***Due:   Wednesday, April 29th, 11:59 PM***

Late submissions not accepted

Game Rules:

* Connect 4 •<https://en.wikipedia.org/wiki/Connect_Four>
* Reversi • <https://en.wikipedia.org/wiki/Reversi>
* TicTacToe • <https://en.wikipedia.org/wiki/Tic-tac-toe>

**Requirements:**

This is MajorPP #3.

Write a LISP program to play either the game "Connect Three", Tic-Tac-Toe , or Reversi on a size 4x4 game board.  (Clarification:  all three games must use a 4x4 game board) Your program must use the minimax-decision search algorithm and should be invoked by the function call:  
  
> (connect-3)

In the case of Connect Three or Tic-Tac-Toe, a win is three in a row.  In the case of Reversi, a win is which ever player has the most pieces on the board.  If you choose to do Reversi, limit your board size to 4 x 4.

The game is single player, human vs the computer AI.

*Additional Information:*

You can represent a 4x4 board in any way you like.   One way to do it would be as a single list:

( 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 )

Corresponding to a board like this:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

Here, the numbers simply represent positions on the board and their relation to the list.

Remember, this is a purely deterministic environment.  There is no need to do any kind of random number generation.

You'll need to represent the each space in one of three ways:  Player Max, Player Min, and an empty space.

So, maybe your start state is something like:

( \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ )

where each \_ represents a blank. 

If X is the human player, and then O is the AI, then if the human player decides to make this first move (in Connect Three), then the board would look like:

|  |  |  |  |
| --- | --- | --- | --- |
| \_ | \_ | \_ | \_ |
| \_ | \_ | \_ | \_ |
| \_ | \_ | \_ | \_ |
| \_ | X | \_ | \_ |

With a corresponding list (and state):

( \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ X \_ \_ )

Now, the minimax-decision search kicks in and a search for the AI's move begins, playing out all possible games in it's "head", virtually, through the search tree.  Max is representing the AI, MIN would be representing the human.

The first state generated by the algorithm in the search space could look like this:

( \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ O X \_ \_ )

Corresponding to a board that looks like:

|  |  |  |  |
| --- | --- | --- | --- |
| \_ | \_ | \_ | \_ |
| \_ | \_ | \_ | \_ |
| \_ | \_ | \_ | \_ |
| 0 | X | \_ | \_ |

The the next state generated could look like

( \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ O X X \_ )

with a board like:

|  |  |  |  |
| --- | --- | --- | --- |
| \_ | \_ | \_ | \_ |
| \_ | \_ | \_ | \_ |
| \_ | \_ | \_ | \_ |
| O | X | X | \_ |

This recursively repeats until a terminal state is generated.  If the terminal state is in favor of MAX, then the AI will take the move indicated by the very first state generated through which the path to the favorable terminal state passed.  Otherwise, the depth-first search continues until a the best terminal state for MAX is found.

Assuming this search branch terminates in a terminal state good for MAX, the search is over, the AI takes their turn and then the actual game board would get updated to:

|  |  |  |  |
| --- | --- | --- | --- |
| \_ | \_ | \_ | \_ |
| \_ | \_ | \_ | \_ |
| \_ | \_ | \_ | \_ |
| O | X | \_ | \_ |

**Notes:**

* The red board is the actual game board.  The black board are the boards generated in the search.
* The AI's turn may take a LONG time (minutes...).  To speed it up, you could modify minimax-decision to use some form of pruning or depth-limited search (not required).
* To quote my AI professor and former department chair:  *I will not coauthor or debug your program for you.*
* That said, don't hesitate to ask any questions you like.

**Hints:**

* Check out the MINIMAX-DECISION algorithm in the text.  Follow the pseudocode.  Don't reinvent the wheel.
* Instead of lists, arrays may be easier.  Try both.
* Recursion will make this easier.
* If you wait till the last minute, you will not finish it.
* Don't forget, you can run LISP from the command line.  Put all your LISP code in a text file (similar to Python), drop to the command line and type clisp program.lisp.

**Submission Instructions:**

* Submit your program in a single, plain-text source file.
* Do not attach any screenshots.
* Do not archive your submission (no ZIP, TAR, etc).
* Include a brief summary of how your program is built.  This could include a list of functions, how they work, what their purpose is.  Explain your code and how it works.
* If you do not follow these submission instructions, you will lose points.