ith αβ prunning (Tic-Tac-Toe)

Zarrar Husain Zakir Husain.

IT

ISLAB)

* Min Max with Alpha-Beta for Tic-Tac-Toe :-

→ The goal of Tic-Tac-Toe is to be the first player to get three in a row on 3x3 grid.

→ "X" always goes first.

→ Players alternate playing Xs & Os a on board until either :

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

ii) All nine squares are filled.

→ Programmer created in 'winning state' named set containing a list of all possible win conditions inside "properties.py", if a player ~~plays~~ places Xs or Os in any of 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 game board initializes the free spaces to none. (List of Nones).

→ Programmers also created a minMax bot which uses Min Max Algorithm with Alpha Beta prunning to decide the best more.

→ The main.py starts by initialization of two object of minmaxBot & dummyBot. The code then creates a variable judge which is called TicTacToeJudge, to which both objects are passed. The TicTacToeJudge.py decides the winner.

→ Programmer also created a helper method, helper.py which gets the opponent's position to bot & gets the available moves to play, imports properties.py mentiond earlier.

* Inputs:

No inputs from user
(As both the bots, dummyBot & MinMaxBot play the game).

* Output:

Winner name, which can be:
↳ Bot One (MinMaxBot)
↳ Bot Two (DummyBot)
↳ Draw (When all positions are filled c̄ no winner)

The winner is decided if the bot's position is in the set of list of winning~~Bot~~States().

* Analysis of claim by Programmer that it uses MinMax with alpha beta prunning.

> This clam comes from the bestMove() method in minMaxBot.py as it ~~was~~ uses recursion to find the next best move.

> ~~It starts by getting the next best move~~

> It starts by getting the winner() state, & checks if the game already ended by comparing the winner available with self.char, self.Opponent or draw state & returns, 1, -1, ~~or~~ 0 respectively.

> The method ~~move; the~~ then starts a for loop which iterates through all possible moves in the gameboard.

> After every move, the bestmove() calls itself recursively to figure out next best move by the MinMaxBot.

> The Bot then places the marker on best move and updates the alpha, beta variables.

> The Alpha, Beta variables are checked with value and are updated accordingly. If value is greater than Alpha, Alpha is assignet to value & if it is lower than Beta, Beta's value is updated to value.

Thus, the claim by programmer that it uses minmax with alpha beta prunning is correct.

Outputs:-

i) Bot one

$$\begin{bmatrix} \text{'X'}, & \text{'O'}, & \text{'X'} \\ \text{'O'}, & \text{'O'}, & \text{none} \\ \text{'X'}, & \text{'O'}, & \text{'X'} \end{bmatrix}$$

ii) Bot one

$$\begin{bmatrix} \text{'O'}, & \text{'X'}, & \text{'X'} \\ \text{'O'}, & \text{none}, & \text{none} \\ \text{'O'}, & \text{none}, & \text{'X'} \end{bmatrix}$$

iii) Bot ~~one~~ Two

$$\begin{bmatrix} \text{'O'}, & \text{'X'}, & \text{none} \\ \text{'O'}, & \text{'X'}, & \text{'X'} \\ \text{'O'}, & \text{'O'}, & \text{'X'} \end{bmatrix}$$

iv) Bot Two:

$$\begin{bmatrix} \text{'X'}, & \text{'X'}, & \text{'X'} \\ \text{'X'}, & \text{'O'}, & \text{'O'} \\ \text{'O'}, & \text{'X'}, & \text{'O'} \end{bmatrix}$$

v) Bot one

$$\begin{bmatrix} \text{'X'}, & \text{'O'}, & \text{none} \\ \text{'O'}, & \text{'X'}, & \text{none} \\ \text{'X'}, & \text{'O'}, & \text{'X'} \end{bmatrix}$$

vi) Bot One

$$\begin{bmatrix} \text{'O'}, & \text{'X'}, & \text{'X'} \\ \text{'X'}, & \text{'O'}, & \text{none} \\ \text{'X'}, & \text{'O'}, & \text{'O'} \end{bmatrix}$$

vii) ~~B~~ Draw

$$\begin{bmatrix} \text{'O'}, & \text{'X'}, & \text{'X'} \\ \text{'X'}, & \text{'O'}, & \text{'O'} \\ \text{'X'}, & \text{'O'}, & \text{'X'} \end{bmatrix}$$

viii) Bot One

$$\begin{bmatrix} \text{'O'}, & \text{'X'}, & \text{none} \\ \text{none}, & \text{'X'}, & \text{'O'} \\ \text{none}, & \text{'X'}, & \text{none} \end{bmatrix}$$

ix) ~~Bot Two~~ Draw

$$\cancel{\begin{bmatrix} X & O & X \\ O & O & X \\ X & X & O \end{bmatrix}}$$

x) Bot Two:

$$\cancel{\begin{bmatrix} \text{'X'} & O \\ \text{'O'} & O \\ \text{'X'} & X \end{bmatrix}}$$

ix)

$$\begin{bmatrix} \text{'X'}, & \text{'O'}, & \text{'X'} \\ \text{'O'}, & \text{'O'}, & \text{'X'} \\ \text{'X'}, & \text{'X'}, & \text{'O'} \end{bmatrix}$$

~~Bot~~ Draw

x)

$$\begin{bmatrix} \text{'O'}, & \text{'X'}, & \text{'X'} \\ \text{'X'}, & \text{'X'}, & \text{'O'} \\ \text{'X'}, & \text{'O'}, & \text{'O'} \end{bmatrix}$$

Bot Two