CS365 Lab B Adversarial Games
Artificial Intelligence & Mach (Spring 2018)
Maniz Shrestha, Nirdesh Bhandari, Jeremy Swerdlow

For this lab we have implemented a Python code to play the game of Break-Through using different functions that we have designed. In our implementation, player 'O' gets the first move by default.

PART A

Representation scheme – For this lab, we have created a Python class object 'Board' which initializes the board and stores its different states to keep track of the environment. The board itself is stored in a list of lists. This Python class also stores whose turn it is, all the positions of 'X' pieces and 'O' pieces and the state of the board after each move. The 'Board' class contains other helpful functions such as 'possible_moves' , 'move_generator' , 'move', 'terminal_tests' and 'display state' to execute the game.

PART B

Utility Functions:

Evasive AI - The evasive AI does not actively go after the other player's pieces unless that is the only option. It will try to maximize its number of pieces therefore in each level it might choose to eat the opponent's piece if that is the only way to keep its piece. It usually spreads out its pieces and moves most of the pieces forward as opposed to forwarding only a single piece forward as was seen in the case of the Conqueror AI.

Conqueror AI - The Conqueror AI moves its pieces with the goal of capturing the other player's pawns. This AI often moves the same piece multiple times to get closer to and eat another player's piece. This AI often will move only one piece at a time as required to capture other pieces leaving the rest of the pieces behind.

A. When the Evasive AI's play against each other, each player focuses on losing the least amount of pieces and dodges the other player. When we ran tests in (5X5,1) and (8X8,2) boards, the winner was often the AI that dodged its way ahead. Even though the results were inconsistent across different board sizes, the common pattern seen was that Evasive AIs often move a lot of their pieces forward as opposed to the conqueror AI which often only moves individual pieces forward to capture the other player's pieces.

Our Utility Functions:

Collective_utility: This heuristic takes into account several things to calculate the utility of the board state. If the board is winning state, the utility is infinite. If the board state is almost about to win with the piece about to reach the end state not being attacked, it adds 50 to the utility. Similarly, it adds utility of 1 for every piece that is protected and subtract 1 for every piece that is being attacked. Further, this heuristic calculates the threat of each opponent piece and adds negative utility. The opponent pieces that are almost about to win (i.e. far ahead in the board) are marked as more dangerous than the ones that are behind.

Avg_dist_utility: This is the heuristic we designed. This calculates the average progression between how far our pieces have gotten and looks at the same distance between the opponents and maximizes the utility value. This heuristic decides to eat a player's piece if taking out the opponent's piece reduces their score and increases ours (if our piece gets closer). Further, this heuristic calculates the threat of each opponent piece and adds negative utility. The opponent piece that are almost about to win (i.e. far ahead in the board) are marked as more dangerous than the ones that are behind. Similarly, it adds utility of 1 for every piece that is protected and subtracts 1 for every piece that is being attacked.
Each of our functions took roughly 1.5 seconds to calculate the optimal move during each state.

Champions:
When we ran a 100 simulations of our Collective_utility and Avg_dis_utility against evasive and conqueror to get accurate values for testing. Out Collective_Utility function won against Evasive AI 90% of the time while it won 80% of the matches for Conqueror. Likewise the Avg_dis_utility function won 80% of the matches against Evasive AI and 75% of the matches against Conqueror AI.
When we ran a 100 simulations of Collective_utility against Avg_dis_utility function in a 5X5,2 board, the Collective_utility AI won 60% of the time. Upon closer inspection of the matches, we found that the player who made the first move often ended up winning the match. We believe that the weakness of the champion function was smaller boards with two rows of pieces where some pieces were able to slip through. However, out champion Collective_utility function played very well in larger board sizes.

Some Final Board States for different matches and moves made are presented in the table below:

Board Size (Row*Column)	**Player 'X' Utility Function**	**Player 'O' Utility Function**	**Final Board State**	**'X' Pieces Captured**	**'O' Pieces Captured**	**Total Moves Made (X+O)**

(5X5,1)	Evasive	Evasive	`.OX..`
			`X....`
			`..OXX`
			`.O...`
			`..OO.`
			
			`*O wins!*`
				1	0	6+7 = 13

(8X8,2)	Evasive	Evasive	`........`
			`..O.XXXX`
			`.XX....X`
			`X.O..XOX`
			`O.....XO`
				4	4	36+36= 72

			`OO..O.O.` `O....OO.` `....X...` **`*X wins!*`**			
(5X5,1)	Evasive	Conqueror	`.X.OX` `.....` `.X...` `...O.` `OO...` **`*O wins!*`**	2	1	5+6 = 11
(8X8,2)	Evasive	Conqueror	`X.X.X.X.` `..X...X.` `X..X...X` `......O.` `X......X` `.......O` `...O....` `...X.OOO` **`*X wins!*`**	4	10	34+34= 68
(5X5,1)	Conqueror	Conqueror	`..O..` `.....` `.X...` `.X...` `.....` **`*X wins!*`**	3	4	12+11 =23
(8X8,2)	Conqueror	Conqueror	`..XX..XX` `X.......` `........` `....X...` `.X......` `........` `.....X..` `........` **`*X wins!*`**	8	16	39+39 = 78

Some time measurements for large simulations:

Avg time per move: 2.013546668881118
collective vs. evasive: 1.0 0.0 -
Avg time per move: 1.7523705908919083
collective vs. conqueror: 0.8 0.2

Avg time per move: 1.0297829033283705
avg_dist vs. evasive: 0.78 0.22 –80%
Avg time per move: 0.9872959552419341

```
avg_dist vs. conqueror:  0.74 0.26
        Avg time per move:  1.0305195589065552
avg_dist vs. collective:  1.0 0.0

        Avg time per move:  1.3791549880164011
collective vs. avg_dist:  1.0 0.0 ---
```