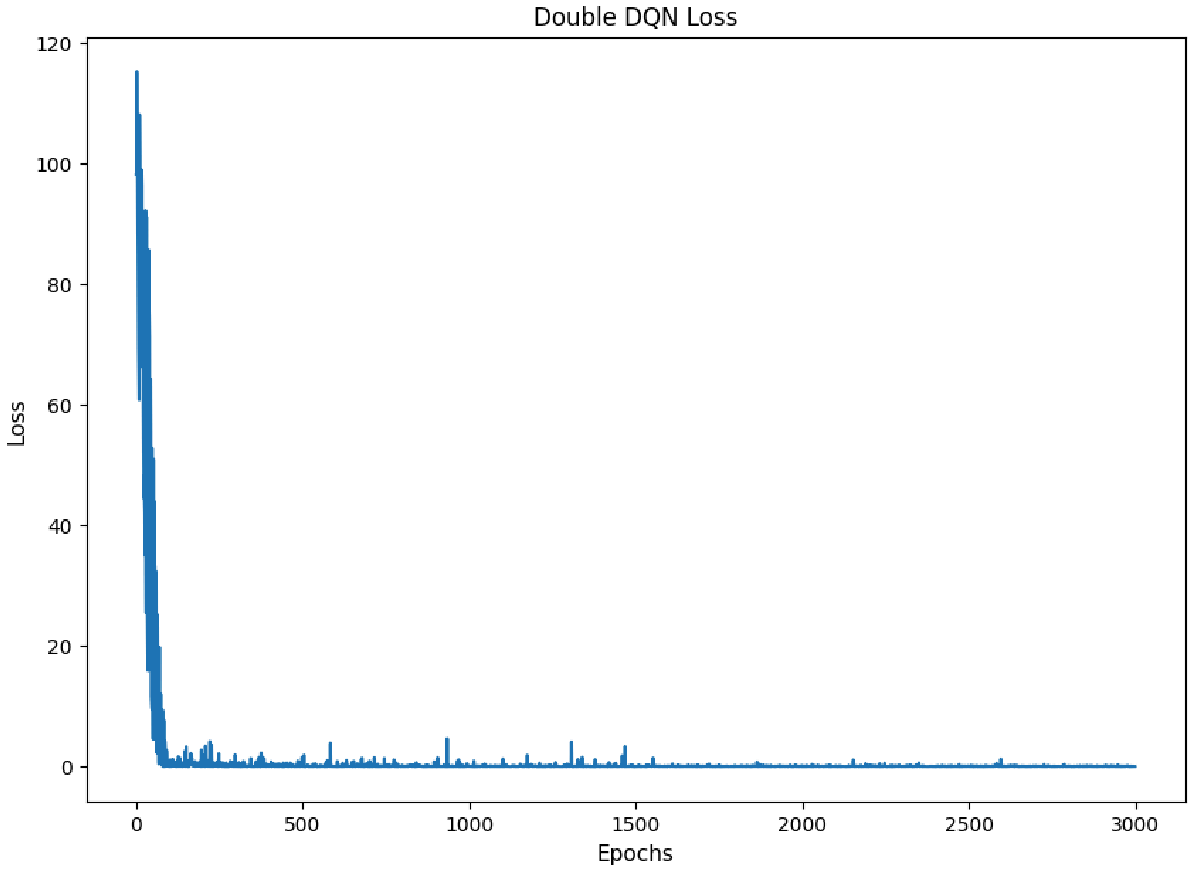
Double DQN for player mode

```
# 定義模型架構
class QNet(nn.Module):
    def __init__(self):
        super(QNet, self).__init__()
        self.fc1 = nn.Linear(64, 128)
        self.fc2 = nn.Linear(128, 4)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

```
# 初始化主網路與目標網路
model = QNet().to(device)
target_model = QNet().to(device)
target_model.load_state_dict(model.state_dict())  # 初始與 model 同權重
target_model.eval()  # 推論用，不需要計算梯度

loss_fn = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
gamma = 0.9
epsilon = 1.0

epochs = 3000
losses = []

# 每隔多少回合同步一次 target_model
sync_interval = 20

for i in range(epochs):
    game = Gridworld(size=4, mode='player')
    state_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/10.0
    state1 = torch.from_numpy(state_).float().to(device)
    status = 1

    while status == 1:
        qval = model(state1)
        qval_ = qval.cpu().data.numpy()

        if random.random() < epsilon:
            action_ = np.random.randint(0, 4)
        else:
            action_ = np.argmax(qval_)

        action = action_set[action_]
        game.makeMove(action)
        state2_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/
        state2 = torch.from_numpy(state2_).float().to(device)
        reward = game.reward()
```

```
        with torch.no_grad():
            # Double DQN 核心邏輯：
            # 主網路選動作
            next_qvals = model(state2)
            next_action = torch.argmax(next_qvals).item()
            # 目標網路估算該動作的 Q 值
            target_qvals = target_model(state2)
            maxQ = target_qvals[0][next_action]

        if reward == -1:
            Y = reward + gamma * maxQ
        else:
            Y = reward

        Y = torch.tensor([Y], dtype=torch.float32).to(device)
        X = qval[0][action_]
        loss = loss_fn(X, Y)

        if i % 100 == 0:
            print(f"Epoch {i}, Loss: {loss.item():.4f}")
            clear_output(wait=True)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        state1 = state2
        if abs(reward) == 10:
            status = 0

    losses.append(loss.item())

    # 更新 ε
    if epsilon > 0.1:
        epsilon -= (1/epochs)

    # 每隔 sync_interval 回合，同步一次 target_model
    if i % sync_interval == 0:
```

```
    target_model.load_state_dict(model.state_dict())

# 畫出 Loss 曲線
plt.figure(figsize=(10, 7))
plt.plot(losses)
plt.xlabel("Epochs", fontsize=11)
plt.ylabel("Loss", fontsize=11)
plt.title("Double DQN Loss")
plt.show()
```



- test in static mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' 'P']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' 'P' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in player mode

```
Initial State:
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 ['P' ' ' ' ' ' ']]
Move #: 0; Taking action: u
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: u
[['+' '-' ' ' ' ']
 ['P' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: u
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Game won! Reward: 10
True
Initial State:
[['+' '-' ' ' ' ']
 [' ' 'W' 'P' ' ']
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in random mode

```
Initial State:
[[' ' ' ' ' ' '-']
 [' ' ' ' '+' ' ']
 ['W' ' ' ' ' ' ']
 [' ' 'P' ' ' ' ']]
Move #: 0; Taking action: l
[[' ' ' ' ' ' '-']
 [' ' ' ' '+' ' ']
 ['W' ' ' ' ' ' ']
 ['P' ' ' ' ' ' ']]
Move #: 1; Taking action: u
[[' ' ' ' ' ' '-']
 [' ' ' ' '+' ' ']
 ['W' ' ' ' ' ' ']
 ['P' ' ' ' ' ' ']]
Move #: 2; Taking action: u
[[' ' ' ' ' ' '-']
 [' ' ' ' '+' ' ']
 ['W' ' ' ' ' ' ']
 ['P' ' ' ' ' ' ']]
Move #: 3; Taking action: u
[[' ' ' ' ' ' '-']
 [' ' ' ' '+' ' ']
 ['W' ' ' ' ' ' ']
 ['P' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 204
Win percentage: 20.4%
```

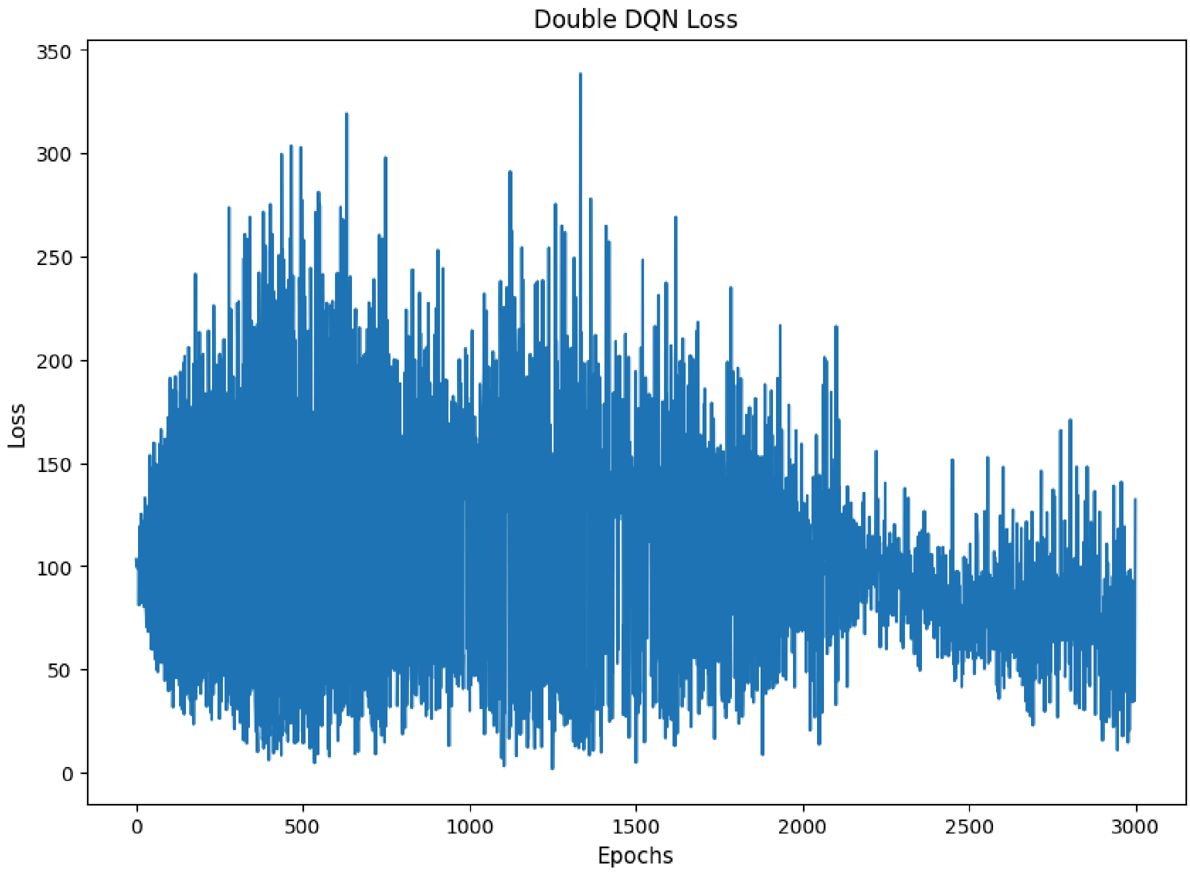
Double DQN for random mode

```
# 定義模型架構
class QNet(nn.Module):
    def __init__(self):
        super(QNet, self).__init__()
        self.fc1 = nn.Linear(64, 128)
        self.fc2 = nn.Linear(128, 4)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

```
# 初始化主網路與目標網路
model = QNet().to(device)
target_model = QNet().to(device)
target_model.load_state_dict(model.state_dict())  # 初始與 model 同權重
target_model.eval()  # 推論用，不需要計算梯度

loss_fn = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
gamma = 0.9
epsilon = 1.0

epochs = 3000
losses = []

# 每隔多少回合同步一次 target_model
sync_interval = 20

for i in range(epochs):
    game = Gridworld(size=4, mode='random')
    state_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64)/10.(
    state1 = torch.from_numpy(state_).float().to(device)
    status = 1

    while status == 1:
        qval = model(state1)
        qval_ = qval.cpu().data.numpy()

        if random.random() < epsilon:
            action_ = np.random.randint(0, 4)
        else:
            action_ = np.argmax(qval_)

        action = action_set[action_]
        game.makeMove(action)
        state2_ = game.board.render_np().reshape(1,64) + np.random.rand(1,64),
        state2 = torch.from_numpy(state2_).float().to(device)
        reward = game.reward()
```

```
        with torch.no_grad():
            # Double DQN 核心邏輯：
            # 主網路選動作
            next_qvals = model(state2)
            next_action = torch.argmax(next_qvals).item()
            # 目標網路估算該動作的 Q 值
            target_qvals = target_model(state2)
            maxQ = target_qvals[0][next_action]

        if reward == -1:
            Y = reward + gamma * maxQ
        else:
            Y = reward

        Y = torch.tensor([Y], dtype=torch.float32).to(device)
        X = qval[0][action_]
        loss = loss_fn(X, Y)

        if i % 100 == 0:
            print(f"Epoch {i}, Loss: {loss.item():.4f}")
            clear_output(wait=True)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        state1 = state2
        if abs(reward) == 10:
            status = 0

    losses.append(loss.item())

    # 更新 ε
    if epsilon > 0.1:
        epsilon -= (1/epochs)

    # 每隔 sync_interval 回合，同步一次 target_model
    if i % sync_interval == 0:
```

```
        target_model.load_state_dict(model.state_dict())

# 畫出 Loss 曲線
plt.figure(figsize=(10, 7))
plt.plot(losses)
plt.xlabel("Epochs", fontsize=11)
plt.ylabel("Loss", fontsize=11)
plt.title("Double DQN Loss")
plt.show()
```



- test in static mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: u
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: u
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: u
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: u
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 548
Win percentage: 54.800000000000004%
```

- test in player mode

```
Initial State:
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' 'P' ' ' ' ']]
Move #: 0; Taking action: u
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: u
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: u
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: u
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 695
Win percentage: 69.5%
```

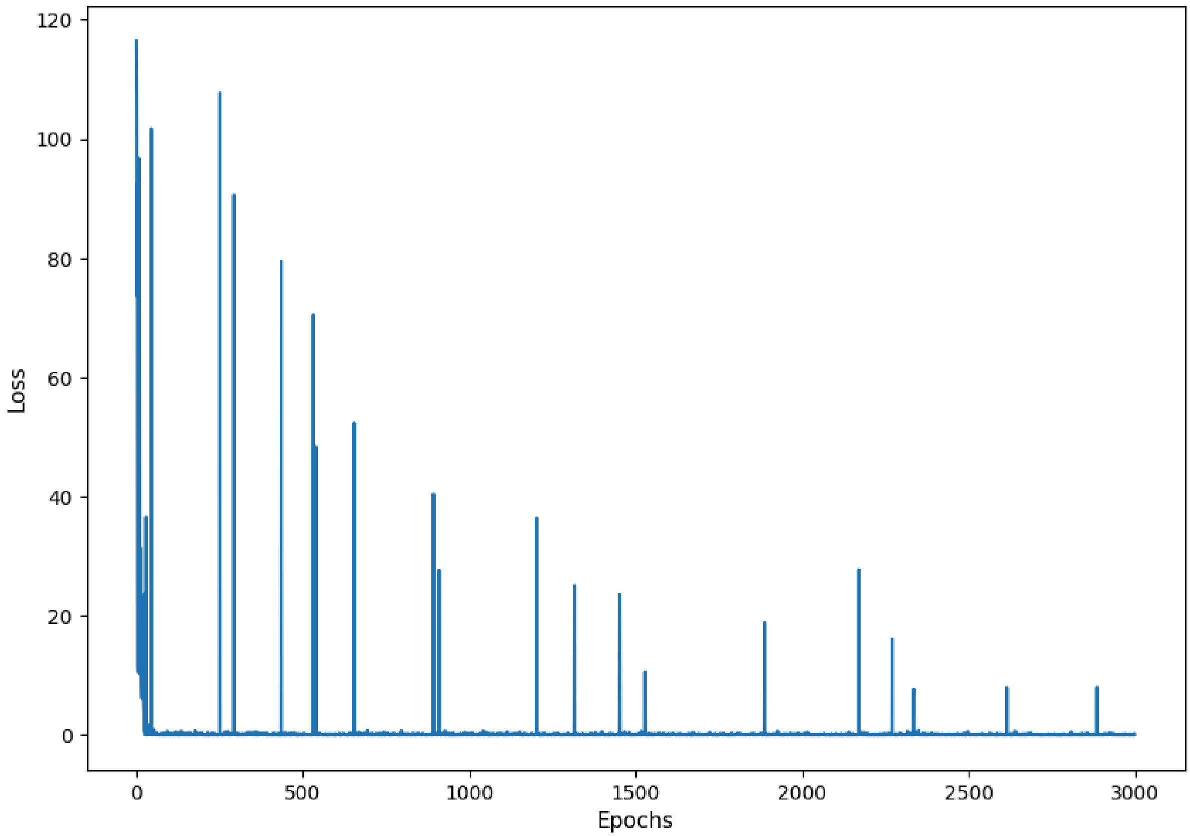
- test in random mode

```
Initial State:
[[' ' ' ' '-' ' ']
 ['W' ' ' 'P' ' ']
 [' ' '+' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: l
[[' ' ' ' '-' ' ']
 ['W' 'P' ' ' ' ']
 [' ' '+' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: d
[[' ' ' ' '-' ' ']
 ['W' ' ' ' ' ' ']
 [' ' '+' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Game won! Reward: 10
True
Initial State:
[[' ' ' ' ' ' 'P']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']
 ['W' '-' '+' ' ']]
Move #: 0; Taking action: l
[[' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 726
Win percentage: 72.6%
```

Dueling DQN for static mode

```
epochs = 3000
losses = [] # 使用串列將每一次的loss記錄下來，方便之後將loss的變化趨勢畫成圖
for i in range(epochs):
    game = Gridworld(size=4, mode='static')
    state_ = game.board.render_np().reshape(1, 64) + np.random.rand(1, 64) / 1
    state1 = torch.from_numpy(state_).float().to(device) # 將NumPy陣列轉換成P
    status = 1 # 用來追蹤遊戲是否仍在繼續（『1』代表仍在繼續）
    while status == 1:
        qval = model(state1) # 執行Q網路，取得所有動作的預測Q值
        qval_ = qval.cpu().data.numpy() # 將qval轉換成NumPy陣列
        if random.random() < epsilon:
```

```
            action_ = np.random.randint(0, 4)  # 隨機選擇一個動作（探索）
        else:
            action_ = np.argmax(qval_)  # 選擇Q值最大的動作（探索）
        action = action_set[action_]  # 將代表某動作的數字對應到makeMove()的英
        game.makeMove(action)  # 執行之前ε—貪婪策略所選出的動作
        state2_ = game.board.render_np().reshape(1, 64) + np.random.rand(1, 64
        state2 = torch.from_numpy(state2_).float().to(device)  # 動作執行完畢，取
        reward = game.reward()
        with torch.no_grad():
            newQ = model(state2.reshape(1, 64))
        maxQ = torch.max(newQ)  # 將新狀態下所輸出的Q值向量中的最大值給記錄
        if reward == -1:
            Y = reward + (gamma * maxQ)  # 計算訓練所用的目標Q值
        else:  # 若reward不等於-1，代表遊戲已經結束，也就沒有下一個狀態了，因此
            Y = reward
        Y = torch.Tensor([Y]).detach()
        Y = Y.to(device)
        X = qval.squeeze()[action_].to(device)  # 將演算法對執行的動作所預測的Q
        loss = loss_fn(X, Y)  # 計算目標Q值與預測Q值之間的誤差
        if i % 100 == 0:
            print(i, loss.item())
            clear_output(wait=True)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        state1 = state2
        if abs(reward) == 10:
            status = 0  # 若 reward 的絕對值為10，代表遊戲已經分出勝負，所以設sta
    losses.append(loss.item())
    if epsilon > 0.1:
        epsilon -= (1 / epochs)  # 讓ε的值隨著訓練的進行而慢慢下降，直到0.1（還是

plt.figure(figsize=(10, 7))
plt.plot(losses)
plt.xlabel("Epochs", fontsize=11)
plt.ylabel("Loss", fontsize=11)
```



- test in static mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' 'P']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' 'P' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in player mode

```
Initial State:
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: u
[['+' '-' ' ' ' ']
 ['P' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: u
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

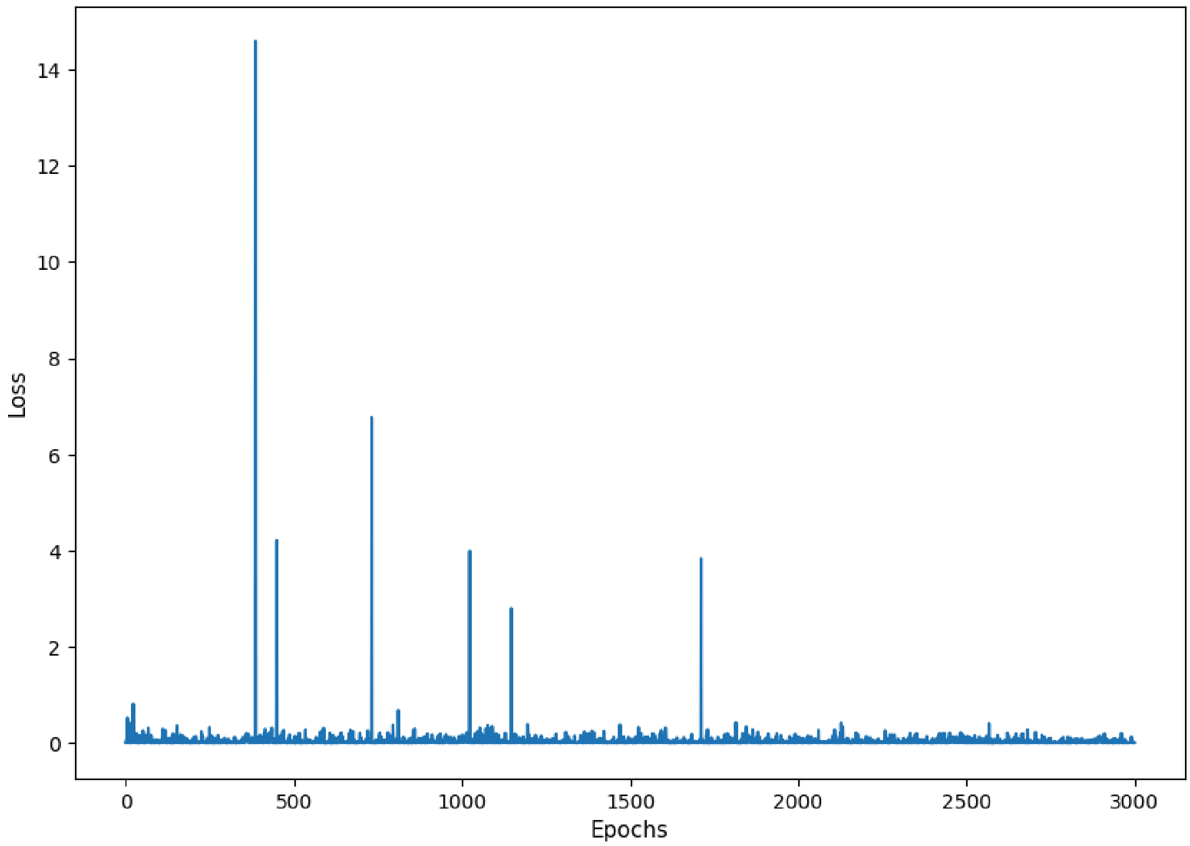
- test in random mode

```
Initial State:
[[' ' ' ' '+' ' ']
 [' ' '-' 'W' ' ']
 [' ' ' ' ' ' ' ']
 ['P' ' ' ' ' ' ']]
Move #: 0; Taking action: u
[[' ' ' ' '+' ' ']
 [' ' '-' 'W' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[[' ' ' ' '+' ' ']
 [' ' '-' 'W' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[[' ' ' ' '+' ' ']
 [' ' '-' 'W' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: l
[[' ' ' ' '+' ' ']
 [' ' '-' 'W' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game lost; too many moves.
False
Games played: 1000, # of wins: 638
Win percentage: 63.800000000000004%
```

Dueling DQN for player mode

```
epochs = 3000
losses = []  # 使用串列將每一次的loss記錄下來，方便之後將loss的變化趨勢畫成圖
for i in range(epochs):
    game = Gridworld(size=4, mode='player')
    state_ = game.board.render_np().reshape(1, 64) + np.random.rand(1, 64) / 1
    state1 = torch.from_numpy(state_).float().to(device)  # 將NumPy陣列轉換成P
    status = 1  # 用來追蹤遊戲是否仍在繼續（『1』代表仍在繼續）
    while status == 1:
        qval = model(state1)  # 執行Q網路，取得所有動作的預測Q值
        qval_ = qval.cpu().data.numpy()  # 將qval轉換成NumPy陣列
        if random.random() < epsilon:
```

```
            action_ = np.random.randint(0, 4)  # 隨機選擇一個動作（探索）
        else:
            action_ = np.argmax(qval_)  # 選擇Q值最大的動作（探索）
        action = action_set[action_]  # 將代表某動作的數字對應到makeMove()的英
        game.makeMove(action)  # 執行之前ε—貪婪策略所選出的動作
        state2_ = game.board.render_np().reshape(1, 64) + np.random.rand(1, 64
        state2 = torch.from_numpy(state2_).float().to(device)  # 動作執行完畢，取
        reward = game.reward()
        with torch.no_grad():
            newQ = model(state2.reshape(1, 64))
        maxQ = torch.max(newQ)  # 將新狀態下所輸出的Q值向量中的最大值給記錄
        if reward == -1:
            Y = reward + (gamma * maxQ)  # 計算訓練所用的目標Q值
        else:  # 若reward不等於-1，代表遊戲已經結束，也就沒有下一個狀態了，因此
            Y = reward
        Y = torch.Tensor([Y]).detach()
        Y = Y.to(device)
        X = qval.squeeze()[action_].to(device)  # 將演算法對執行的動作所預測的Q
        loss = loss_fn(X, Y)  # 計算目標Q值與預測Q值之間的誤差
        if i % 100 == 0:
            print(i, loss.item())
            clear_output(wait=True)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        state1 = state2
        if abs(reward) == 10:
            status = 0  # 若 reward 的絕對值為10，代表遊戲已經分出勝負，所以設sta
    losses.append(loss.item())
    if epsilon > 0.1:
        epsilon -= (1 / epochs)  # 讓ε的值隨著訓練的進行而慢慢下降，直到0.1（還是

plt.figure(figsize=(10, 7))
plt.plot(losses)
plt.xlabel("Epochs", fontsize=11)
plt.ylabel("Loss", fontsize=11)
```



- test in static mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' 'P']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' 'P' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: d
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in player mode

```
Initial State:
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' 'P']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 3; Taking action: u
[['+' '-' ' ' ' ']
 ['P' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

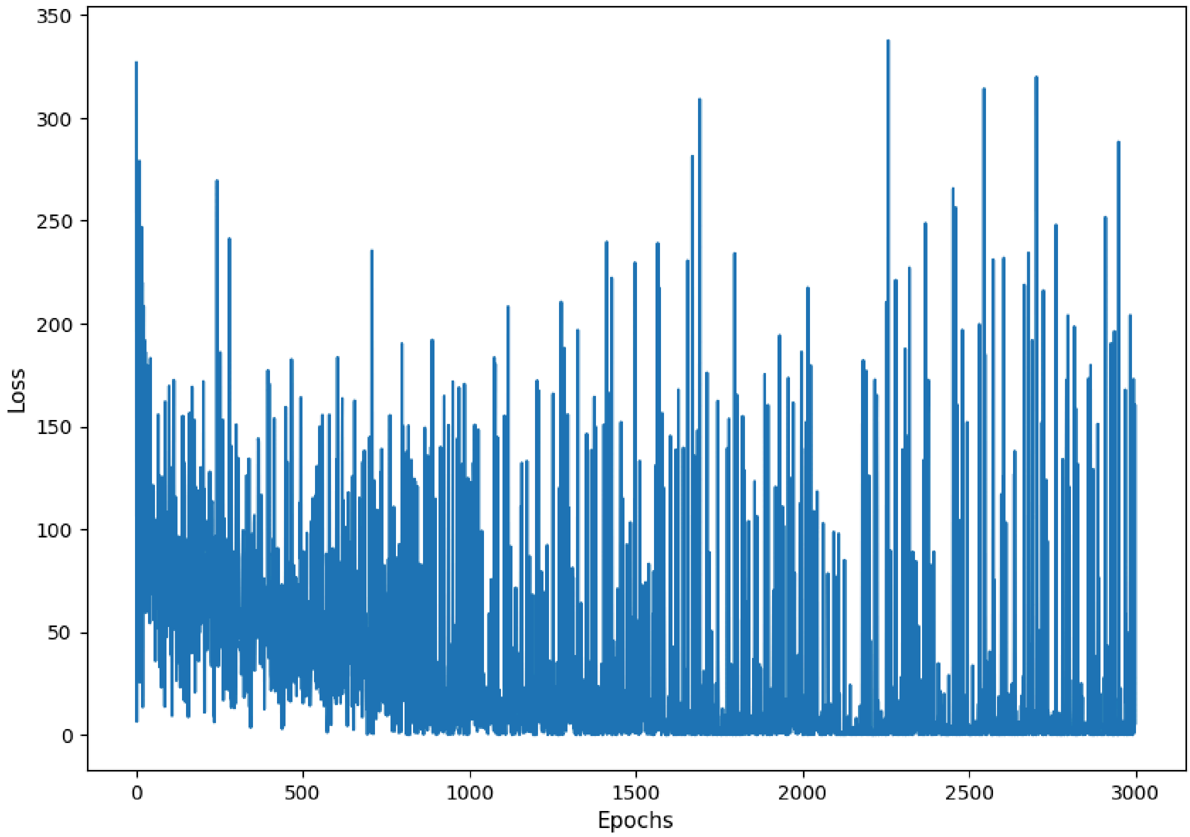
- test in random mode

```
Initial State:
[['-' ' ' ' ' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' '+' ' ']
 [' ' ' ' ' ' 'W']]
Move #: 0; Taking action: r
[['-' ' ' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' '+' ' ']
 [' ' ' ' ' ' 'W']]
Move #: 1; Taking action: r
[['-' ' ' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' '+' ' ']
 [' ' ' ' ' ' 'W']]
Move #: 2; Taking action: d
[['-' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' '+' ' ']
 [' ' ' ' ' ' 'W']]
Game won! Reward: 10
True
Initial State:
[[' ' ' ' ' ' 'P']
 ['-' ' ' ' ' ' ']
...
Game lost; too many moves.
False
Games played: 1000, # of wins: 624
Win percentage: 62.4%
```

Dueling DQN for random mode

```
epochs = 3000
losses = []  # 使用串列將每一次的loss記錄下來，方便之後將loss的變化趨勢畫成圖
for i in range(epochs):
    game = Gridworld(size=4, mode='random')
    state_ = game.board.render_np().reshape(1, 64) + np.random.rand(1, 64) / 1
    state1 = torch.from_numpy(state_).float().to(device)  # 將NumPy陣列轉換成P
    status = 1  # 用來追蹤遊戲是否仍在繼續（『1』代表仍在繼續）
    while status == 1:
        qval = model(state1)  # 執行Q網路，取得所有動作的預測Q值
        qval_ = qval.cpu().data.numpy()  # 將qval轉換成NumPy陣列
        if random.random() < epsilon:
```

```
            action_ = np.random.randint(0, 4)  # 隨機選擇一個動作（探索）
        else:
            action_ = np.argmax(qval_)  # 選擇Q值最大的動作（探索）
        action = action_set[action_]  # 將代表某動作的數字對應到makeMove()的英
        game.makeMove(action)  # 執行之前ε—貪婪策略所選出的動作
        state2_ = game.board.render_np().reshape(1, 64) + np.random.rand(1, 64
        state2 = torch.from_numpy(state2_).float().to(device)  # 動作執行完畢，取
        reward = game.reward()
        with torch.no_grad():
            newQ = model(state2.reshape(1, 64))
        maxQ = torch.max(newQ)  # 將新狀態下所輸出的Q值向量中的最大值給記錄
        if reward == -1:
            Y = reward + (gamma * maxQ)  # 計算訓練所用的目標Q值
        else:  # 若reward不等於-1，代表遊戲已經結束，也就沒有下一個狀態了，因此
            Y = reward
        Y = torch.Tensor([Y]).detach()
        Y = Y.to(device)
        X = qval.squeeze()[action_].to(device)  # 將演算法對執行的動作所預測的Q
        loss = loss_fn(X, Y)  # 計算目標Q值與預測Q值之間的誤差
        if i % 100 == 0:
            print(i, loss.item())
            clear_output(wait=True)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        state1 = state2
        if abs(reward) == 10:
            status = 0  # 若 reward 的絕對值為10，代表遊戲已經分出勝負，所以設sta
    losses.append(loss.item())
    if epsilon > 0.1:
        epsilon -= (1 / epochs)  # 讓ε的值隨著訓練的進行而慢慢下降，直到0.1（還是

plt.figure(figsize=(10, 7))
plt.plot(losses)
plt.xlabel("Epochs", fontsize=11)
plt.ylabel("Loss", fontsize=11)
```



- test in static mode

```
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: l
[['+' '-' 'P' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 2; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Game won! Reward: 10
True
Initial State:
[['+' '-' ' ' 'P']
 [' ' 'W' ' ' ' ']
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 1000
Win percentage: 100.0%
```

- test in player mode

```
Initial State:
[['+' '-' ' ' ' ']
 ['P' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: u
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']]
Game won! Reward: 10
True
Initial State:
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' ' ' 'P']
 [' ' ' ' ' ' ' ']]
Move #: 0; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' ' ' ' ']]
Move #: 1; Taking action: l
[['+' '-' ' ' ' ']
 [' ' 'W' ' ' ' ']
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 857
Win percentage: 85.7%
```

- test in random mode

```
Initial State:
[['+' ' ' 'W' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' 'P' ' ']
 [' ' ' ' '-' ' ']]
Move #: 0; Taking action: l
[['+' ' ' 'W' ' ']
 [' ' ' ' ' ' ' ']
 [' ' 'P' ' ' ' ']
 [' ' ' ' '-' ' ']]
Move #: 1; Taking action: l
[['+' ' ' 'W' ' ']
 [' ' ' ' ' ' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' '-' ' ']]
Move #: 2; Taking action: u
[['+' ' ' 'W' ' ']
 ['P' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' '-' ' ']]
Move #: 3; Taking action: u
[['+' ' ' 'W' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' ' ' ' ']
 [' ' ' ' '-' ' ']]
...
Game won! Reward: 10
True
Games played: 1000, # of wins: 931
Win percentage: 93.10000000000001%
```