ARTIFICIAL INTELLIGENCE

Assignment # 3 - Report

21K-4523 Nabiha Rajani
BCS-6Z

I have made one samples.py file with the sample boards. All the other files are importing the same samples from there so that we can have a fair comparison.

I have assumed that ‘x’ goes first and, at the current state of the board, the AI plays for ‘x’.

```
samples = [
        [
            ['x', 'o', 'x'],
            ['o', '_', '_'],
            ['_', '_', '_']
        ],
        [
            ['o', '_', 'x'],
            ['_', 'x', '_'],
            ['o', '_', '_']
        ],
        [
            ['x', 'o', '_'],
            ['_', 'o', 'x'],
            ['_', '_', '_']
        ]
    ]
```

is_moves_left() returns true if there are any moves left

evaluate() returns 10 if the player has won, -10 if the opponent has won.

find_best_move() is called from the main function. Minimax is called for each empty position on the board which the player can mark as their own. The move that gives the highest score is maintained and the value for that score is assigned to alpha.

minimax() iterates over each position in the board and recursively calls minimax at each.

But in the alpha-beta pruning version, we maintain the highest alpha score (for max player i.e. agent) and the lowest beta score (for min player i.e. opponent) from the values returned from each instance of minimax called. These indicate the scores for best possible moves that either side can guarantee. Hence the loop breaks when beta is less than or equal to alpha, since we have better options we don't need to explore that branch.

	Without Alpha-Beta Pruning	With Alpha-Beta Pruning

No.of nodes evaluated	Run1	Run2	Run3	Average	Run1	Run2	Run3	Average
	233	185	205		102	128	161	

With alpha-beta pruning the number of evaluated nodes is greatly reduced.

	Parallelized Alpha-Beta Pruning				Serial Alpha-Beta Pruning			
No.of nodes evaluated	Run1	Run2	Run3	Average	Run1	Run2	Run3	Average
	179	164	187		102	128	161	
Time Taken (seconds)	0.00436	0.00362	0.00324		0.00066	0.00083	0.00078	

Parallelization takes more time and evaluates more nodes. The overhead cost is more than the benefit we get from load-balancing. The number of nodes increases even though we are not evaluating more branches, because some nodes are evaluated twice by both threads.

	Alpha-Beta Pruning				Enhanced Alpha-Beta Pruning			
No.of nodes evaluated	Run1	Run2	Run3	Average	Run1	Run2	Run3	Average
	102	128	161		47	56	61	

The heuristic I've used is simple; the function goes across the rows, columns and both diagonals and adds up the number of spots where the player and opponent can each place their mark in order to win, and returns the difference between them (player - opponent).

For the weak-player game, the agent interacts with the user. A random tile is generated which the AI player will always avoid. This is passed to weak_player_move() which will randomly pick a position besides that one. play_game() calls evaluate() and is_moves_left() and the loop (within which the moves are made and the board is continuously displayed) ends when there are no moves or a player has won.

Then the board is evaluated again to decide the winner and it is output.