Min Max with Alpha beata for Tic Tac Toe

Name : Pritam .R. Turalkar

Class :- BE - IT

Roll No:- 71

Subject :- IS Lab

DOP	DOA	Remark	Sign

* Min Max with Alpha Beta for Tic Tac Toe :

→ The goal af Tic - Tac - Toe is to be the first player to get three in a row on 3X3 grid.

→ "X" always goes first.

→ Players alternate placing 'X's and "O's" on board until either:

i) One player has three in a row horizontally, vertically or diagonally.

ii) All nine squares are filled.

→ Programmer created in 'WinnigStates' ~~va~~ named Set containing a list af all possible win conditions inside "Properties.py", if a player places 'X's or 'Os in any af the list, they are declared winner!

The winning states are:

WinningStates = ([0, 1, 2], [3, 4, 5], [6, 7, 8],
[0, 3, 6], [1, 4, 7], [2, 5, 8],
[0, 4, 8], [2, 4, 6]).

→ Programmer has created a dummy bot which chooses positions randomly as DummyBot.py. The GameBoard initializes the free spaces to None. (List af Nones).

→ Progsammer also created a minmax bot which uses Min Max Algorithm with Alpha Beta pruning to decide the best move.

→ The main.py starts by initialization of two objects ~~named~~ of MinMaxBot and DummyBot. The code ~~then~~ creates a variable Judge which called TicTacToeJudge, to which both objects are passed. The ~~Tic~~ TicTacToeJudge.py decides the winner.

→ Programmer also created a Helper method, Helper.py which gets the opponent's position to bot and gets the available moves to play. It imports properties.py mentioned earlier.

✱ Inputs : → No inputs from user.
(As both the bots, DummyBot and MinMaxBot play the game).

☆ Output : i) Winner Name which can be:
a) Bot One (MinMax Bot)
b) Bot Two (Dummy Bot).
c) Draw (When all positions are filled with no winner).

The winner is decided if the bot's position is in the set of list of WinningStates().

α **Analysis af claim by Programmer that it uses MinMax with alpha beta pruning.**

→ i) This claim comes from the Bestmove() method in MinMaxBot.py as it uses recussion to find the next best move.

ii) It starts by getting the winner() state and checks if the game already ended by comparing ~~at~~ the winner variable with Self.char, Self.Opponent or Draw state and returns 1, -1, 0 respectively.

iii) The method then starts a for loop ~~for~~ which iterates through all possible moves in the gameboard.

iv) After every move, the Bestmove() calls itself recursively to figure out next best move by the MiMaxBot.

v). The Bot then places the marker on best move and updates the Alpha, Beta variables.

vi) The Alpha, Beta variables are checked with Value and are updated accordingly. If Value is greater than Alpha, Alpha is assigned to value & if its lower than Beta, Beta's value is updated to Value.

Thus, the claim by Programmer that it uses MinMax with Alpha Beta Pruning is correct.

Outputs :

i) Bot one

```
['x' , '0' , 'x'
 '0' , '0' , None
 'x' , '0' , 'x']
```

ii) Bot one

```
['0' , 'x' , 'x'
 '0' , None , None
 '0' , None , 'x']
```

iii) Bot Two

```
['0' , 'x' , None
 '0' , 'x' , 'x'
 '0' , '0' , 'x']
```

ix) Bot Two

```
['x' , 'x' , 'x'
 'x' , '0' , '0'
 '0' , 'x' , '0']
```

v) Bot One

```
['x' , '0' , None
 '0' , 'x' , None
 'x' , '0' , 'x']
```

vi) Bot one

```
['0' , 'x' , X
  X    O   None
  X    O    O ]
```

vii) Draw

```
['0' , 'x' , 'x'
 'x' , '0' , '0'
 'x' , '0' , 'x']
```

viii) Bot one

```
['0' , 'x' , None
 None , 'x' , '0'
 None , 'X' , None]
```

ix) Bot Two

```
['0' , 'x' , 'x'
 'x' , 'x' , '0'
 'x' , '0' , '0']
```

x) Draw

```
['x' , '0' , 'x'
 '0' , '0' , 'x'
 'x' , 'x' , '0']
```