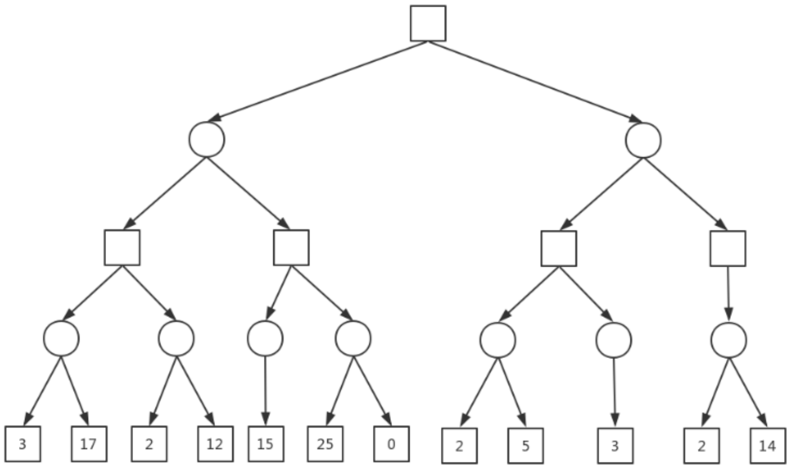
Lab 3: Python GUI Programming Report

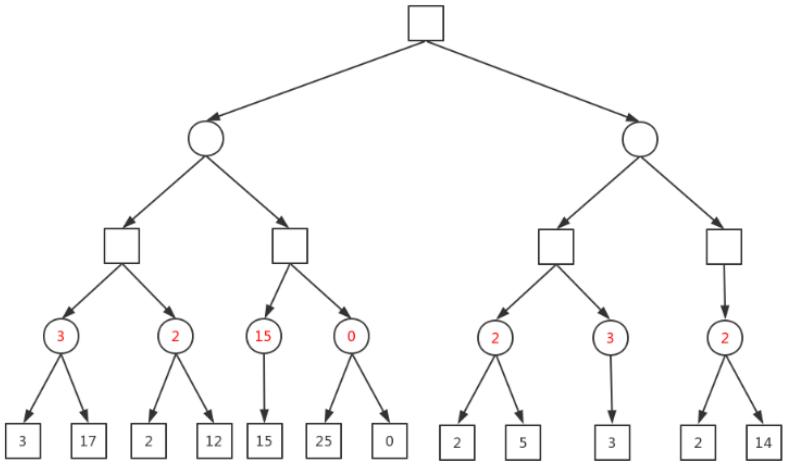
學號： 109511207 姓名： 蔡宗儒

1. 若要設計一個電腦玩家與你對抗，你會如何設計，請簡述你的演算法

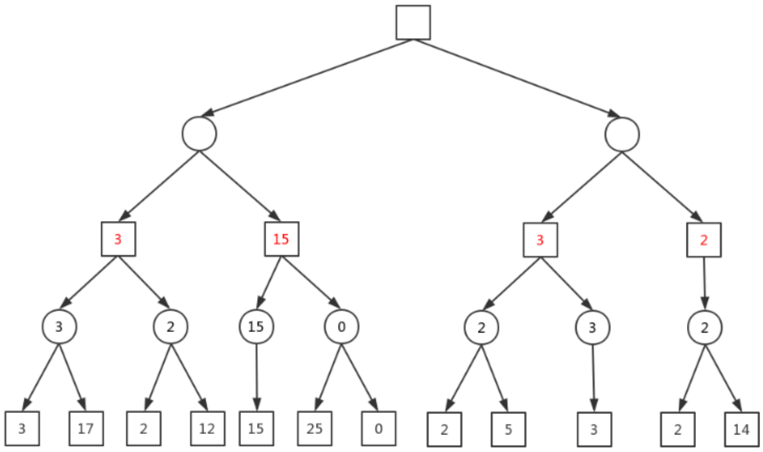
可以使用 MinMax 演算法，這是一種尋找在最大可能失敗性中的最小值的演算法。這種演算法通常應用於棋類遊戲，其中兩個玩家交替下棋。先手希望下一步的局面使自己的勝算最大化，而後手則希望下一步的局面使先手的勝算最小化。



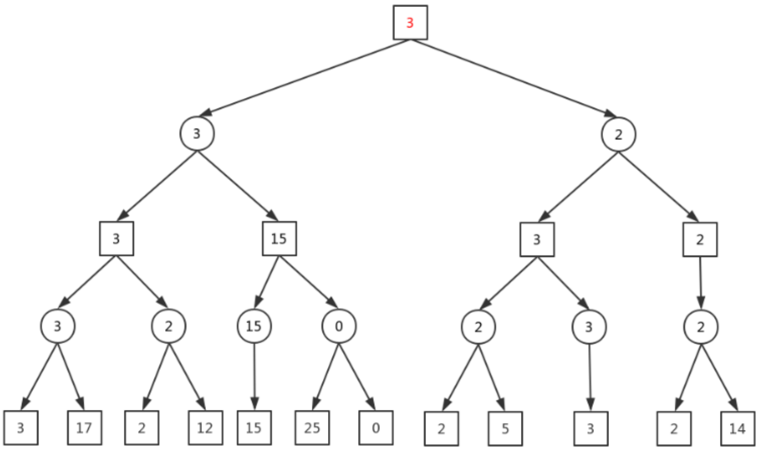
以一個博奕樹的例子來說，其中正方形代表先手，圓形代表後手。首先，先手應該計算後手在第四步時會選擇價值最小的局面，即在所有子節點中選擇最小值，如下圖紅字。



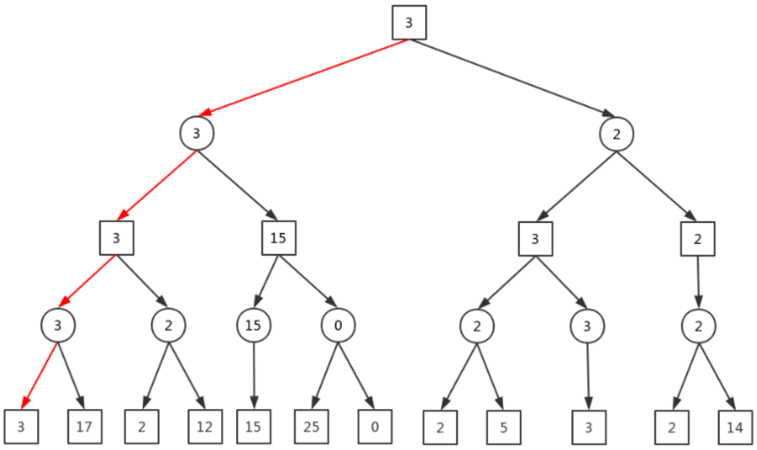
接著，先手計算自己在第三步時該如何選擇，即在所有子節點中選擇最大值，如下圖紅字。



最後先手可以根據所有子節點中的最大值來決定下一步應該怎麼走，如下圖紅字。



因此，如果先手和後手都做出最佳決策，棋局的走向將按照最大值的方向進行，如下圖紅線。



然而如果後手未做出最佳決策，則棋局的走向將不同。MinMax 演算法的計算複雜度隨著步數的增加呈指數級成長，但其中包含了許多不必要的狀態。因此可以考慮使用 Alpha-Beta 剪枝演算法來提高效率。

2. 請貼上自己的程式碼並附上註解

```
# import package
import tkinter as tk
from tkinter import messagebox

# create board
def  create_board(window):

    # create a 3x3 board
    for i in range(3):
           for j in range(3):
              button = tk.Button(window, text="", font=("Arial", 50),
                                 height = 2, width = 6, bg = 'gray',
                                 command = lambda row = i, col = j: handle_click(row, col))
              button.grid(row = i, column = j, sticky = "nsew")
```

```
# handle button clicks
def handle_click(row, col):
        global current_player
        global board

        # check which button has been clicked and change player
        if board[row][col] == 0:

            # 1 means 'O' & 2 means 'X'
            if current_player == 1:
                board[row][col] = 1
                current_player = 2
            else:
                board[row][col] = 2
                current_player = 1

            # change text on button to show 'O' or 'X'
            button = window.grid_slaves(row = row, column = col)[0]
            if board[row][col] == 1:
                button.config(text = 'O', bg = 'gray')
            else:
                button.config(text = 'X', bg = 'gray')
            check_winner()

# check for a winner or a tie
def check_winner():
    winner = None
    winner_path = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

    # check rows
    for i in range(3):
        # if there is a row connected as a line, store this row in the winner_path
        if board[i][0] == board[i][1] and board[i][0] == board[i][2] and board[i][0] != 0 :
            winner = board[i][0]
            winner_path[i][0] = 1
            winner_path[i][1] = 1
            winner_path[i][2] = 1

    # check columns
    for i in range(3):
        # if there is a column connected as a line, store this column in the winner_path
        if board[0][i] == board[1][i] and board[0][i] == board[2][i] and board[0][i] != 0 :
            winner = board[0][i]
            winner_path[0][i] = 1
            winner_path[1][i] = 1
            winner_path[2][i] = 1

    # check diagnoals
    # if there is a diagnoal connected as a line, store this diagnoal in the winner_path
    if board[0][0] == board[1][1] and board[0][0] == board[2][2] and board[0][0] != 0 :
        winner = board[0][0]
        winner_path[0][0] = 1
        winner_path[1][1] = 1
        winner_path[2][2] = 1
    if board[0][2] == board[1][1] and board[0][2] == board[2][0] and board[0][2] != 0 :
        winner = board[0][2]
        winner_path[0][2] = 1
        winner_path[1][1] = 1
        winner_path[2][0] = 1

    # check if tie
    # if every board is filled with values but no lines are connected, it results in a tie
    if winner == None and not(0 in board[0]) and not(0 in board[1]) and not (0 in board[2]):
        winner = "tie"

    # if there is a winner or a tie
    if winner:
        declare_winner(winner, winner_path)

# declare the winner and ask to restart the game
def declare_winner(winner, winner_path):
    if winner == "tie":

        # output message
        message = "It's a tie! Do you want to restart the game?"

        # refresh board color
        for i in range(3):
                for j in range(3):
                    button = window.grid_slaves(row = i, column = j)[0]
                    button.config(bg = 'red')
    else:
        # output message
        if winner == 1:
            message = "Player O wins! Do you want to restart the game?"
        elif winner == 2:
            message = "Player X wins! Do you want to restart the game?"

        # refresh board color
        for i in range(3):
                for j in range(3):
                    if winner_path[i][j] == 1 :
                        button = window.grid_slaves(row = i, column = j)[0]
                        button.config(bg = 'blue')

    # ask if the player wants to continuing
    answer = tk.messagebox.askyesno(title = "Game Over", message = message)

    if answer:

        # player another round
        global board
        board = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

        # reset the board
        for i in range(3):
            for j in range(3):
                button = window.grid_slaves(row = i, column = j)[0]
                button.config(text = "", bg = 'gray')

        # reset player
        global current_player
        current_player = 1

    else:

        # destroy window
        window.destroy()


if __name__ == '__main__':

    # create main window
    window = tk.Tk()
    window.title("Lab3 Tic-Tac-Toe")
    # create game board
    create_board(window)

    # initial variables
    board = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
    current_player = 1

    window.mainloop()
```

3. 心得

這次 Lab 實作的項目從 android studio 變成 python，自己寫一個圈圈叉叉的小遊戲出來非常的有趣，而我也大致了解到了一般的棋類遊戲背後的相關演算法像是 MinMax 演算法和 Alpha-Beta 修剪法為何。

4. Reference

[1] 最清晰易懂的 MinMax 算法和 Alpha-Beta 剪枝详解

[2] 電腦下棋的關鍵： Min-Max 對局搜尋與 Alpha-Beta 修剪算法