## Stretch Task – Impress me!

I choose Impress me! as my stretch task because I thought I can make my agent more interesting to play against by choosing couple of options:

- Agent that never lose
- Agent that never win

I tried few options to make the algorithm for agent that never lose or never win, but unfortunately, I don't really know what other complex python command/function/syntax that I should use to make this interesting agent. So, I changed my plan, instead of making the agent more interesting, I want to impress people by making the Tic Tac Toe board more interesting. So, I changed the board from 3x3 squares to 4x4 squares instead. This idea was inspired by my son who likes to play 3x3 rubiks cube and now he's exploring the 4x4 which he found it harder.

I designed my agent similar to my "kennedy_agent" that I submitted for this fortnight 2 homework, which has the probability of wins higher than losses (61:33) so I kept it the same. To change the board however, I realised I have to update the engine.py file as well as the agent will play the game using the engine.py and interact with the Game Engine.

Below is the pseudo-code for the Tic Tac Toe game using 4x4 squares board:

## Algorithm and Pseudocode

1. Define the board as per below position:

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

2. Find out if you are playing as "X" or "O" by checking the number of how many X's and O's on the board. If it's equal number, then you are playing as "X", if not then you are playing as "O". If you're playing as "X" then you will make the first move.

3. To win the game, you need to have four of the same symbols in a row, column or diagonal (i.e., 4 of X's or 4 of O's on the same row/column/diagonal). So, you need to place your agent in any of these 10 position combinations:

| | | |
|---|---|---|
| a. | 0, 1, 2, 3 | =1st row (top-bottom) |
| b. | 4, 5, 6, 7 | = 2nd row (top-bottom) |
| c. | 8, 9, 10, 11 | = 3rd row (top-bottom) |
| d. | 12, 13, 14, 15 | = 4th row (top-bottom) |
| e. | 0, 4, 8, 12 | = 1st column (left-right) |
| f. | 1, 5, 9, 13 | = 2nd column (left-right) |
| g. | 2, 6, 10, 14 | = 3rd column (left-right) |
| h. | 3, 7, 11, 15 | = 4th column (left-right) |
| i. | 0, 5, 10, 15 | = Top-left to bottom-right diagonal |
| j. | 3, 6, 9, 12 | = Top-right to bottom-left diagonal |

4. Also, the same strategy to block the opponent from winning the game if your opponent already has 3 out of 4, you need to place your agent on the 4th position from any of these 10 possible combinations:
    a. 0, 1, 2, 3 =1st row (top-bottom)
    b. 4, 5, 6, 7 = 2nd row (top-bottom)
    c. 8, 9, 10, 11 = 3rd row (top-bottom)
    d. 12, 13, 14, 15 = 4th row (top-bottom)
    e. 0, 4, 8, 12 = 1st column (left-right)
    f. 1, 5, 9, 13 = 2nd column (left-right)
    g. 2, 6, 10, 14 = 3rd column (left-right)
    h. 3, 7, 11, 15 = 4th column (left-right)
    i. 0, 5, 10, 15 = Top-left to bottom-right diagonal
    j. 3, 6, 9, 12 = Top-right to bottom-left diagonal

5. If the winning move if possible then take the winning move first then follow by blocking the opponent from winning as the next priority.

6. Then next priority is taking any of the corner and middle positions as per below:
    a. If position #0 (top-left corner) is empty, place your agent here.
    b. If position #3 (top-right corner) is empty, place your agent here.
    c. If position #12 (bottom-left corner) is empty, place your agent here.
    d. If position #15 (bottom-right corner) is empty, place your agent here.
    e. If position #5 (middle: 2nd row and 2nd column) is empty, place your agent here.
    f. If position #6 (middle: 2nd row and 3rd column) is empty, place your agent here.
    g. If position #9 (middle: 3rd row and 2nd column) is empty, place your agent here.
    h. If position #10 (middle: 3rd row and 3rd column) is empty, place your agent here.

7. Otherwise, last priority is taking any available position on the board randomly.

## Python implementation

I rename the engine to engine_impress and update few things for the board for the following:

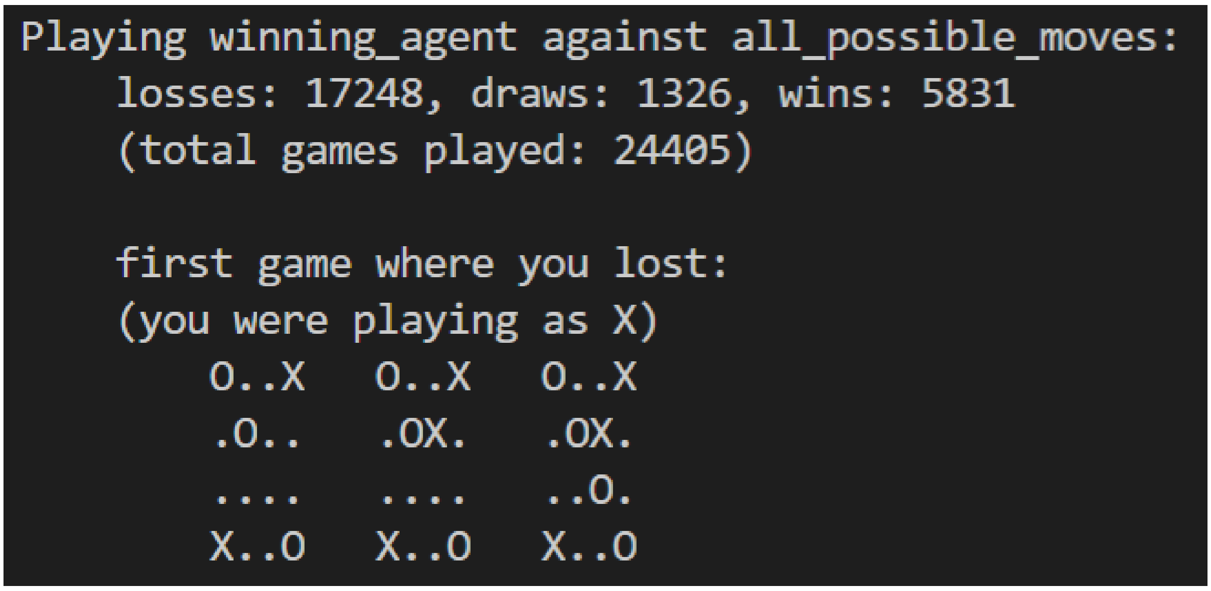
1.
   ```
   def winner(board: str):
   for a, b, c, d in (
           (0, 1, 2, 3),          # 1st row (top-bottom)       Here's the board:
           (4, 5, 6, 7),          # 2nd row (top-bottom)          0   1   2   3
           (8, 9, 10, 11),        # 3rd row (top-bottom)          4   5   6   7
           (12, 13, 14, 15),      # 4th row (top-bottom)          8   9   10  11
           (0, 4, 8, 12),         # 1st column (left-right)       12  13  14  15
           (1, 5, 9, 13),         # 2nd column (left-right)
           (2, 6, 10, 14),        # 3rd column (left-right)
           (3, 7, 11, 15),        # 4th column (left-right)
           (0, 5, 10, 15),        # top-left to bottom-right diagonal
           (3, 6, 9, 12),         # top-right to bottom-left diagonal
   ):
   ```

```
            if board[a] == board[b] == board[c] == board[d] != ".":
                    return board[a]
        return None
```

2. def check_valid_moves(board: str, moves: Tuple[str, ...]) -> None:
   ```
   if not len(board) == 16:
           raise ValueError(msg + "must have sixteen characters")
   ```
3. def engine(agent: GameAgent, *, opponent: GameAgent = all_possible_moves, print_report=True) -> MoveCache:
   ```
   queue: Deque[str] = deque(("................",) + opponent("................"))
   ```

I call my agent as "kennedy_impress" (please see the kennedy_impress.py for the details of the codes).

When I played this code, it took a long time for the output to come up on the terminal. I think it's because the total games played were more than 20,000 games! Even though my agent is losing more than winning as the result shown below, I was pretty impressed with the output because my agent can play on the 4x4 Tic Tac Toe board.



```
Playing winning_agent against all_possible_moves:
    losses: 17248, draws: 1326, wins: 5831
    (total games played: 24405)

    first game where you lost:
    (you were playing as X)
        O..X   O..X   O..X
        .O..   .OX.   .OX.
        ....   ....   ..O.
        X..O   X..O   X..O
```

## Feedback from games against human opponents

When I tried to play against my agent on a piece of paper, I realised that it's really hard to win the game for both of us. This is because we have to make so many moves and combinations to complete/win the game. So, I don't think many people will like to play this 4x4 Tic Tac Toe board, but I'm still impressed that my agent can play this new game!
Another interesting way of playing this Tic Tac Toe in future is maybe using the 3-D cubes (3x3x3) just like Rubiks cube (but I don't know how to program this code in python yet) 😊