Adversarial Search

LESSON 6

Reading

Chapter 6

Outline

- Last time: Heuristic, Informed search
  - h(x): utility function

- Optimal decisions
- α-β pruning
- Imperfect, real-time decisions

Games vs. Search problems

- “Unpredictable” opponent -> specifying a move for every possible opponent reply
- Time limits -> unlikely to find goal, must Approximate
- Hmm: Is Ataxx a game or a search problem by this definition?

Two player games

Max always moves first.

Min is the opponent.

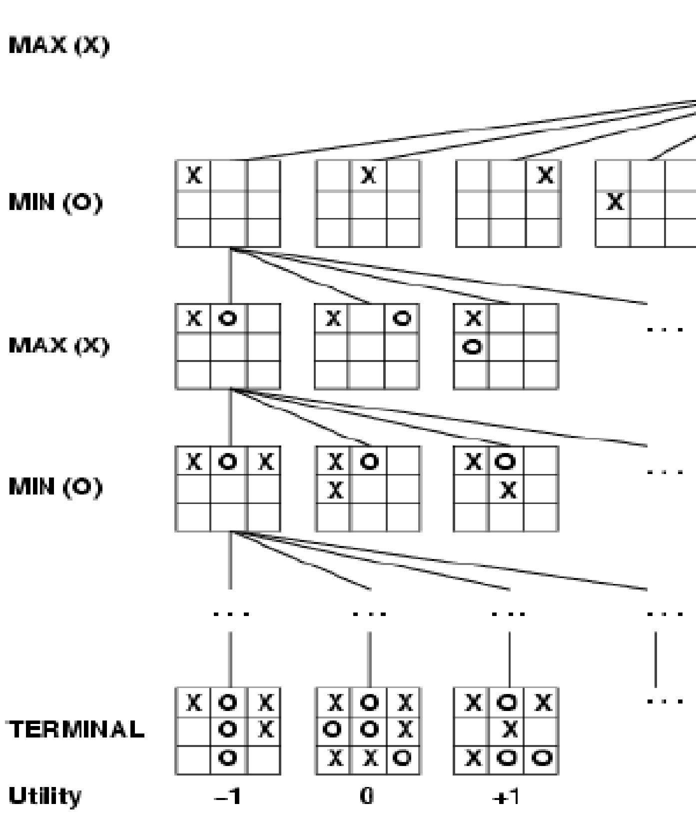
We have

- An initial state.
- A set of operators.
- A terminal test (which tells us when the game is over).
- A utility function (evaluation function).


Max Vs Min

The utility function is like the heuristic function we have seen in the past, except it evaluates a node in terms of how good it is for each player. Positive values indicate states advantageous for Max, negative values indicate states advantageous for Min.

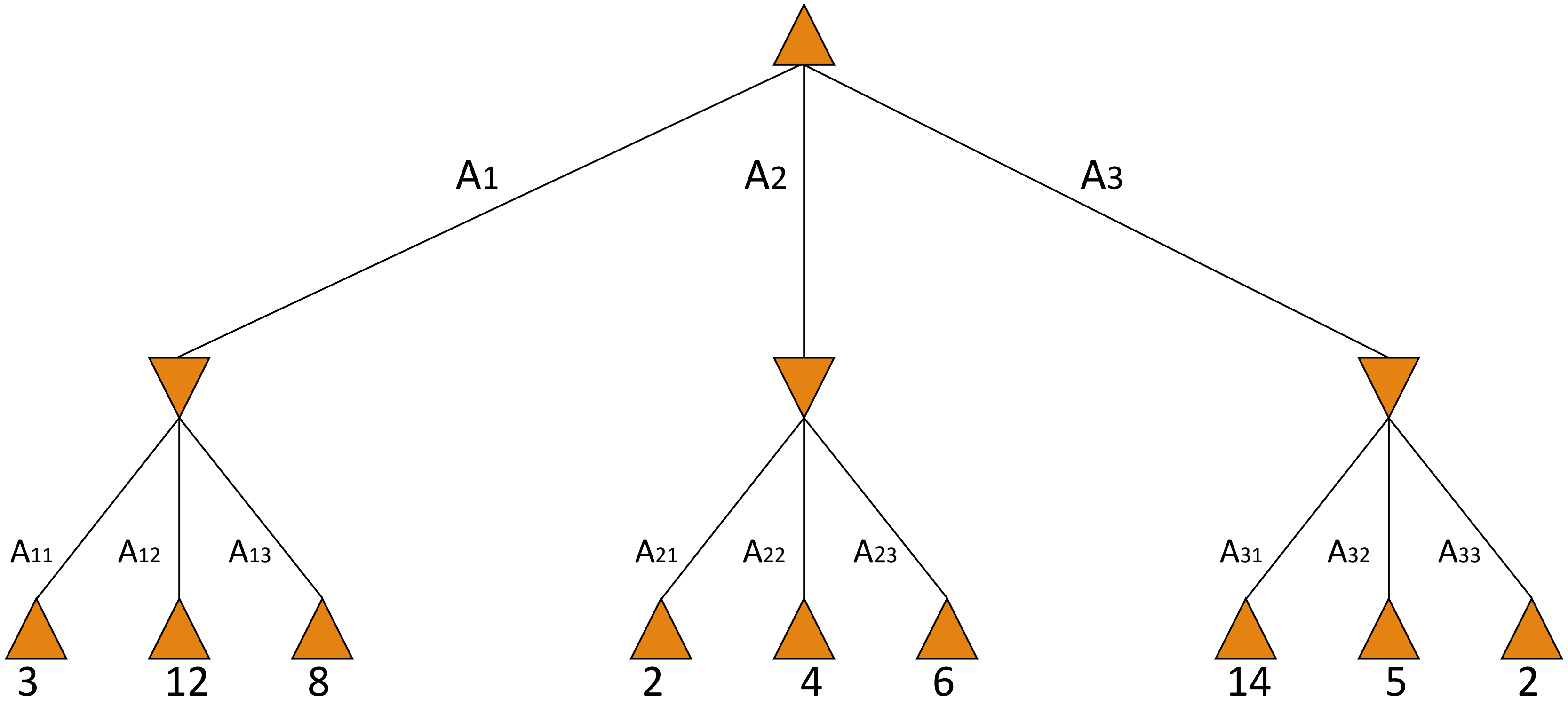
Game tree (2-player, deterministic, turns)



Key property: we have a **zero-sum** game.
- Loosely, it means that there's a loser for every winner.
- Total utility score over all agents sum to zero.
- Makes the game adversarial.

To think about: not zero sum?

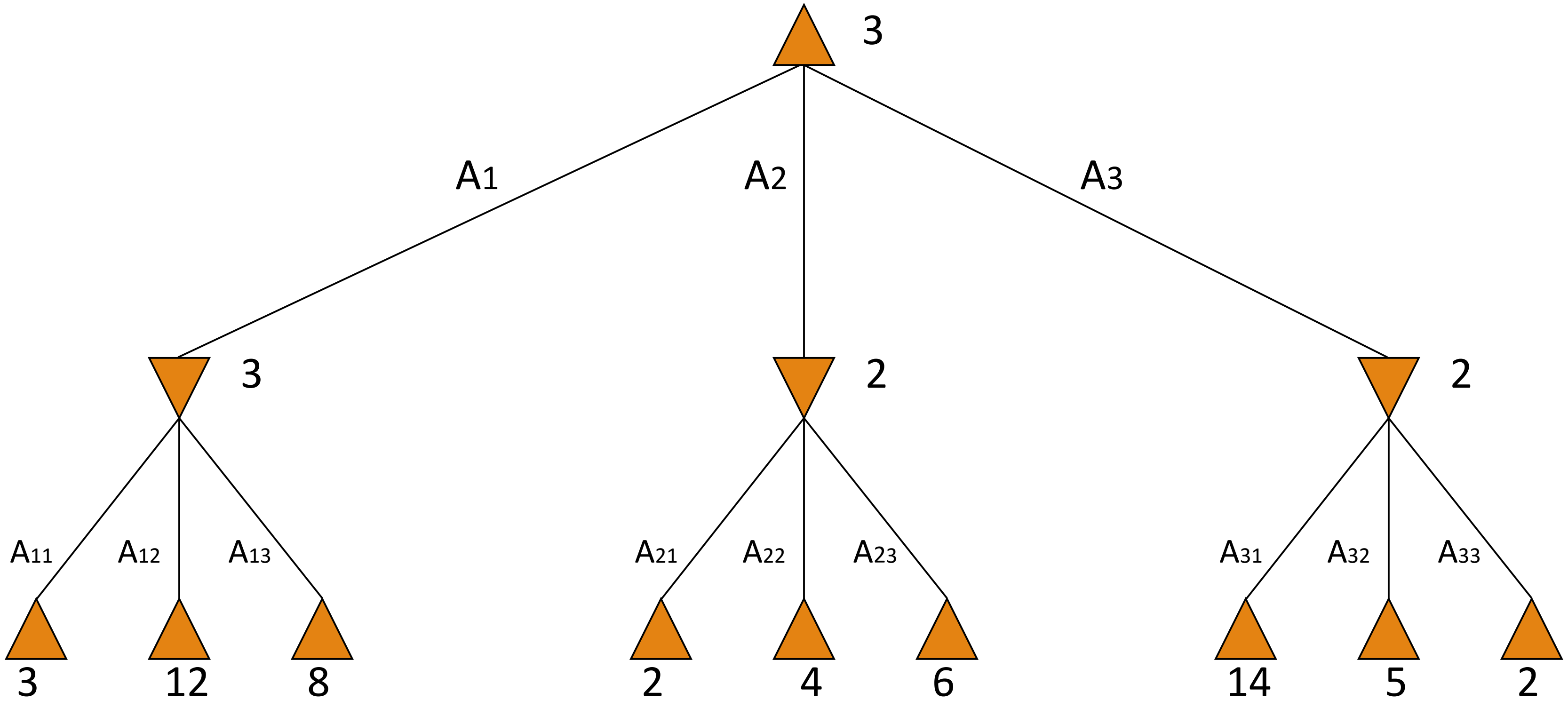
A simple abstract game.
Max makes a move, then **Min** replies.



An action by one player is called a *ply*, two ply (a action and a counter action) is called a *move*.

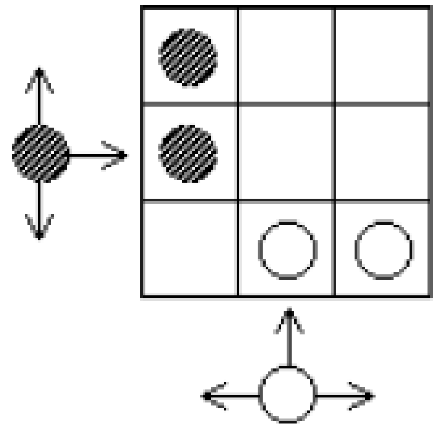
The Minimax Algorithm

- Generate the game tree down to the terminal nodes.
- Apply the utility function to the terminal nodes.
- For a **S** set of sibling nodes, pass up to the parent…
  - the lowest value in **S** if the siblings are $\triangle$
  - the largest value in **S** if the siblings are $\triangledown$
- Recursively do the above, until the backed-up values reach the initial state.
- The value of the initial state is the minimum score for Max.

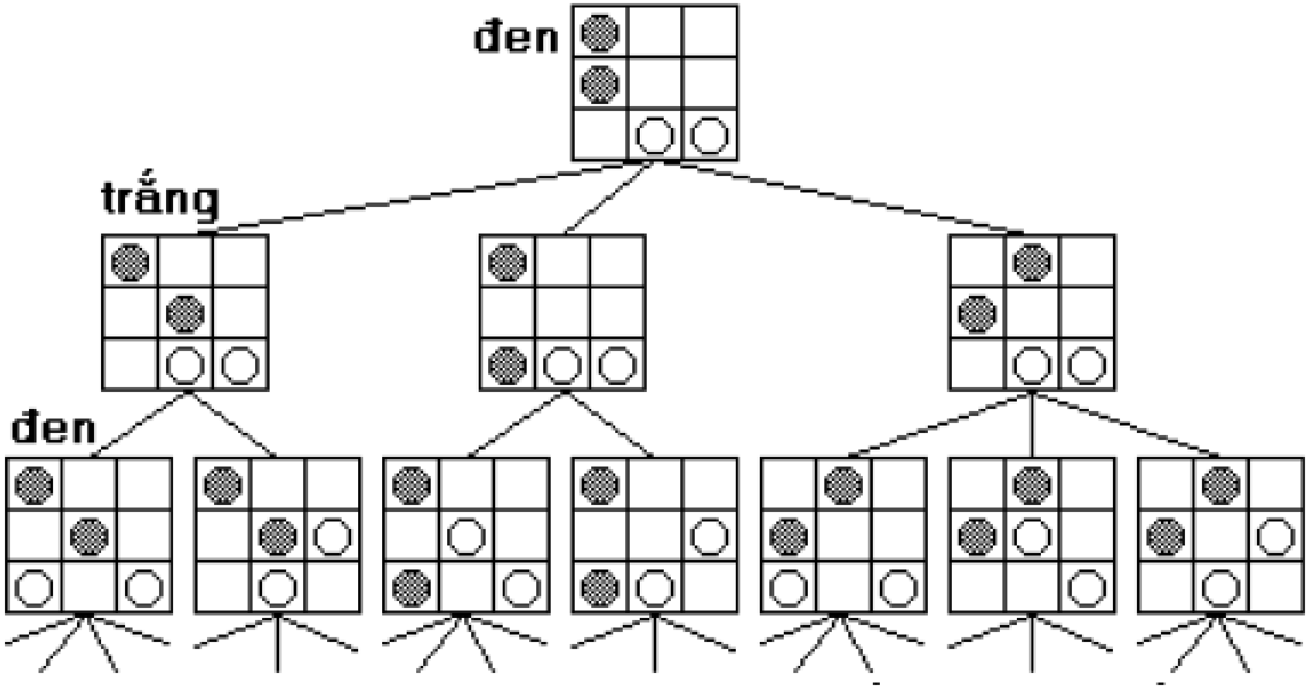


In this game Max's best move is A1, because he is guaranteed a score of at least 3.

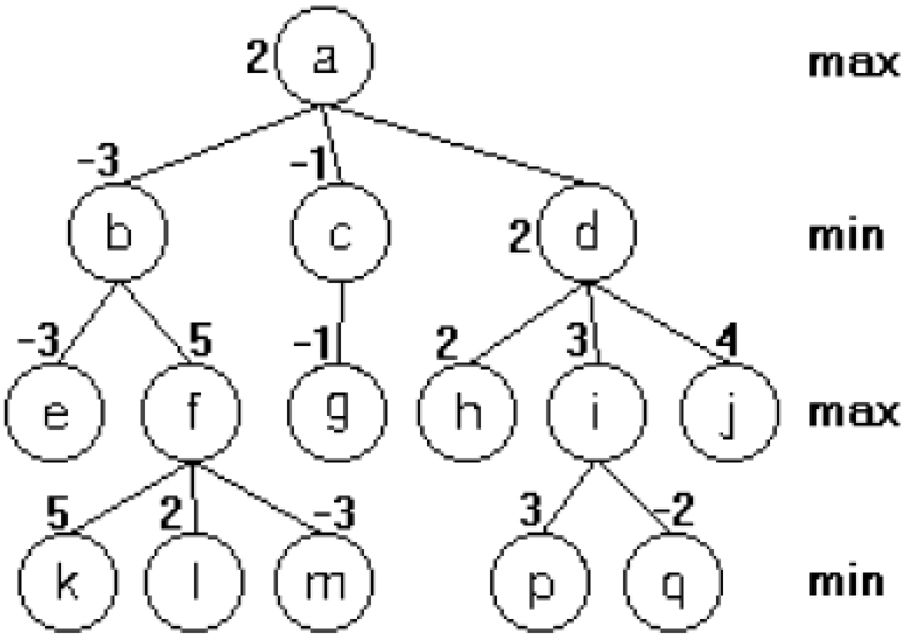
Example: Dodgen game



Example: Dodgen game



Example: Dodgen game



Evaluation Function

function *MaxVal(u);*

begin

  if *u là đỉnh kết thúc* **then** *MaxVal(u)* $\leftarrow$ *f(u)*

  else *MaxVal(u)* $\leftarrow$ *max{MinVal(v) | v là đỉnh con của u}*

end;

function *MinVal(u);* ?

Evaluation Function

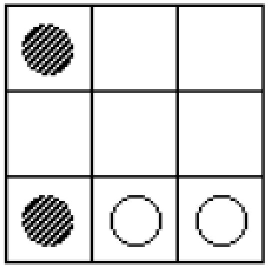
- Eval(u)
  - Eval(u) > 0 : prefer Max
  - Eval(v) < 0 : prefer Min

30	35	40
15	20	25
0	5	10

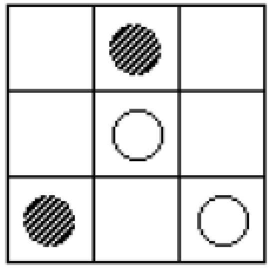
Giá trị quân Trắng.

-10	-25	-40
-5	-20	-35
0	-15	-30

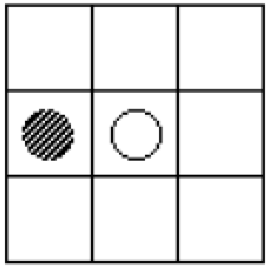
Giá trị quân Đen.



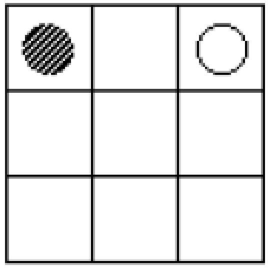
?



?

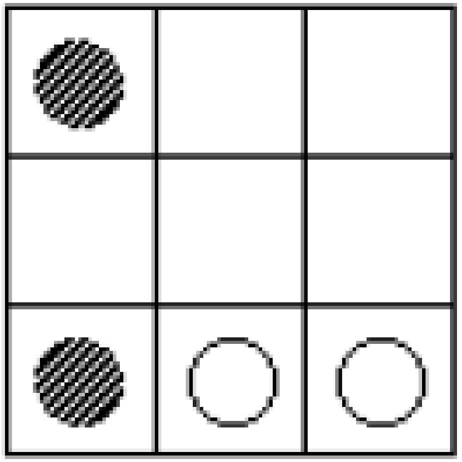


Trắng cản trực tiếp Đen được thêm 40 điểm.

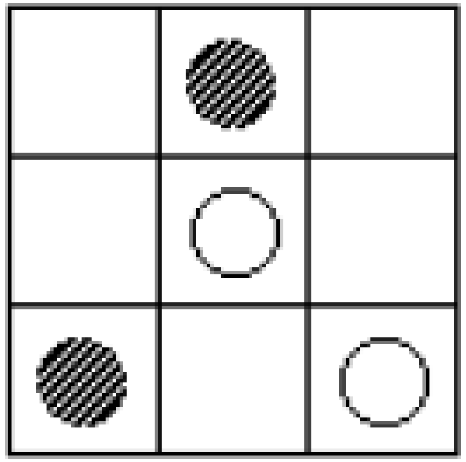


Trắng cản gián tiếp Đen được thêm 30 điểm.

Evaluation Function



75



–5

Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Properties of Minimax

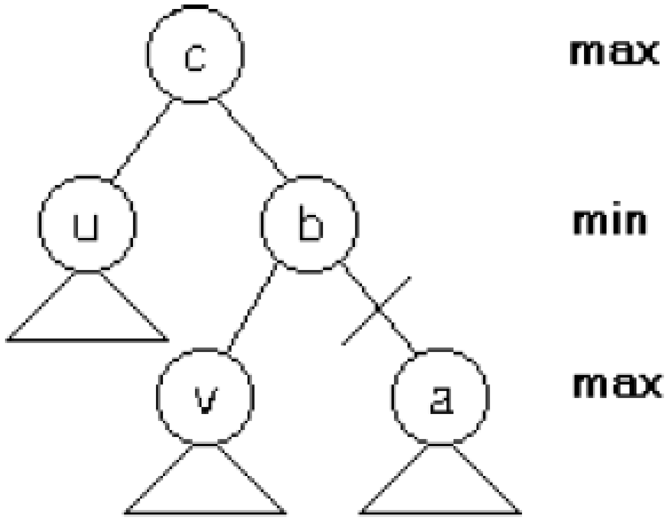
- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? O(b^m), where *b* is the effective branching factor and *m* is the depth of the terminal states.
- Space complexity? O(bm) (depth-first exploration)

For chess, b ≈ 35, m ≈ 100 for "reasonable" games

-> exact solution completely infeasible

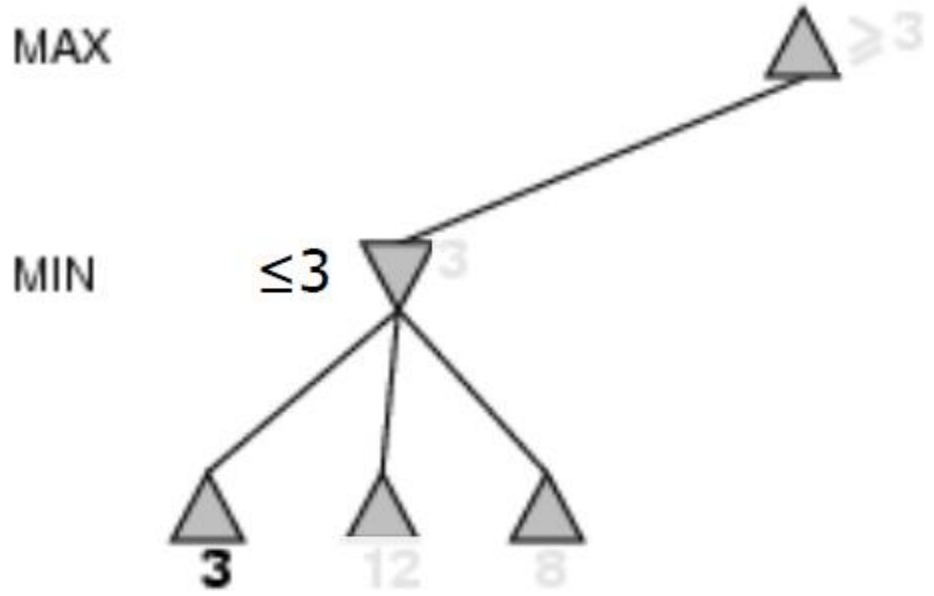
What can we do? PRUNING!

α-β Pruning

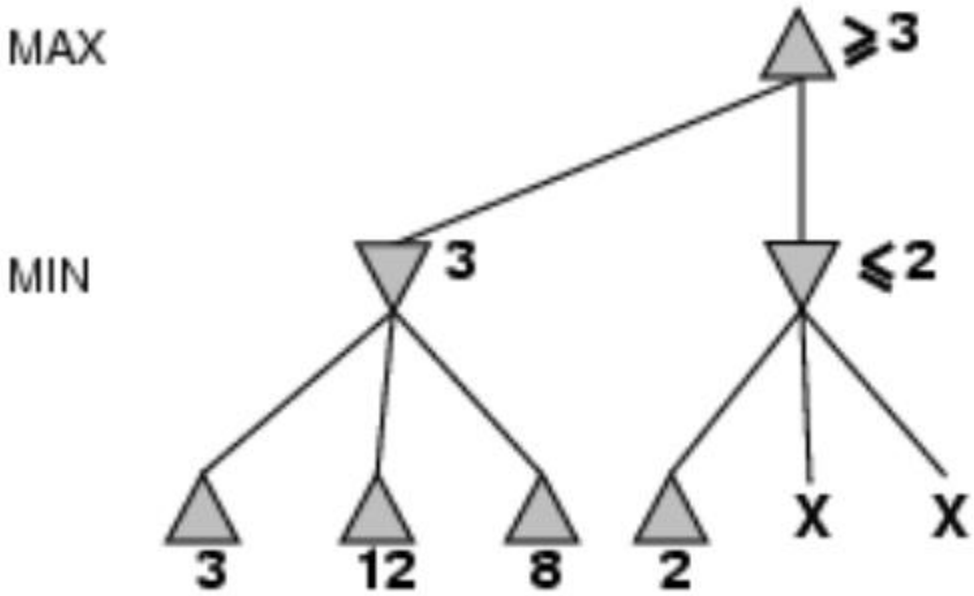


- Prunning *sub-tree a* if *Eval(u) > Eval(v)*

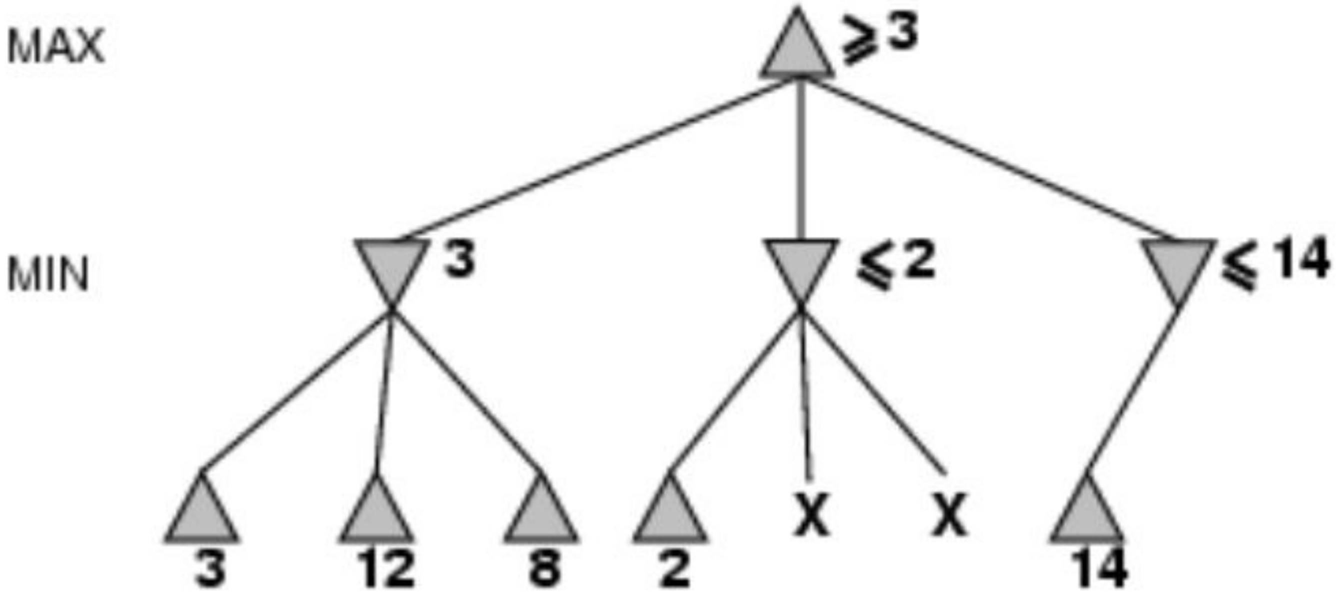
α-β Pruning example



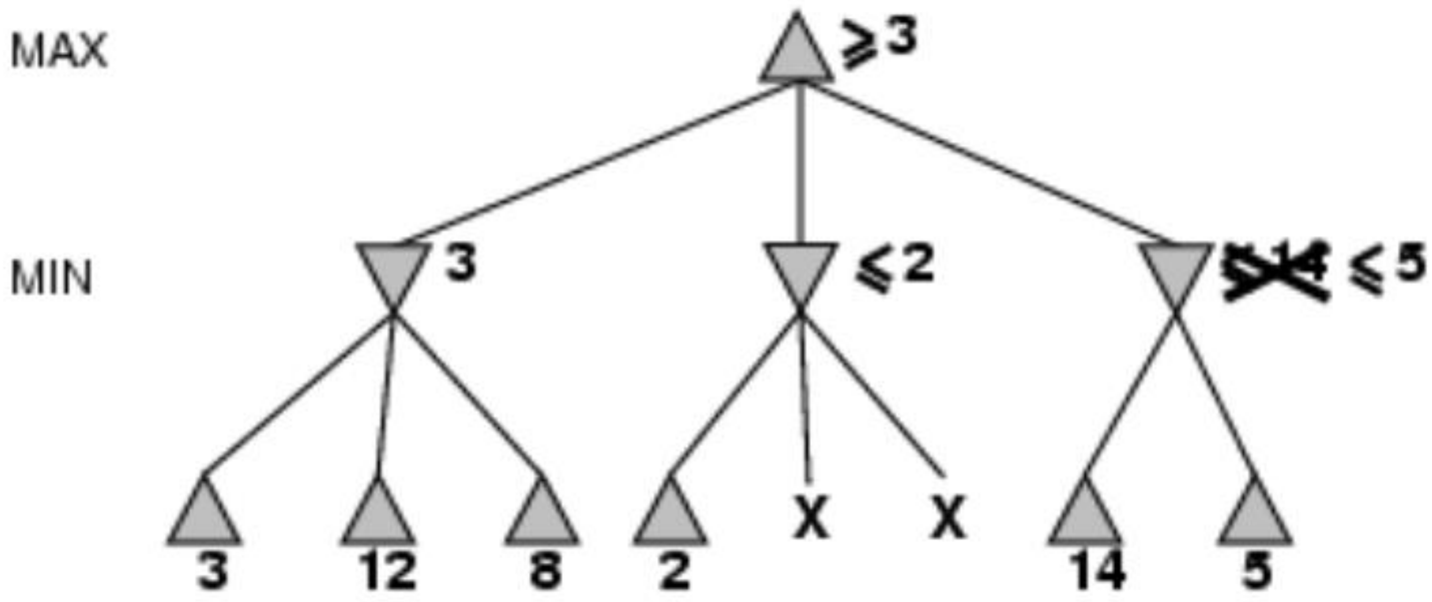
α-β Pruning example



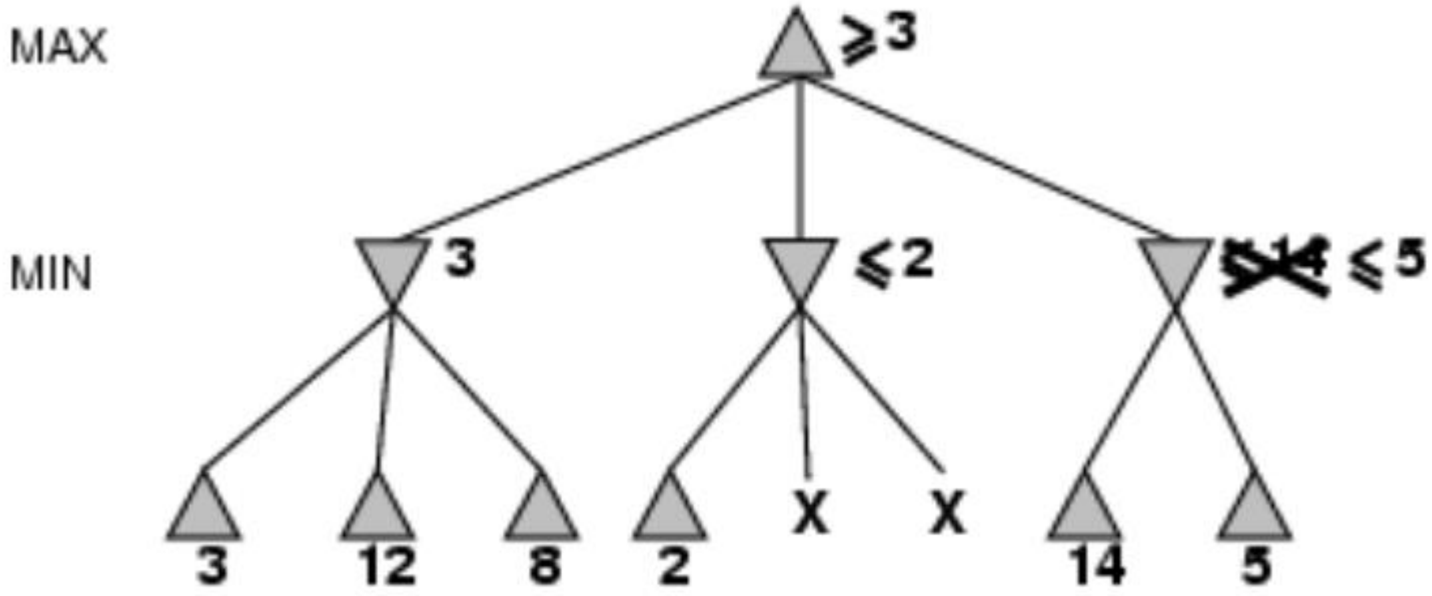
α-β Pruning example



α-β Pruning example



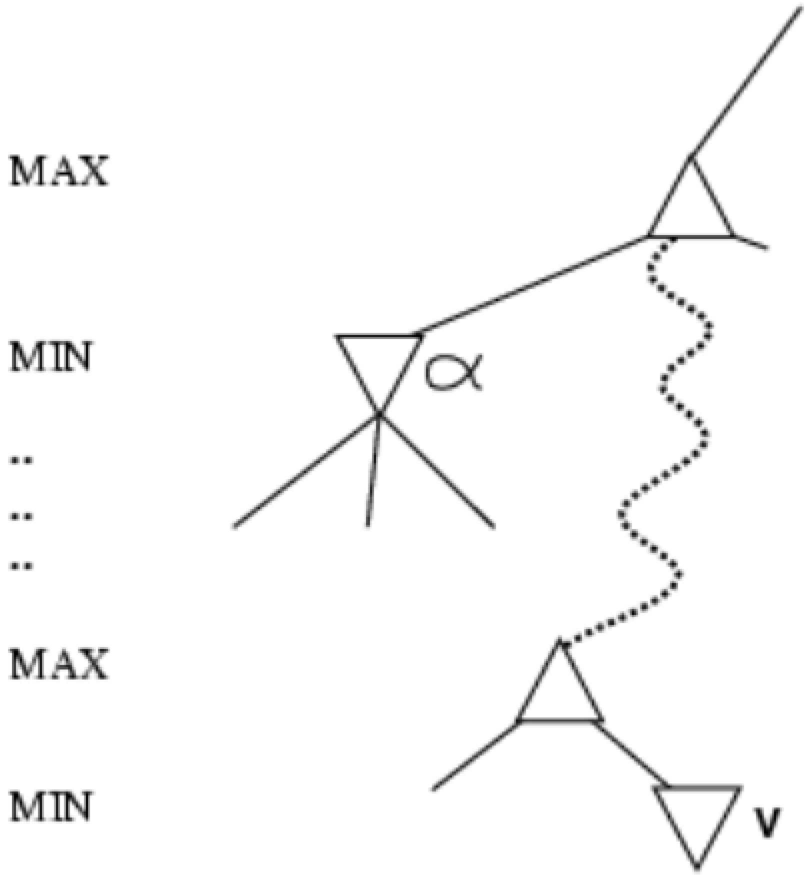
α-β Pruning example



Why is it called α-β

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If v is worse than α, *max* will avoid it

  → prune that branch
- Define β similarly for *min*



The α-β Algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game

    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v
```

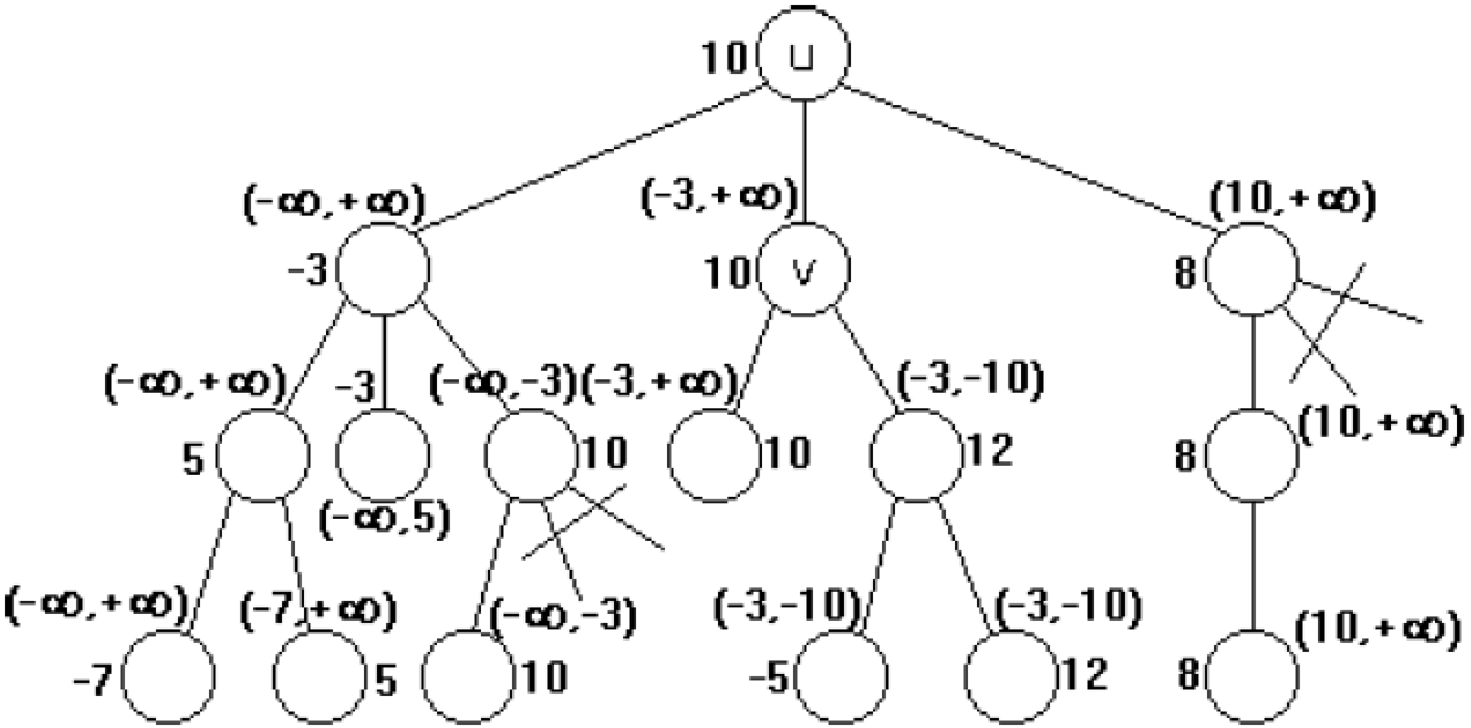
```
function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for  MAX along the path to state
            β, the value of the best alternative for  MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

The α-β Algorithm

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
    inputs: *state*, current state in game
            α, the value of the best alternative for MAX along the path to *state*
            β, the value of the best alternative for MIN along the path to *state*

    if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
    $v \leftarrow +\infty$
    for a, s in SUCCESSORS(*state*) **do**
        $v \leftarrow$ MIN(v, MAX-VALUE(s, α, β))
        if $v \leq \alpha$ **then return** v
        $\beta \leftarrow$ MIN(β, v)
    return v

The α-β Algorithm: example



Resource Limits

- The big problem is that the search space in typical games is very large.
- Suppose we have 100 secs, explore 104 nodes/sec -> 106 nodes per move

Standard approach:

- cutoff test:

  e.g., depth limit

- evaluation function

  = estimated desirability of position

Evaluation functions

- For chess, typically linear weighted sum of features

$$Eval(s) = w_1\ f_1(s) + w_2\ f_2(s) + \ldots + w_n\ f_n(s)$$

- e.g., $w_1 = 9$ with
  $f_1(s)$ = (# of white queens) – (# of black queens), etc.
- Caveat: assumes independence of the features
  - Bishops in Western chess better at endgame
  - Unmoved king and rook needed for castling
- Should model the *expected utility value* states with the same feature values lead to.

Other problems

- Applying utility functions on end game scenarios may not solve the game
- Use a policy or lookup table (taken from previous game history)
- Stochastic games
- Calculate the expected value of a position

What do you need to do

- Implement Minimax
- Implement Pruning (optional)
- Implement an evaluation function
  - ✓ Input: board, selected grid location
  - ✓ Output: continuous value

- (really optional) use state

Summary

- Games are fun to work on!
- They illustrate several important points about AI
- Perfection is unattainable -> must approximate
- Good idea to think about what to think about