Spike: Spike Week 4
Title: Graphs, Search & Rules

Author: Nguyen Khanh Toan - 104180605

Goals / deliverables:

The Tic-Tac-Toe game can be modified the game state as a graph. In addition, the game can involve Tic-Tac-Toe AI with random move or efficiency and effectiveness than move randomly.

Summarise from the spike plan goal
Besides this report, what else was created?
For example: UML diagram, code, reports

- Code see /spikes/spike04/
- Short report titled "ide comparison"
- …

Technologies, Tools, and Resources used:
List of information needed by someone trying to reproduce this work

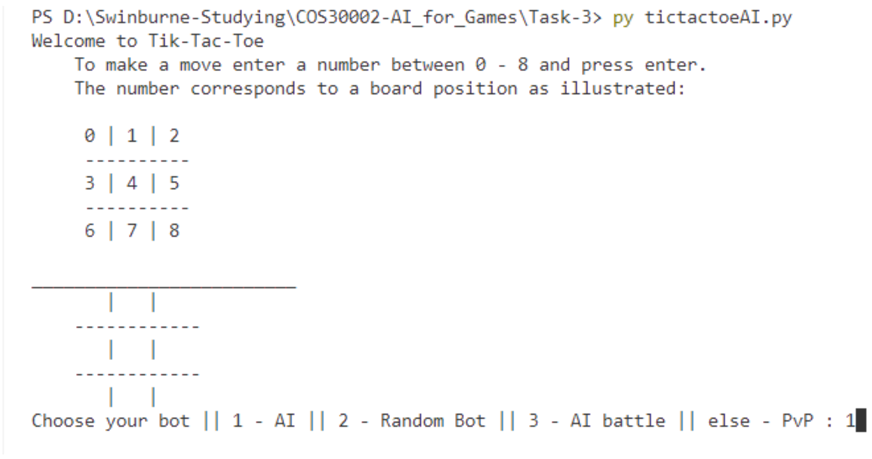
- Visual Studio Code
- Python 3.12

Tasks undertaken:

- Install Python 3+
- Install and setup compatible IDE for the language, e.g.: Visual Studio Code
- Pay attention to the comment of how the code work and functionality. Can use debug tool to observe the program more clearly.
- Run the code and observing the output.

What we found out:

- The users can choose to play with an Minimax AI, a random AI, or let the Minimax and random AI battle, and the option of playing normal Tictactoe with other player.

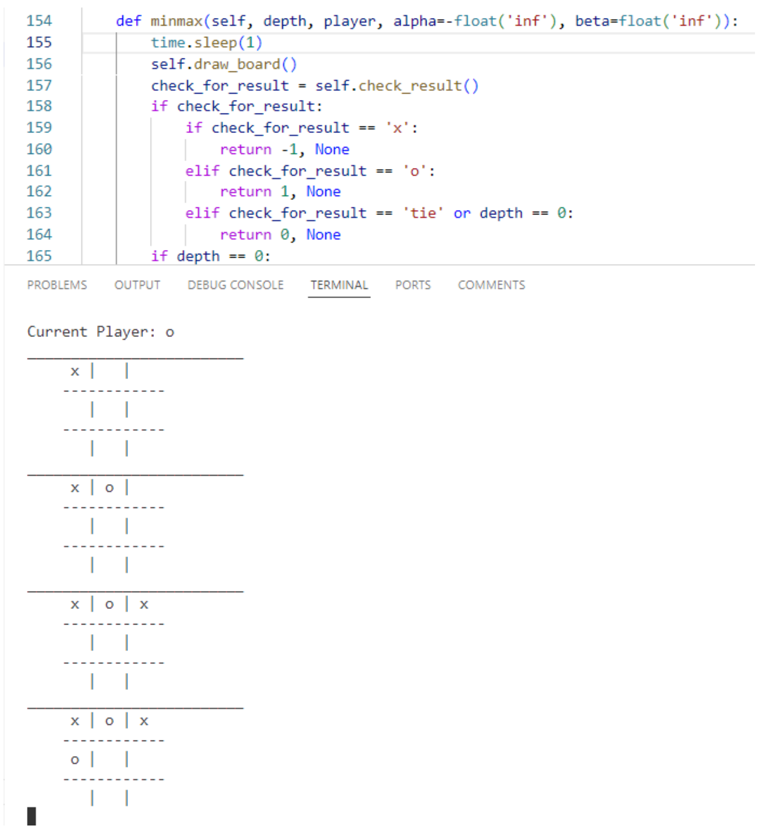


```
PS D:\Swinburne-Studying\COS30002-AI_for_Games\Task-3> py tictactoeAI.py
Welcome to Tik-Tac-Toe
    To make a move enter a number between 0 - 8 and press enter.
    The number corresponds to a board position as illustrated:

     0 | 1 | 2
     ----------
     3 | 4 | 5
     ----------
     6 | 7 | 8

_________________________
       |   |
    ------------
       |   |
    ------------
       |   |
Choose your bot || 1 - AI || 2 - Random Bot || 3 - AI battle || else - PvP : 1
```

- For the random bot, its move pretty “dumb” because it generates a random move in the possibles moves list. In addition, it has no “sensor” to observe what is going on in the world surrounding it, therefore, there is no calculation for the best move for the random bot to be action. This bot causing non-optimise in both efficiency and effectiveness.
- Minimax or Alpha-beta pruning AI can be a good solution for this game, by combining DFS to search all of the possible move that lead to win or at least draw state. For the Minimax, there might not be effective because it will search all of the branch trees (BFS the DFS =)))). However, alpha-beta pruning could solve this issue by cut off the branch not having better move than the best available move. This alpha-beta pruning can achieve optimizing in both efficiency and effectiveness.
- AI can be able to draw out the state that represent its move in respond to opponent move, and for the minimax bot, it can also draw out the state of opponent’s possible move.



```
    def minmax(self, depth, player, alpha=-float('inf'), beta=float('inf')):
        time.sleep(1)
        self.draw_board()
        check_for_result = self.check_result()
        if check_for_result:
            if check_for_result == 'x':
                return -1, None
            elif check_for_result == 'o':
                return 1, None
            elif check_for_result == 'tie' or depth == 0:
                return 0, None
        if depth == 0:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
Current Player: o
_________________________
     x |   |
    -----------
       |   |
    -----------
       |   |
_________________________
     x | o |
    -----------
       |   |
    -----------
       |   |
_________________________
     x | o | x
    -----------
       |   |
    -----------
       |   |
_________________________
     x | o | x
    -----------
     o |   |
    -----------
       |   |
```