# AI Programming (20%)

1. **Program the pebble distribution game we have been discussing in class**
    1. ~~**2 person teams: select a partner or I can assign one MONDAY 10.27**~~
    2. **Develop and submit the following to Canvas (PDF or WORD file) SATURDAY11.9 BEFORE 11:59PM**
        1. **One heuristic function, to be called H1 henceforth, (an English explanation and a corresponding algorithm that given a game board, returns its heuristic value from the perspective of the computer) – 6600 students need to provide two such functions H1 and H2 with corresponding explanations and algorithms, and these should be as different from each other as possible.**
        2. **Modified Alpha-Beta Minimax Game Playing algorithm (MODIFY THE ALGORITHM GIVEN IN THE NEXT SLIDE) to do the following:**
            1. **Look-ahead a fixed number of moves (plys) and count states at the lowest level as leaves (i.e., terminal states)**
            2. **Abandon a path if it reaches a game board or state that has already appeared in that path (i.e., no loops – note that this is different from redundant paths leading to transpositions)**
            3. **To store any new game board with its minimax value in a hash table (Transposition Table) and each time to first look up this table and execute the MAX or MIN algorithm for that state only if the game board is not in the Transposition Table (if it is, directly return the corresponding minimax value) – this needs to be implemented only within the MIN and MAX algorithms, not the αβsearch algorithm 6600 ONLY**
        3. **Come up with a way to order the moves to improve the alpha-beta pruning performance; explain it in English and provide a corresponding algorithm that takes a game state/board and the player to move next as parameters and returns a set of ordered moves**
    3. **Implement a simple GUI as the game interface:**
        1. **Show the two row game board with # of pebbles in each square shown as an integer**
        2. **At game start, GUI should ask for the following parameters: # of squares per player (assume only 2 players), # pebbles per square, # plys AND whether the game should "run" or "step" through each move.**
        3. **"Run" is applicable only to computer vs. computer games. In this case you may choose to continuously update the GUI as the game is being played, or to show the game ending state only. "Step" may be chosen for computer vs. computer games and computer vs. human games. In this case the GUI should behave as follows:**
            1. **If compute vs. human, after a computer move the GUI game board should update and wait for human to type input to indicate which square the human's move will take place, then update the board to indicate that move's result and wait for human to hit any pre-determined key to indicate that now the computer could make a move and update the game board and so on.**
            2. **If compute vs. computer, after each move the GUI game board should update and wait for human to type a pre-determined key to indicate that now the next move could be made and update the game board and so on.**
    4. **Implement the AND-OR search algorithm to play the game**
    5. **Implement the Alpha-Beta Minimax algorithm you submitted**
    6. **Pit one against the other in tournaments, collect data and submit your results along with explanations and your source code (Details on this forthcoming; December 4; No modification of your program allowed after this date)**
    7. **Meet with me to demo your program, let me play against it, explain your results, and answer my questions (December 5 – December 11)**

**function αβsearch**(state) returns a:action

v=−∞; a=NIL; α=−∞; β=+∞ //v, a, α & β are variables local to these functions

for each action in ACTIONS(state) do

state'=RESULT(state, action)

v'=MIN(state', α, β)

if v'>v then v=v'; a=action end if

if v≥β then return a //prune based on β

else if v>α then α=v //update α

return a

---

**function MAX**(state, α, β) returns v:utility-value

if TERMINAL-TEST(state)==true then return UTILITY(state)

// UTILITY(state) computes the utility value your heuristic function assigns to state, which should be greater for states that have a greater likelihood of leading to a win for the computer. You need to provide the algorithm for UTILITY as part of your first submission (6600 students should provide two algorithms corresponding to H1 and H2)

v=−∞

for each action in ACTIONS(state) do

state'=RESULT(state, action)

v'=MIN(state', α, β)

if v'>v then v=v' end if

if v≥β then return v //prune based on β

else if v>α then α=v //update α

return v

---

**function MIN**(state, α, β) returns v:utility-value

if TERMINAL-TEST(state)==true then return UTILITY(state)

v=+∞

for each action in ACTIONS(state) do

state'=RESULT(state, action)

v'=MAX(state', α, β)

if v'<v then v=v' end if

if v≤α then return v //prune based on α

else if v<β then β=v //update β

return v

# Notes

- I strongly recommend that you work out these algorithms' operation on the game tree of Fig 5.5 to get an intuitive understanding of the minimax value propagation, how alpha and beta values are updated, and when and how pruning takes place before you start modifying and implementing. Meet me if you have difficulty doing this.
- The heuristic function for games is different from heuristics for search that we discussed in chapter 3. Search heuristics are estimates of the cost of reaching the goal state from the current state being evaluated. Therefore those needs to be admissible (never overestimate the cost) and tend to decrease as one gets near the goal state (with the goal state's heuristic value being zero). Game heuristics are estimates of the likelihood of the computer winning the game from the current state, so tend to increase for states that are closer to winning states for the computer, with the highest value for states that are wins for the machine. Since these are not "costs," admissibility and consistency are not relevant (unless the moves cost something and that needs to be taken into account – in the pebble game, moves don't cost anything).