Lab 3: Python GUI Programming Report

學號：109511286　　姓名：蔡佩蓉

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

一、 基於遊戲規則的策略

在井字棋（tic-tac-toe）中，玩家需要用任兩個不同的符號標記九個空格。Table 1 給出了井字棋中玩家獲勝的五種基於規則的策略。

Table 1. 井字棋規則

規則	說明
1	如果玩家有獲勝的棋步，就採取它。
2	如果對手有必勝的棋步，則阻止它。
3	如果玩家可以在這步棋後創建一個分叉（兩種獲勝方式），則採取它。
4	在玩家移動後，不要讓對手可以創建分叉。
5	以玩家可以贏得最多可能方式獲勝的方式移動。

二、 極小化極大化（Minimax）搜尋演算法

Minimax 演算法是決策論和賽局理論中使用的一種回溯演算法。它使用博弈論、決策論、統計學和哲學來找到玩家的最佳行動，假設對手也採取最佳行動。它通常用於兩人回合製遊戲，如井字遊戲、國際象棋等。

玩家需要滿足兩個條件才能贏得遊戲。首先，玩家需要最大化贏得比賽的機會。其次，玩家需要盡量減少對手獲勝的機會。Minimax 演算法的原理是尋找最優路徑，使最大可能損失最小化。兩個可能的結果，+∞表示計算機獲勝，-∞表示計算機失敗。

Minimax 演算法的步驟總結如下。

a. 建構一個完整的競賽樹（complete game tree）
b. 使用 evaluation function 評估 leaves 的分數
c. 考慮玩家類型，備份從 leaves 到 root 的分數：
   - 對於 max 玩家，選擇得分最高的 child
   - 對於 min 玩家，選擇得分最低的 child
d. 在 root node，選擇數值最大的節點，進行對應的移動

Figure 1 說明了極小化極大化演算法，其中它使用啟發式評估函數（heuristic evaluation function）評估 leaf 節點（終止節點或 max

depth 為 4）。演算法繼續交替評估子節點（child）的最大值和最小值，直到到達 root 節點，並在 root 節點選擇最大值。

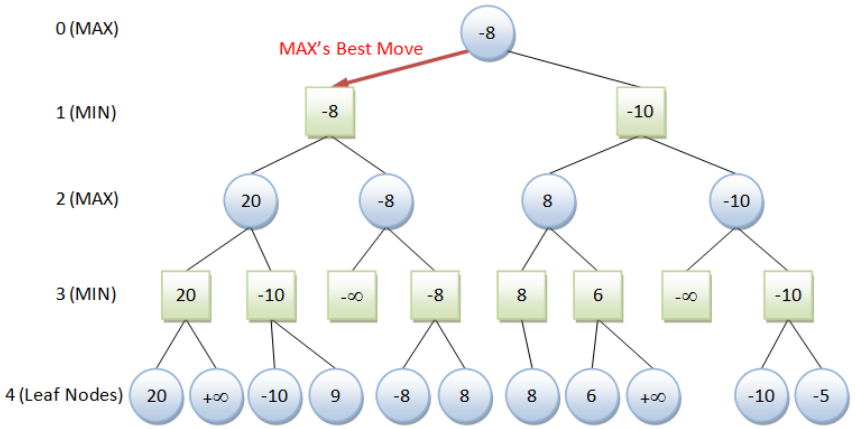

Figure 1. 極小化極大化演算法示意圖

三、 Alpha-Beta 剪枝（Pruning）

Alpha-Beta 剪枝演算法是極小化極大化演算法的優化演算法。它大大減少了計算時間。它允許更快的搜索，甚至可以進入競賽樹的更深層。當演算法評估出某策略的後續走法比之前策略的還差時，就會停止計算該策略的後續發展（剪斷不影響最終決定的競賽樹分支）。

Alpha-beta 剪枝旨在減少搜尋樹中需要透過極小化極大化演算法評估的節點數量。如 Figure 2 所示的 alpha cut-off，節點 C (MIN)不能大於 1，因為節點 D 回傳 1。由於節點 B 的值為 4，因此節點 C 的剩餘子節點不需要再搜尋（剪斷），因為節點 A 肯定會選擇節點 B。

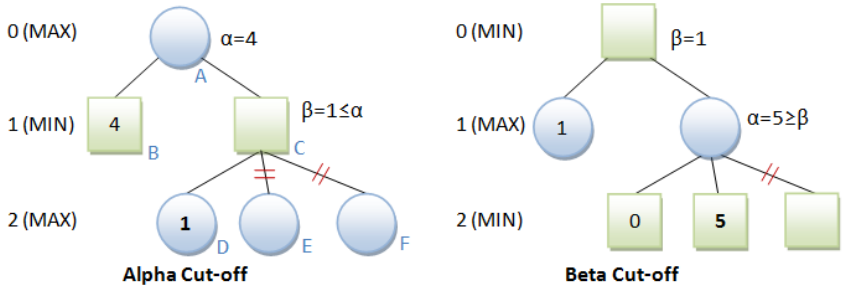

Figure 2. Alpha-Beta 剪枝演算法示意圖

在這個演算法中，需要兩個參數：一個 alpha 值，它保存為 MAX 節點找到的最佳 MAX 值；以及一個 beta 值，它保存為 MIN 節點找到的最佳 MIN 值。如圖所示，對於 alpha cut-off 和 beta cut-off，如果

alpha ≥ beta，則可以中止搜尋剩餘的子節點。Alpha 和 beta 分別初始化為-∞和+∞。

四、 實現策略

a. 針對目前遊戲狀態，需要一個評估函數來評估每個可能的移動。這個函數可以考慮當前局面的優勢劣勢，並預測未來可能的狀態等因素。
b. 使用極小化極大化演算法來搜尋最優解。這需要遞歸搜尋遊戲樹，評估每個可能的狀態。優化極小化極大化演算法（Alpha-Beta 剪枝等技術），提高搜尋效率，減少需要評估的狀態數量。
c. 允許調整電腦玩家的難度級別，例如在評估函數或搜尋深度上進行調整，使得遊戲更具挑戰性。

另外，我們可以使用機器學習方法，例如強化學習，讓電腦玩家透過與玩家對戰不斷學習最佳化策略。

2. 請貼上自己的程式碼並附上註解

```
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 22 15:40:41 2023

@author: 109511286
"""

import tkinter as tk
from tkinter import messagebox

# Create board
def create_board():
    for i in range(3):
        for j in range(3):
            # 建出button
            button = tk.Button(window, text="", font=("Arial", 50), height=2, width=6,
                               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

    # Check which button has been clicked and change player
    if board[row][col] == 0:
        if current_player == 1:
            # 設定board的值
            board[row][col] = "X"
            current_player = 2
        else:
            # 設定board的值
            board[row][col] = "O"
            current_player = 1

        # 設定特定row和column的button
        button = window.grid_slaves(row=row, column=col)[0]
        button.config(text=board[row][col])

        check_winner()
```

```python
# Check for a winner or a tie 檢查遊戲使否結束
def check_winner():
    winner = None

    winner_path = []

    # Check rows (檢查是否有row連成一線)
    for row in board:
        if row.count(row[0]) == len(row) and row[0] != 0:
            winner = row[0]
            winner_path = [(i, board.index(row)) for i in range(len(row))]
            break

    # Check columns (檢查是否有column連成一線)
    for col in range(len(board)):
        if board[0][col] == board[1][col] == board[2][col] and board[0][col] != 0:
            winner = board[0][col]
            winner_path = [(col, i) for i in range(len(board))]
            break

    # Check diagonals (檢查對角線是否連成一線)
    if board[0][0] == board[1][1] == board[2][2] and board[0][0] != 0:
        winner = board[0][0]
        winner_path = [(i, i) for i in range(len(board))]

    elif board[0][2] == board[1][1] == board[2][0] and board[0][2] != 0:
        winner = board[0][2]
        winner_path = [(i, len(board) - 1 - i) for i in range(len(board))]

    # Check if tie (檢查是否平手)
    if all([all(row) for row in board]) and winner is None:
        declare_winner("tie", [])

    if winner:
        declare_winner(winner, winner_path)


# Declare the winner and ask to restart the game
def declare_winner(winner, winner_path):
    # 遊戲平手、整個board變紅色
    if winner == "tie":
        message = "It's a tie!"
        # Change all button colors
        for i in range(3):
            for j in range(3):
                button = window.grid_slaves(row=i, column=j)[0]
                button.config(bg="indianred1")
    # 遊戲有勝負、標出勝利者的連線
    else:
        message = f"Player {winner} wins!"
        # Change button colors for winning path
        for i in winner_path:
            button = window.grid_slaves(row=i[1], column=i[0])[0]
            button.config(bg="lightskyblue")
    # ask if the player wants to continue 視窗詢問玩家是否繼續新一輪遊戲
    answer = messagebox.askyesno("Game Over", message + " Do you want to restart the game?")

    if answer:
        # play another round
        global board
        board = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

        for i in range(3):
            for j in range(3):
                button = window.grid_slaves(row=i, column=j)[0]
                button.config(text="", bg=default_bg_color)

        global current_player
        current_player = 1

    else:
        # close window
        window.destroy()

if __name__ == '__main__':
    # create main window 創建主視窗
    window = tk.Tk()
    window.title("Lab3")

    # create game board
    create_board()
    default_bg_color = window.cget('bg')

    # Initialize variable
    board = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
    current_player = 1

    window.mainloop()
```

3. 心得

透過這次的實驗，我深刻了解了使用 Python 和 Tkinter 來實現簡單的井字棋，並實現了一個具有基本遊戲邏輯和 GUI 的應用。這次的實現主要涉及了 Tkinter 的視窗和按鈕操作，以及遊戲邏輯的設計。

在實施遊戲邏輯方面，我學到了如何用列表來表示遊戲棋盤，以及如何檢查遊戲的勝利條件，包括橫向、縱向和對角線。我還實現了遊戲的重新啟動功能，使得在遊戲結束後可以輕鬆開始新的一輪。

對於電腦玩家的實現，我了解到了 Minimax 演算法的應用，這是一種適用於兩人零和遊戲的算法，它通過反复遞歸搜索競賽樹，評估每個可能的移動，以找到最佳的決策。 Alpha-Beta 剪枝的概念也被引入以減少搜尋空間，提高演算法效率。

這次實作讓我更深入了解遊戲演算法的實際應用，並了解到如何在簡單的遊戲中實現一個具有基本智慧的電腦對手。同時，我也明白遊戲開發中 GUI 的重要性，以提供更好的用戶體驗。

總的來說，這個專案擴展了我的 Python 和 Tkinter 知識，並深化了我對遊戲算法和遊戲開發的理解。這是一次有趣又實用的經驗，讓我更有自信地應對未來涉及遊戲和 GUI 的程式開發挑戰。