**CS 6613 * CS 4613** **E. K. Wong**
**Project: *4 X 4 Tic-Tac-Toe Game*** **Spring 2017**

***Total points:*** *100* (*for basic part*). ***Optional extra credits:*** *up to 35 points.*

***Project Description:*** Design and implement an *interactive 4×4 Tic-Tac-Toe game* for a person to play against a computer. The game consists of a *4×4* grid. To win, a player must place 4 of his/her symbols on 4 squares that line up vertically, horizontally or diagonally (45 or 135 degrees.) Figure 1 gives an example of a winning pattern for the player with the X symbol.

| | | | |
|---|---|---|---|
| | O | | X |
| | O | X | |
| | X | | O |
| X | | | |

Figure 1. Play X wins.

Use the ALPHA-BETA SEARCH algorithm in Figure 2 to implement your game. Note that in the algorithm in Figure 2, the main function uses *min_utility* and *max_utility* as the initial alpha and beta values for the MAX-VALUE function, where *min_utility* and *max_utility* are the minimum and maximum utility values defined for terminal nodes in the game. This will allow your program to prune more branches and run more efficiently than using minus infinity and positive infinity as initial values as in the original alpha-beta search algorithm.

If your program cannot search to the maximum depth in reasonable time for a move (say 10 seconds), then you should cutoff the search at a certain depth level *l* and use the following evaluation function to estimate the expected utility value for non-terminal nodes

$$Eval(s) = 6X_3 + 3X_2(s) + X_1(s) - (6O_3 + 3O_2(s) + O_1(s))$$

where *s* is the current state, $X_n$ is the number of rows, columns, or diagonals with exactly *n* $X's$ and no *O's.* Similarly, $O_n$ is the number of rows, columns, or diagonals with just *n O*'s. To compute the above evaluation function, you should consider every row and every column but only the middle 45 degree diagonal (with 4 squares) and the middle 135 degree diagonal (with 4 squares). You should set the cutoff level *l* as far down as possible for your program to perform well (in terms of intelligence) and yet finish computing for a move within 10 seconds.

There are three types of terminal nodes: (1) player X wins, (2) player O wins, and (3) draw -- *4×4* grid is filled up and neither player has 4 symbols lined up. Assign a utility value of +1000, -1000 and 0 to the three types of terminal nodes, respectively.

Each time the ALPHA-BETA SEARCH function is called to return the best action, output the following information (statistics) on the screen: (1) whether cutoff occurred, (2) maximum depth reached, (3) total number of nodes generated (including root node), and (4) number of times pruning occurred within the MAX-VALUE function and (5) number of times pruning occurred within the MIN-VALUE function. As a minimum requirement, your program will run in *command line mode* that allows the player to specify which square to place his/her symbol. Python, C/C++, C# or Java are preferred programming languages. If you plan to use another language, send me an e-mail first.

Design three different levels of difficulty: 1 (easy), 2 (intermediate) and 3 (difficult). Let the design above be leve1 3 (difficult). You can think about how to modify the above design for the easy and intermediate levels. The human player can choose which level to play when the game starts.

***Extra Credits:*** The followings are optional. You can implement one or both of the followings to get extra credits.

- [15 points] Design and implement a graphical user interface (GUI) that displays the 4 x 4 grid on the screen and allows the user to use the mouse to pick the square to place the symbol.
- [20 points] Invent another evaluation function *Eval(s)* and *show* that it performs better than the evaluation function above. If you choose to do this, you still have to implement the *Eval (s*) described above and then make comparison with yours.

***What do hand in:*** You are required to hand in the following by uploading to NYU Classes.

- Source code for your program. Complete documentation and in-line comments are required for your source code. Points will be taken off if you do not have these.
- Your executable code.
- An MS Words, PDF, or plain text file that contains: instructions on how to compile and run your program, and a high-level description of your program design.

***Demo:*** A 15-min demo of your program on your notebook computer will be required.

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, min_utility, max_utility)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← -∞
    for the next a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for the next a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

**Figure 2. Alpha-Beta Algorithm**