長庚大學 資訊工程學系

112-2 人工智慧課程

Reversed Reversi 專題書面報告

組別：第 8 組

組長：資工四　B0929009　宋昭佑

組員：資工四　B0843042　黃子庭

資工四　B0922044　張祐瑜

資工三　B0829002　廖洺玄

目錄

壹、比賽規則

依照黑白棋的規則下棋，以O與X進行比賽：

1. 開局：X置於(3,3)、(4,4)，O置於(3,4)、(4,3)（如圖 1-1）。
2. 先後順序：X先手O後手，雙方輪流，只在無法夾殺對方時可PASS。
3. 落子機制：新落子在 8 方向上與己方另一子連線，把連線上的對方棋子變成己方（如圖 1-2）。
4. 輸入輸出：輸入X先手開局後即開始計算時間並輸出耗時、落子位置、狀態繪圖。
5. 時間計算：每次輸入之後的回應時間必須小於 60 秒，每局有 3 次機會可延長該次回應最多至 120 秒。

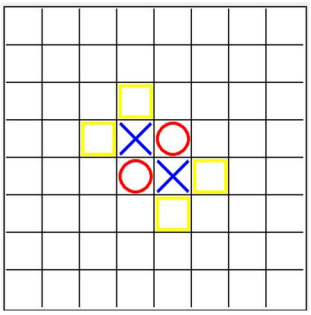

圖 1-1：開局X與O置於指定位置
黃色框為可下之位置選項

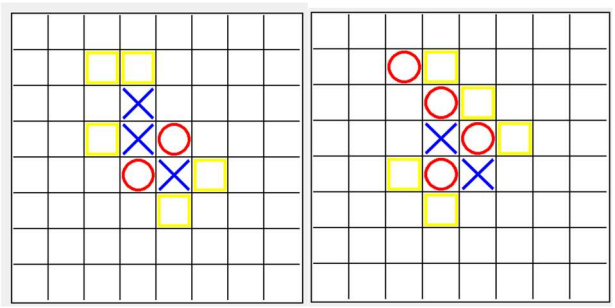

圖 1-2：開局後先經X(3,2)（左圖）
後經O(2,1)（右圖）之結果示意圖

貳、實作方法

一、初始設定

首先，設定棋盤的初始位置，1 代表 X，2 代表 O，3 代表可以下的位置，之後透過黑白棋的遊戲規則去設定棋盤中每個位置的權重值（如圖 2-1-1），其他參數是提供給所有函式所使用（如圖 2-1-2）。

```
# Board initialization
''' 0: empty, 1: cross, 2: circle, 3: points can be placed'''
board = np.array([[0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 3, 0, 0, 0],
                  [0, 0, 0, 1, 2, 3, 0, 0],
                  [0, 0, 3, 2, 1, 0, 0, 0],
                  [0, 0, 0, 3, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0, 0, 0, 0]])
initBoard = board.copy()
# Board weight
weight = np.array([[-3,  9, -1, -1, -1, -1,  9, -3],
                   [ 9,  9, -2, -2, -2, -2,  9,  9],
                   [-1, -2, -2, -2, -2, -2, -2, -1],
                   [-1, -2, -2, -3, -3, -2, -2, -1],
                   [-1, -2, -2, -3, -3, -2, -2, -1],
                   [-1, -2, -2, -2, -2, -2, -2, -1],
                   [ 9,  9, -2, -2, -2, -2,  9,  9],
                   [-3,  9, -1, -1, -1, -1,  9, -3]])
initweight = weight.copy()
```

圖 2-1-1：設定棋盤的初始位置與權重值

```
# Record moves & hands & positions & notes
boardRecord = [board.copy()]
whoRecord = [2]
nowPos = 0
notelist = [''] * 64
gameover = 0
# 0: close, 1: cross, 2: circle
aiturn = 0
start_time = 0
end_time = 0
wcount = 0
bcount = 0
total_time = 0
```

圖 2-1-2：設定其他參數

二、畫面描繪

透過 render()函式（如圖 2-2）將製作棋盤上的內容，除了整體棋盤的架構，亦包含棋盤每個格子的判斷，其中包括 0 代表空值、1 代表藍色 X、2 代表紅色 O、3 則代表黃色框框以標示可下的位置。

```
def render():
    global board, notelist, nowPos
    gameBox.delete("all")
    for i in range(1, 8):
        gameBox.create_line(60*i, 0, 60*i, 480, width=2, fill='black')
        gameBox.create_line(0, 60*i, 480, 60*i, width=2, fill='black')
    for i in range(8):
        for j in range(8):
            if board[i][j] == 0:    # empty
                gameBox.create_oval(i*60+5, j*60+5, i*60+55, j*60+55, fill='white', outline='white')
            elif board[i][j] == 1:  # cross
                gameBox.create_line(i*60+5, j*60+5, i*60+55, j*60+55, fill='#0000ff', width=5)
                gameBox.create_line(i*60+5, j*60+55, i*60+55, j*60+5, fill='#0000ff', width=5)
            elif board[i][j] == 2:   # circle
                gameBox.create_oval(i*60+5, j*60+5, i*60+55, j*60+55, fill='', outline='#ff0000', width=5)
            elif board[i][j] == 3:  # points can be placed
                gameBox.create_rectangle(i*60+5, j*60+5, i*60+55, j*60+55, fill='', outline='yellow', width=5)
    gameBox.update_idletasks()
    pos['text'] = str(nowPos)
```

圖 2-2：render 函式

三、模式切換

本專題提供兩種模式，第一種為對方先攻（如圖 2-3-1），由點擊畫面中的「You First」進入，此時對方棋子為X，己方棋子為O；第二種是己方先攻（如圖 2-3-2），由點擊畫面中的「AI First」進入，此時己方棋子為X，對方棋子為O。在程式中 whoturn 為 1 時代表先手，aiturn 代表己方棋手，當目前是己方先攻時，使用 runAI()函式執行棋路預判，同時紀錄 AI 計算時間並顯示於畫面上，接著使用 reversi()函式進行落子以及計算可下之位置，最後使用 render()函式將計算結果顯示於棋盤上，並使用 writeNote()函式將落子位置座標紀錄於 Note 中。

```
def switchMode(mode):
    global whoturn, aiturn, board, nowPos, total, final_x, final_y, total_time
    # Player mode
    if mode == 0:
        whoturn = 1
        player['relief'] = SUNKEN
        player['state'] = DISABLED
        ai['relief'] = RAISED
        ai['state'] = DISABLED
        aiturn = 2
```

圖 2-3-1：對手先攻模式

```
else:
    whoturn = 1
    ai['relief'] = SUNKEN
    ai['state'] = DISABLED
    player['relief'] = RAISED
    player['state'] = DISABLED
    aiturn = whoturn
    start_time = time.time()
    runAI()
    end_time = time.time()
    total_time = total_time + (end_time - start_time)
    AItime['text'] = str("{:1.10f}".format(end_time - start_time))
    Totaltime['text'] = str("{:1.10f}".format(total_time))
    if whoturn == 1:
        step = f"Move {nowPos+1} : X ({ final_x}, {final_y})\n"
        writeNote(step)
        print(str(nowPos) + " X " + str((final_x, final_y)))
    else:
        step = f"Move {nowPos+1} : O ({ final_x}, {final_y})\t"
        writeNote(step)
        print(str(nowPos) + " O " + str((final_x, final_y)))
    nowboard = board.copy()
    whoturn, chesscount, g = reversi(nowboard, final_x, final_y, whoturn)
    nowPos += 1
    board = nowboard.copy()
    render()
    numHint()
    boardRecord.append(board)
    whoRecord.append(whoturn)
```

圖 2-3-2：己方先攻模式

四、點擊偵測

首先，判定是對方或己方出手，若是對方出手（如圖 2-4-1），則針對點擊位置確認是否為可落子之位置，並將該位置利用 writeNote()函式將位置座標紀錄於 Note 中。將原先的棋盤資訊清掉後，使用 reversi()函式進行落子以及計算可下之位置，使用 weightChange()函式重新計算權重值，使用 render()函式將計算結果顯示於棋盤上，最後使用 numHint()函式去計算目前比數並顯示在畫面上。

而若是己方出手（如圖 2-4-2），使用 runAI()函式執行棋路預判，同時紀錄 AI 計算時間並顯示於畫面上，接著同樣使用 reversi()函式進行落子以及計算可下之位置，使用 weightChange()函式重新計算權重值，使用 render()函式將計算結果顯示於棋盤上，最後使用 numHint()函式去計算目前比數並顯示在畫面上。

```
def mouseCatch(event):
    global board, whoRecord, boardRecord, nowPos, aiturn, whoturn, final_x, final_y, gameover, weight, total_time
    mx = int(event.x/60)
    my = int(event.y/60)
    if aiturn != whoturn and gameover == 0:
        if board[mx][my] == 3:
            if whoturn == 1:
                step = f"Move {nowPos+1} : X ({mx}, {my})\n"
                writeNote(step)
                print(str(nowPos) + " X " + str((mx, my)))
            else:
                step = f"Move {nowPos+1} : O ({mx}, {my})\t"
                writeNote(step)
                print(str(nowPos) + " O " + str((mx, my)))

            if nowPos < (len(whoRecord)-1):
                del boardRecord[nowPos+1:]
                del whoRecord[nowPos+1:]
            nowboard = board.copy()
            whoturn, chesscount, gameover = reversi(nowboard, mx, my, whoturn)
            weightChange(final_x, final_y, 1)
            nowPos += 1

            board = nowboard.copy()
            boardRecord.append(board)
            whoRecord.append(whoturn)
            render()
            numHint()
```

圖 2-4-1：mouseCatch 函式（對方出手）

```
if aiturn == whoturn:
    if aiturn != 0 and nowPos+1 == len(whoRecord) and aiturn == whoturn:
        while whoturn == aiturn and gameover == 0:
            numHint()
            start_time = time.time()
            runAI()
            end_time = time.time()
            total_time = total_time + (end_time - start_time)
            AItime['text'] = str("{:1.10f}".format(end_time - start_time))
            Totaltime['text'] = str("{:1.10f}".format(total_time))
            if whoturn == 1:
                step = f"Move {nowPos+1} : X ({ final_x}, {final_y})\n"
                writeNote(step)
                print(str(nowPos) + " X " + str((final_x, final_y)))
            else:
                step = f"Move {nowPos+1} : O ({ final_x}, {final_y})\t"
                writeNote(step)
                print(str(nowPos) + " O " + str((final_x, final_y)))
            nowboard = board.copy()
            whoturn, chesscount, gameover = reversi(nowboard, final_x, final_y, whoturn)
            weightChange(final_x, final_y, 0)
            nowPos += 1
            board = nowboard.copy()
            boardRecord.append(board)
            whoRecord.append(whoturn)
            render()
            numHint()
```

圖 2-4-2：mouseCatch 函式(己方出手)

五、翻棋功能

透過 reversi()函式（如圖 2-5-1）將落子位置轉換成目前棋手的棋子，再使用 turnOver()函式（如圖 2-5-2）將該位置的八個方位進行搜尋，包含上下左右、右上右下、左上左下，找到己方棋子後將夾在中間的對方棋子都轉換成己方棋子。此外，還要偵測兩次棋盤上對方或己方是否沒有位置可落子，第一次偵測來判斷是否 pass，第二次偵測來判斷是否已結束遊戲。

```python
def reversi(nowboard, mx, my, whoturn):
    global board, boardRecord, whoRecord, wcount, bcount
    gameover = 0

    if whoturn == 2:
        nowboard[mx][my] = 2
        chesscount = turnOver(nowboard, mx, my, 2)
        whoturn = 1
    else:
        nowboard[mx][my] = 1
        chesscount = turnOver(nowboard, mx, my, 1)
        whoturn = 2

    permit = 0
    for i in range(8):
        for j in range(8):
            if nowboard[i][j] == 3:
                nowboard[i][j] = 0
            if nowboard[i][j] == 0:
                tryboard = nowboard.copy()
                turnOver(tryboard, i, j, whoturn)
                if (tryboard != nowboard).any():
                    permit = 1
                    nowboard[i][j] = 3
```

```python
if permit == 0:
    if whoturn == 2:
        step = f"Move {nowPos+1} : O Pass\n"
        writeNote(step)
        print(str(nowPos) + " O pass")
        whoturn = 1
    else:
        step = f"Move {nowPos+1} : X Pass\n"
        writeNote(step)
        print(str(nowPos) + " X pass")
        whoturn = 2
    for i in range(8):
        for j in range(8):
            if nowboard[i][j] == 3:
                nowboard[i][j] = 0
            if nowboard[i][j] == 0:
                tryboard = nowboard.copy()
                turnOver(tryboard, i, j, whoturn)
                if (tryboard != nowboard).any():
                    permit = 1
                    nowboard[i][j] = 3
if permit == 0 :
    if wcount < bcount:
        step = f"Move {nowPos+1} : X win\n"
        writeNote(step)
        print(str(nowPos) + " X win")
    elif wcount > bcount:
        step = f"Move {nowPos+1} : O win\n"
        writeNote(step)
        print(str(nowPos) + " O win")
    else:
        step = f"Move {nowPos+1} : Draw\n"
        writeNote(step)
        print(str(nowPos) + " Draw")
    gameover = 1

return whoturn, chesscount, gameover
```

圖 2-5-1：reversi 函式

```python
def turnOver(nowboard, mx, my, whoturn):
    chesscount = 0
    # Search up
    for i in range(my-1, -1, -1):
        if nowboard[mx][i] == 0 or nowboard[mx][i] == 3:
            break
        if nowboard[mx][i] == whoturn:
            for j in range(my-1, i, -1):
                nowboard[mx][j] = whoturn
                chesscount += 1
            break
    # Search right
    for i in range(mx+1, 8):
        if nowboard[i][my] == 0 or nowboard[i][my] == 3:
            break
        if nowboard[i][my] == whoturn:
            for j in range(mx+1, i):
                nowboard[j][my] = whoturn
                chesscount += 1
            break
    # Search down
    for i in range(my+1, 8):
        if nowboard[mx][i] == 0 or nowboard[mx][i] == 3:
            break
        if nowboard[mx][i] == whoturn:
            for j in range(my+1, i):
                nowboard[mx][j] = whoturn
                chesscount += 1
            break
    # Search left
    for i in range(mx-1, -1, -1):
        if nowboard[i][my] == 0 or nowboard[i][my] == 3:
            break
        if nowboard[i][my] == whoturn:
            for j in range(mx-1, i, -1):
                nowboard[j][my] = whoturn
                chesscount += 1
            break
```

```python
# Search to the upper right
for i in range(1, 8):
    if (mx+i) <= 7 and (my-i) >= 0:
        if nowboard[mx+i][my-i] == 0 or nowboard[mx+i][my-i] == 3:
            break
        if nowboard[mx+i][my-i] == whoturn:
            for j in range(1, i):
                nowboard[mx+j][my-j] = whoturn
                chesscount += 1
            break
# Search to the lower right
for i in range(1, 8):
    if (mx+i) <= 7 and (my+i) <= 7:
        if nowboard[mx+i][my+i] == 0 or nowboard[mx+i][my+i] == 3:
            break
        if nowboard[mx+i][my+i] == whoturn:
            for j in range(1, i):
                nowboard[mx+j][my+j] = whoturn
                chesscount += 1
            break
# Search to the lower left
for i in range(1, 8):
    if (mx-i) >= 0 and (my+i) <= 7:
        if nowboard[mx-i][my+i] == 0 or nowboard[mx-i][my+i] == 3:
            break
        if nowboard[mx-i][my+i] == whoturn:
            for j in range(1, i):
                nowboard[mx-j][my+j] = whoturn
                chesscount += 1
            break
# Search to the upper left
for i in range(1, 8):
    if (mx-i) >= 0 and (my-i) >= 0:
        if nowboard[mx-i][my-i] == 0 or nowboard[mx-i][my-i] == 3:
            break
        if nowboard[mx-i][my-i] == whoturn:
            for j in range(1, i):
                nowboard[mx-j][my-j] = whoturn
                chesscount += 1
            break
return chesscount
```

圖 2-5-2：turnOver 函式

六、AI 運算

使用 runAI()函式（如圖 2-6-1）定義好所需變數，之後呼叫 myAI()函式（如圖 2-6-2）開始進行 AI 運算並偵測所有可落子的位置，並透過 alpha_beta()函式（如圖 2-6-3）進行棋路決策預判，當計算出最佳分數時，則選擇該位置作為下一步落子位置。此處的 alpha_beta()函式是本組使用上課教過的 Alpha-Beta 剪枝法來幫助計算最好的棋路位置，當迴圈只有零層時，使用 heuristic()函式（如圖 2-6-4）根據棋盤的權重值計算出分數；當現在迴圈多於零層時，根據此次的遊戲規則先決定 min 層代表己方，max 層代表對方，之後再透過 Alpha-Beta 剪枝的方式計算出分數。

```
def runAI():
    global tmp_x, tmp_y, final_x, final_y, tmp_chesscount, final_min, board, whoturn, aiturn, total, weight
    tmp_x, tmp_y, final_x, final_y, tmp_chesscount, final_min = -1, -1, -1, -1, -1, 100
    myAI(whoturn, board, total, weight)
```

圖 2-6-1：runAI 函式

```
def myAI(whoturn, board, total, weight):
    best_move = (-1, -1)
    best_score = float('-inf')
    for i in range(8):
        for j in range(8):
            if board[i][j] == 3:
                print(f"Evaluating move at ({i}, {j})")
                aiboard = board.copy()
                _, _, _ = reversi(aiboard, i, j, whoturn)
                score = alpha_beta(whoturn, aiboard, 5, float('-inf'), float('inf'), True)
                print(f"Score for move ({i}, {j}): {score}")
                if score > best_score:
                    best_score = score
                    best_move = (i, j)
    print(f"Best move selected: {best_move} with score {best_score}")
    global final_x, final_y
    final_x, final_y = best_move
    return best_move
```

圖 2-6-2：myAI 函式

```python
tmp_x, tmp_y, final_x, final_y, tmp_chesscount, final_min = -1, -1, -1, -1, -1, 100
def alpha_beta(whoturn, board, depth, alpha, beta, is_maximizing_player):

    if depth == 0 or is_game_over(board):
        return heuristic(board)

    if is_maximizing_player:
        value = float('-inf')
        for i in range(8):
            for j in range(8):
                if board[i][j] == 3:
                    aiboard = board.copy()
                    _, _, _ = reversi(aiboard, i, j, whoturn)
                    value = max(value, alpha_beta(whoturn, aiboard, depth - 1, alpha, beta, False))
                    alpha = max(alpha, value)
                    if alpha >= beta:
                        break
        return value
    else:
        value = float('inf')
        for i in range(8):
            for j in range(8):
                if board[i][j] == 3:
                    aiboard = board.copy()
                    _, _, _ = reversi(aiboard, i, j, whoturn)
                    value = min(value, alpha_beta(whoturn, aiboard, depth - 1, alpha, beta, True))
                    beta = min(beta, value)
                    if beta <= alpha:
                        break
        return value
```

圖 2-6-3：alpha_beta 函式

```python
def heuristic(board):
    weights = [
        [ 20, -10,  5,  5,  5,  5, -10,  20],
        [-10, -20, -1, -1, -1, -1, -20, -10],
        [  5,  -1, -2, -2, -2, -2,  -1,   5],
        [  5,  -1, -2,  0,  0, -2,  -1,   5],
        [  5,  -1, -2,  0,  0, -2,  -1,   5],
        [  5,  -1, -2, -2, -2, -2,  -1,   5],
        [-10, -20, -1, -1, -1, -1, -20, -10],
        [ 20, -10,  5,  5,  5,  5, -10,  20]
    ]

    my_score = 0
    opponent_score = 0
    for i in range(8):
        for j in range(8):
            if board[i][j] == 1:
                my_score += weights[i][j]
            elif board[i][j] == 2:
                opponent_score += weights[i][j]
    return opponent_score - my_score
```

圖 2-6-4：heuristic 函式

七、計算權重

當現在 AI 計算出的最佳位置在四個角落時，weightChange()函式將棋盤的四個邊的權重值重新計算。

```
def weightChange(x, y, who):
    global weight
    if x == 0 and y == 0:
        weight[x+1][y+1] = 9
        for i in range(1,7):
            weight[x+i][y] = 10-i
            weight[x][y+i] = 10-i
    elif x == 7 and y == 0:
        weight[x-1][y+1] = 9
        for i in range(1,7):
            weight[x-i][y] = 10-i
            weight[x][y+i] = 10-i
    elif x == 7 and y == 7:
        weight[x-1][y-1] = 9
        for i in range(1,7):
            weight[x-i][y] = 10-i
            weight[x][y-i] = 10-i
    elif x == 0 and y == 7:
        weight[x+1][y-1] = 9
        for i in range(1,7):
            weight[x+i][y] = 10-i
            weight[x][y-i] = 10-i
```

圖 2-7：weightChange 函式

八、重新功能

restart()函式會將所有參數初始化，棋盤和權重值也都會重置，連同上一局棋盤的紀錄也會刪除。

```
def restart():
    global board, boardRecord, whoRecord, initBoard, whoturn, nowPos, aiturn, total, gameover, weight, initweight, final_x, final_y, notelist, total_time
    total = 0
    gameover = 0
    notelist = [''] * 64
    note_text.set('')
    board = initBoard.copy()
    weight = initweight.copy()
    del boardRecord[1:]
    del whoRecord[1:]
    whoturn = 0
    nowPos = 0
    aiturn = 0
    total_time = 0
    render()
    numHint()
    player['relief'] = RAISED
    player['state'] = NORMAL
    ai['relief'] = RAISED
    ai['state'] = NORMAL
    AItime['text'] = str(0)
    Totaltime['text'] = str(0)
```

圖 2-8：restart 函式

九、判定結束

利用 is_game_over()函式偵測棋盤是否已經沒有可落子的棋路。

```
def is_game_over(board):
    for i in range(8):
        for j in range(8):
            if board[i][j] == 3:
                return False
    return True
```

圖 2-9：is_game_over 函式

十、計算分數

numHint()函式會去計算目前棋盤內雙方的棋子數量來計算比分，並顯示在畫面上，同時也會利用框線顯示目前下棋者是哪方。

```
def numHint():
    global total, whoturn, board, wcount, bcount
    whitenum = 0
    blacknum = 0
    for i in range(8):
        for j in range(8):
            if board[i][j] == 1:
                whitenum += 1
            elif board[i][j] == 2:
                blacknum += 1
    total = whitenum + blacknum
    wcount = whitenum
    bcount = blacknum
    wnum['text'] = str(whitenum)
    bnum['text'] = str(blacknum)
    if whoturn == 1:
        wnum['relief'] = "solid"
        bnum['relief'] = "flat"
    elif whoturn == 2:
        bnum['relief'] = "solid"
        wnum['relief'] = "flat"
```

圖 2-10：numHint 函式

十一、紀錄棋路

writeNote()函式會將目前的棋路記錄起來，並顯示在畫面中。

```
def writeNote(step):
    global notelist, nowPos, note_text
    notation = step
    notelist[nowPos] = notation
    note_content = "".join(notelist)
    note_text.set(note_content)
```

圖 2-11：writeNote 函式

十二、介面設計

本組設計本專題介面採用 tkinter 套件建構，首先建構出本專題核心的棋盤置於中央（如圖 2-5-1），接著設置模式控制（如圖 2-5-2），其中包含重置的 restart 鍵、決定先後手的 You First/AI First 鍵、以及相對應的 Move 次序顯示，且為方便在對奕之時即時觀察戰況，除了設置雙方之比數顯示（如圖 2-5-3），亦設計回應時間顯示（如圖 2-5-4），包含每次輸入之後的回應時間以及總回應時間；最後，以條列每一步落子位置的方式，進行雙方座標紀錄（如圖 2-5-5）。結合以上元素，以完成本組專題的介面呈現（如圖 2-5-6）。

```
root = Tk()
root.state("zoomed")
root.resizable(False, False)

window_width = root.winfo_screenwidth()
window_height = root.winfo_screenheight()
myframe = Frame(root, bg='black', padx=2, pady=2)
myframe.place(x=(window_width - 480) // 2, y=(window_height - 480) // 2)
mysize = 480
gameBox = Canvas(myframe, width=mysize, height=mysize, bg='white')
gameBox.pack()
for i in range(1,8):
    gameBox.create_line(60*i, 0, 60*i, mysize+2, width=2, fill='black')
    gameBox.create_line(0, 60*i, mysize+2, 60*i, width=2, fill='black')
```

圖 2-5-1：建構棋盤

```
restart = Button(root, text="Restart", width=12, height=1, bg='#55ee55', font=('Arial', 20), command=restart)
restart.place(x=(window_width - 480) // 2 + 600, y=(window_height - 480) // 2 + 50)
player = Button(root, text="You First", width=12, height=1, bg='blue', fg='white', font=('Arial', 20), command=lambda:switchMode(0))
player.place(x=(window_width - 480) // 2 + 600, y=(window_height - 480) // 2 + 200)
ai = Button(root, text="AI First",  width=12, height=1, bg='red', fg='white', font=('Arial', 20), command=lambda:switchMode(1))
ai.place(x=(window_width - 480) // 2 + 600, y=(window_height - 480) // 2 + 250)
hand = Label(root, text='      Move', font=('Arial', 40))
hand.place(x=(window_width - 480) // 2 + 600, y=(window_height - 480) // 2 + 400)
pos = Label(root, text='0', font=('Arial', 40))
pos.place(x=(window_width - 480) // 2 + 600, y=(window_height - 480) // 2 + 400)
```

圖 2-5-2：模式控制

```
wnum = Label(root, text="2", borderwidth=3, fg='blue', font=('Arial', 50))
wnum.place(x=(window_width - 480) // 2 + 600, y=(window_height - 480) // 2 - 50)
ver = Label(root, text=':', font=('Arial', 50))
ver.place(x=(window_width - 480) // 2 + 700, y=(window_height - 480) // 2 - 50)
bnum = Label(root, text="2", borderwidth=3, fg='red', font=('Arial', 50))
bnum.place(x=(window_width - 480) // 2 + 780, y=(window_height - 480) // 2 - 50)
```

圖 2-5-3：比數顯示

```
# Calculate AI time
aititle = Label(root, text='AI Time', font=('Arial', 30))
aititle.place(x=(window_width - 480) // 2 - 25, y=(window_height - 480) // 2 - 150)
AItime = Label(root, text="0", width=12, borderwidth=3, bg='black', fg='white', font=('Arial', 30))
AItime.place(x=(window_width - 480) // 2 - 100, y=(window_height - 480) // 2 - 100)
totaltitle = Label(root, text='Total Time', font=('Arial', 30))
totaltitle.place(x=(window_width - 480) // 2 + 350, y=(window_height - 480) // 2 - 150)
Totaltime = Label(root, text="0", width=12, borderwidth=3, bg='black', fg='white', font=('Arial', 30))
Totaltime.place(x=(window_width - 480) // 2 + 300, y=(window_height - 480) // 2 - 100)
```

圖 2-5-4：時間顯示

```
# Note positions & pass
notelabel = Label(root, text="Note", font=('Arial', 50, 'bold'))
notelabel.place(x=(window_width - 480) // 2 - 340, y=(window_height - 480) // 2 - 50)
note_frame = Frame(root)
note_frame.place(x=(window_width - 480) // 2 - 470, y=(window_height - 480) // 2 + 50)
canvas = Canvas(note_frame, bg="#dddddd", width=420, height=400, scrollregion=(0, 0, 1000, 1000))
scrollbar = Scrollbar(note_frame, orient="vertical", command=canvas.yview)
scrollable_frame = Frame(canvas)
canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
canvas.configure(yscrollcommand=scrollbar.set)
note_text = StringVar()
note_text_widget = Label(scrollable_frame, bg="#dddddd", textvariable=note_text, font=('Arial', 18))
note_text_widget.pack()
note_text.trace_add("write", lambda *args: update_scrollregion())
canvas.pack(side="left", fill="both", expand=True)
scrollbar.pack(side="right", fill="y")
```

圖 2-5-5：座標紀錄

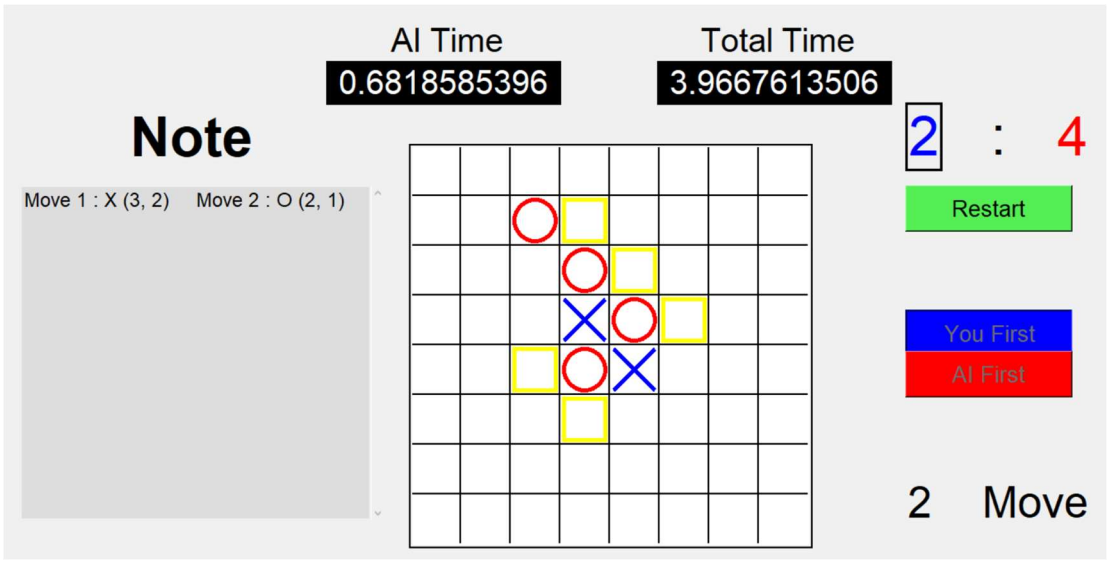

圖 2-5-6：介面呈現

參、競賽結果

＜預賽＞

比賽時間：2024/05/16

比賽對手：第 22 組

組別	第 8 組	第 22 組
對戰比分	24	40
花費時間	114.3s	3099998ns
排名	第二名	第一名

比賽結果：以組內第二名晉級八強賽

＜複賽＞

比賽時間：2024/05/28

比賽對手：第 4 組

組別	第 8 組	第 4 組
對戰分數（分）	15	49
花費時間（秒）	139.1s	76.7s
排名	第二名	第一名

比賽結果：止步十六強賽

肆、心得討論

組員姓名	心得感想
宋昭佑	這個專案是黑白棋，只是老師把黑白棋的規則做了微調，在最後的勝負規則由場上多子的勝，改成場上少子的贏，也就是和原規則相反。這樣造成的改變主要是下棋策略的部分。我們參考了一些GITHUB上相關資料並自行製作此次專案，我們針對黑白棋的規則修改場上每個位置的權重，並且新增 Alpha-Beta 剪枝法來協助我們的 AI 來進行最佳移動的計算，在正常的Alpha-Beta剪枝法中，分成權重越高越好和權重越低越好，權重高代表對白棋越有利，權重越低代表對黑棋越有利，我們使用的 DEPTH 是 5，也就是預測未來五步的最好結果來選擇最佳走法，但是因為規則改成相反的了，所以白旗反而是要找 MIN 權重的值來走，也就是原先黑棋的最佳路徑，黑色的走尋找方法也是相反，並且加入 Alpha、Beta 這兩個變數來做修剪的工作，來省去多餘的運算，達到節省運算資源時間的效果，這是我們在決定每一步棋路的演算法裡面最重要的一環。做完這個專案讓我對人工智慧有了更宏觀的認識，原本自己對於人工智慧的了解僅僅是一些名詞的堆疊，像是 CNN、DATA TRINING 等等，好像丟資料給他訓練才是人工智慧，但是這個專案裡我們用到的比較像是演算法，利用路徑的決策樹來達到最佳走法，沒有事先 TRAINING 或是記憶任何資料在程式中，單純依靠電腦的強大運算能力來進行推論，所以這堂課讓我對人工智慧有哪部分更深的認識?我會說是除了學習以及採取行動決，強大的演算法來做推論也是人工智慧重要的一環。
黃子庭	起初在聽到是黑白棋規則時，我感到非常慌張，因為自己是完全沒有下過棋，且根據教授提供的資源去做試玩都一竅不通的人，所幸在組員討論期間我也一步步開始理解甚至可以幫忙偵錯，這是對我來說在本次專題中相當大的成長。而這本次專題中我主要還是負責做操作介面設計，起初在查找資料時，大多數的資源都是 javascript 的撰寫方式，而我看到組員在進行演算法時使用了 python，因此決定統一 python 以方便大家進行溝通，製作了雖然不太華麗但淺顯易懂的介面。在製作過程中，出現的問題大多數是圖形顛倒或先後手

	顛倒的狀況，就如我前面所述，在黑白棋規則上我花了很多時間，原先想直接使用黑白棋顯示，但看到教授給的範例而使用ＯＸ做替代，也因此搞錯了圖形表示的內容。 在進行函式及演算法撰寫時，我覺得這是我最無力的地方，因為我的電腦不如他人的快速，因此在測試及偵錯時花了比別人多的時間，也因此我認知到人工智慧並不如理論中所述的簡單明瞭，而是需要許多資源去支撐及建構，這可以算是給這堂課畫下了一個意義中大的句號。
張祐瑜	當初聽到要做此專案時，我是相當感興趣的，之前就有聽說過修過這堂課的同學分享對戰的過程等等，所以就一直很期待製作專案，但是當看到專案題目有點手足無措，因為我本人是沒有接觸過黑白棋，所以對於黑白棋規則是非常不熟悉，這也導致我在最初編寫黑白棋規則的程式碼時相當頭疼，後來也是經過多方搜尋資料後才大致上明白黑白棋真正的玩法，以及其中的有趣之處，也是在查詢完資料後我才決定如何設定棋盤的權重值，我發現當在正常玩法中，下在棋盤四個角落才是最好的地方時，我就將棋盤四個角落的權重值設為最低，其他部分的權重值也是透過相同的道理去進行設定。此外，在撰寫程式碼的過程中，我覺得在撰寫switchMode()函式和 reversi()函式比較麻煩，因為在撰寫程式時腦袋一直在思考到底目前下棋方是誰，reversi()函式應該要翻轉哪一方的棋子，最麻煩的是如何判定pass和結束遊戲，後來經過思考之後決定透過對雙方都互相檢查一遍的方式來實作這兩個功能。我覺得經過製作這次的專案讓我了解到人工智慧是如何做出最佳決策的，並且權重值的設定對於未來的結果是非常的重要，即便演算法再遞迴更多層，如果當初權重值本身就是設定錯的，那最終結果肯定也是錯的。
廖洺玄	這次期末專題是要做黑白棋（Othello、Reversi）的變體，將原有的勝利條件從多子數玩家贏改為少子數玩家贏。我覺得這在設計 Alpha-Beta 和 Heuristic Value Estimation 上需要下一點功夫，因為要從原本的規則改過來看似輕鬆，但實際上要克服的難點還是有的。首先，在 Alpha-Beta 剪枝演算法上，傳統的設計是為了最大化己方棋子的數量，但在新的規則下，我們需要將目標改為最小化己方棋子的數量。這意味著我們需要重新評估每一步棋的價值，並調整剪枝策

	略，以確保它們仍然有效。在 Heuristic Value Estimation（啟發式估值）方面，挑戰在於如何重新定義估值函數。在原有規則下，估值函數通常會根據棋盤上己方與對方棋子的比例來決定，而現在我們需要設計一個新的估值模型，可能需要考慮到更多的棋盤控制和潛在翻轉的局面。這種改變不僅涉及數學模型的調整，還需要在實際測試中不斷調整和優化，以找到最佳的估值方法。加上期末的競賽也要比速度，所以在演算法的效能方面也要進行近一步的調整。這意味著我們需要在正確性和效率之間找到平衡點。除了傳統的 Alpha-Beta 剪枝技術，可能還需要引入一些進階的優化技術，以提升搜尋速度和減少重複計算。這次的專題的挑戰不僅在原有的規則進行改變，甚至是於如何在新規則下保持或提升演算法的速度。這讓我們在設計和實施過程中花費許多時間不斷試驗和調整，並在實際競賽中驗證我們的解決方案。通過這次專題我發現我提升自己在演算法設計和問題解決方面的能力，並為未來的研究打下堅實的基礎。

伍、組員貢獻度

組員姓名	負責工作	貢獻度
宋昭佑	隊伍協調、程式測試	22%
黃子庭	介面設計、程式偵錯	26%
張祐瑜	黑白棋規則程式開發	26%
廖洺玄	AI 演算法開發	26%