**CS 4613 * CS 6613** **Spring 2018**
**Project Hints** **E. K. Wong**

**Main function:** As discussed in class, you need to write a MAIN function that will call the ALPHA-BETA-SEARCH (ABS) function every time the computer makes a move. Given the current *game state* (board configuration), the ABS function returns the next best move for the computer to execute. The human then makes the next move and a new game state will be generated. The MAIN function will then call the ABS function again with the new game state as input. The ABS function returns the next best move and the computer executes it. The cycle repeats until there is a winner or a draw.

**Cutoff and evaluation function:** The project requires that you cutoff the search and use an evaluation function if you cannot finish computing in 15 seconds. As discussed in class, you can add a CUTOFF function and an EVAL function in the MAX-VALUE and MIN-VALUE functions as shown below. One way to implement the CUTOFF function is simply by setting a fixed depth limit (see below.) As soon as the search reaches the depth limit, you cutoff the search and compute a "score of desirability" for the current state. You should design your program and set a depth limit such that **all the actions** in the for loop (for the next *a* in ACTIONS(*state*) do) in the MAX-VALUE function at the root node will be executed within the 15 second time limit. In case there is pruning, you can exit the *for* loop without finishing all the actions, but you should not exit the for loop simply because the 15 seconds time is up. Try running your program a few times and time it, you should be able to manually set a depth limit that satisfies the above.

**Terminal states and utility values:** You can define three terminal states: MAX wins, draw, and MIN wins and **assign the utility values of +1,000, 0 and -1,000 to the three terminal states,** respectively. Your evaluation function should return a value in the **interval (-1000, +1000)** depending on how favorable the current board position is to the MIN player or the MAX player. You program will run faster if you use an initial $\alpha$ value of *minimum_utility* = -1,000 instead of $-\infty$ and an initial $\beta$ value of *maximum_utility* = +1,000 instead of $+\infty$ (see algorithm below.)

**Capture moves:** To deal with obligatory *capture moves*, you can write code to check whether there are capture moves for the current game state before entering the *for* loop in the MAX-VALUE and MIN-VALUE functions. If there is no capture move, you continue and enter the *for* loop in the normal way. If there are one or more capture moves, you can *mask* all the non-capture moves and skip them in the *for* loop since they will not be executed in the real game. This way, only the actions associated with capture moves will be executed and produce child nodes. As a result, your AI will be smarter and run more efficiently.

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, minimum_utility, maximum_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)
    if CUTOFF(state) then return EVAL(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)
    if CUTOFF(state) then return EVAL(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

function CUTOFF(state) returns true or false
    if state.depth = limit then return true else return false
```

**Alpha-Beta Search Algorithm**