ATCS 2021-2022
Ms. Namasivayam

Tic Tac Toe Minimax

Part III - Minimax

A) **Base Case**

The base case for minimax returns a score if the board is currently in an end state:

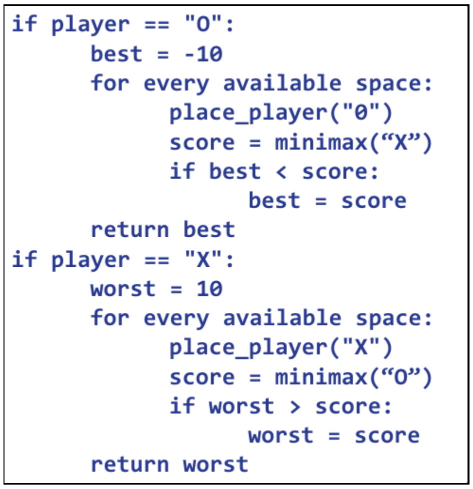
- 10 if player “O” won the game
- 0 if there is a tie
- -10 if “X” won the game

We’ve already created a series of functions that check if a certain player has won or tied in the game. Use those functions in minimax to determine which score should be returned.

B) **Recursive Case**

Now that we have the base case for minimax, it’s time to implement the recursive case.

As we learned, the recursive case is



```
if player == "O":
      best = -10
      for every available space:
            place_player("0")
            score = minimax(“X”)
            if best < score:
                  best = score
      return best
if player == "X":
      worst = 10
      for every available space:
            place_player("X")
            score = minimax(“O”)
            if worst > score:
                  worst = score
      return worst
```

In order to get this to work, you will need to do the following:

1. “for every available space” - create a for loop that iterates through all available rows and cols. place_player should be placing the correct player on an open row, col. You already created a place_player function.
2. Using if statements, set the return value equal to best and worst.
3. In our current version of minimax, the board is being changed every time that minimax is called in real time. Once minimax tries a move, we need the board to return to its original state. You can use place_player to re-add a “-” back to the row, col that we placed a hypothetical move.

C) **Getting the Row and Col values**

Right now `minimax` is returning just the score - instead it will return the score, the current row, and the current column. We can do this by returning a tuple that includes the best score coupled with the row and col it can be found at. For the base case, we can return the score, accompanied by the values `None` and `None` to indicate that there is no row, col to return.

Ex. `return (-10, None, None)`

Now, best and worst should be compared to the first index of any call to `minimax` because that contains the score.

Ex. `score = minimax("X")[0]`

Next, we need to store the value of the current optimal row and optimal column. Create two variables, `opt_row` and `opt_col` to store the best row/col pair and set their value to `-1` in the `minimax` function. When the value of best or worst is updated, these two variables should be updated as well. When the recursive case is being returned, the values returned should be:

```
return (best, opt_row, opt_col)
return (worst, opt_row, opt_col)
```

D) **Complete the Game with Minimax**

Now that we have a working version of minimax, let's add it to our original game!

1. Create a function `take_minimax_turn(self, player)`
   Here you will call `minimax` for the player. Since the value being returned is a tuple, you can store the score, row, and col directly as follows:

   `score, row, col = self.minimax(player)`

   The row and col being returned will be the optimal move, and should be used for the `place_player` function.

2. Modify `take_turn` to call `take_minimax_turn` when it is player "`O`"s turn