ARTIFICIAL INTELLIGENCE (CSE 537) FALL-2015
ASSIGNMENT 2: CONNECT FOUR

Submitted by:
MOHIT GOYAL (110349552)
DEEPAK GOYAL (110347387)

INTRODUCTION:

In this assignment, we have implemented various game search algorithms for "connect four game" to make the game run more efficiently. The default game option is:

run_game(human_player, basic_player)

One need to uncomment lines to run other player games, like to play new player vs basic player, please uncomment line number 164 in lab3.py

run_game(new_player, basic_player)

Similarly all other options for play are given in lab3.py.

1. Minimax Search Algorithm:

Minimax search algorithm has been implemented for Connect Four game. For each move other player makes, one with maximum value is used by minimax player. This procedure is repeated until the game is over (win, lose or tie) or the tree depth has been explored.

Depending upon the basic evaluation function or new evaluation function we are calculating the nodes expanded by our minimax search algorithm and time has been calculated of the game.
Following is the screenshot attached of game played using minimax search algorithm.

basic player vs. basic player:

```
Player 2 (O) puts a token in column 6
It's a tie!  No winner is declared.

  0 1 2 3 4 5 6
0 O X X X O O O
1 X X O O O X X
2 O O X X X O O
3 X X O O O X X
4 O O O X X O O
5 X O X X X O X

basic player Nodes expanded = 45346
Total Nodes expanded = 45346
Running Time: 43.0739998817
```

human player vs. basic player:

```
Player 2 (O) puts a token in column 0
Win for O!

  0 1 2 3 4 5 6
0       O
1     O X   O
2     O O O X
3 O O O O X X
4 X O X O X X X
5 X X O X X O X

basic player Nodes expanded = 23084
Total Nodes expanded = 23084
Running Time: 96.881000042
```

2. **Better Evaluation Function:**
   Basic computer player was using simple basic evaluation function to check its chance of winning or losing. We have written a new player which is much better than basic player and to calculate its winning or losing chance, it uses new_evaluate function which calculates winning position by checking for every cell of board in a current state where it can make continuous four X's in all possible directions and it also checks for every cell of the board for the opponents winning chance (which will be losing chance for it). In our evaluation function we are also checking the position whether it is in middle of board as there is maximum chance of winning there.
   Following is the screenshot of game played using new evaluation function.

   new player vs. basic player:

```
Player 1 (X) puts a token in column 0
Win for X!

  0 1 2 3 4 5 6
0   X O O
1   O O X X
2   O X X X O
3   X O O O X
4 X X O X X O
5 O O X X X O

basic player Nodes expanded = 22744
new player Nodes expanded = 23642
Total Nodes expanded = 46386
Running Time: 48.1390001774
```

3. **Alpha Beta Pruning:**
   In our minimax search algorithm it was calculating the winning or losing chance of it by expanding the nodes and using the maximum value of evaluation function to run its move. But the major drawback of minimax algorithm is that it will expand all the nodes and which will increase the time to make the next move of the computer player also the number of nodes expanded will be very much. In order to make the minimax search algorithm more efficient, we have implemented alpha beta pruning algorithm. So to increase efficiency, In this algorithm, we will not expand the nodes which have less value than maximum value returned by other nodes at the same level. These nodes will get pruned and the efficiency of our search algorithm increase many fold.
   Following is the screenshot attached of alpha beta pruning algorithm which shows nodes expanded and running time of game.

   As we can see that in alpha beta pruning the number of nodes expanded and running time of the game decreases.

   alpha beta player vs. basic player:

```
Player 1 (X) puts a token in column 0
Win for X!

  0 1 2 3 4 5 6
0   X O O
1   O O X X
2   O X X X O
3   X O O O X
4 X X O X X O
5 O O X X X O

basic player Nodes expanded = 22744
alpha beta Nodes expanded = 8181
Total Nodes expanded = 30925
Running Time: 31.1610000134
```

4. **Connect-K Problem:**
   We have generalized the connect-four game to connect-k game. Now, it can take the k value (e.g. 3, 4, 5…) and instead of checking for 4 continuous tokens, it will check for k continuous tokens.
   Following are the screenshot of connect-k game.

E.g. Connect-5 game:
K=5
run_game(alphabeta_player, basic_player, ConnectFourBoard(K))

```
Player 2 (O) puts a token in column 6
It's a tie!  No winner is declared.

  0 1 2 3 4 5 6
0 X O X O X O O
1 O X O O O X X
2 X O X X X O O
3 X O X O O X X
4 O X X X X O O
5 X X O O O O X

basic player Nodes expanded = 26800
alpha beta Nodes expanded = 9661
Total Nodes expanded = 36461
Running Time: 42.2179999352
```

5. **Longest Streak to Win:**
   In the longest streak to win the game will not stop at any fixed number k. Now the game will stop after 20 rounds i.e. each player will play 10 moves and after 20 rounds the player with longest streak in any direction horizontal, vertical or diagonal is the winner. And if both players have same maximum number of continuous tokens then the game is tie.

   <u>Note:</u> Please uncomment the lines from 535 to 546 in connectfour.py to play longest streak to win game.

   Following is the screenshot of longest streak to win:

```
Player 1 (X) puts a token in column 4

  0 1 2 3 4 5 6
0     O O
1     O X
2   O X X X
3   X O O O
4   X O X X
5   O X X X O

Player 1 and Player 2 both have a tie with streak = 3
Running Time: 27.8220000267
```