Adversarial Search

Chapter 5

Sections 1 – 5

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

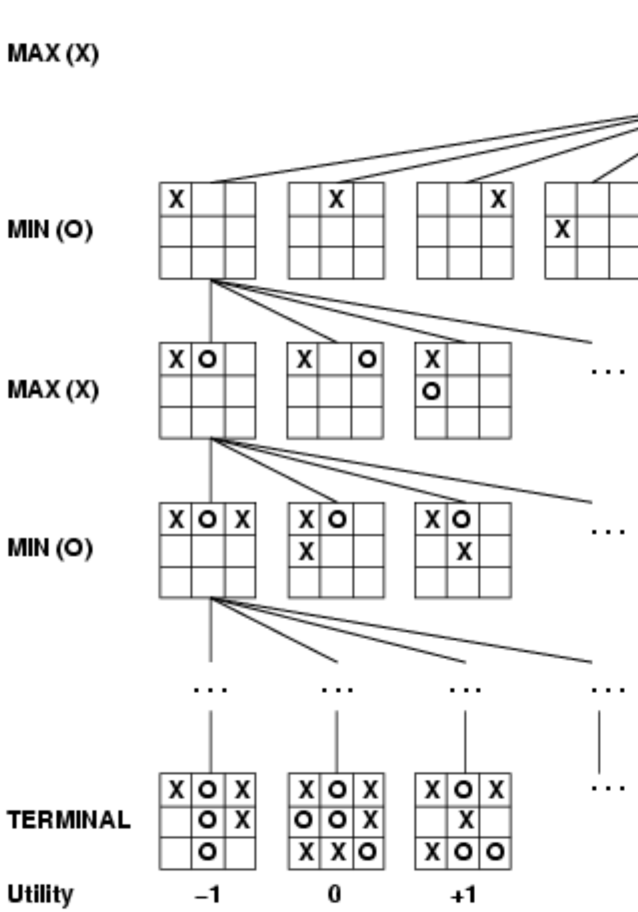
- Hmm: Is Ataxx a game or a search problem by this definition?

Let's play!

- Two players:
  - Max
  - Min

Formal Description:
- An initial state
- Successor function
- Terminal Test
- Utility Function

Game tree (2-player, deterministic, turns)



Key property: we have a **zero-sum** game.

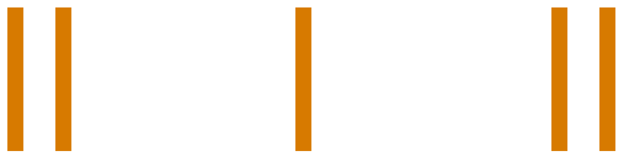
- Loosely, it means that there's a loser for every winner.
- Total utility score over all agents sum to zero.
- Makes the game adversarial.

To think about: not zero sum?

Example : Game of NIM

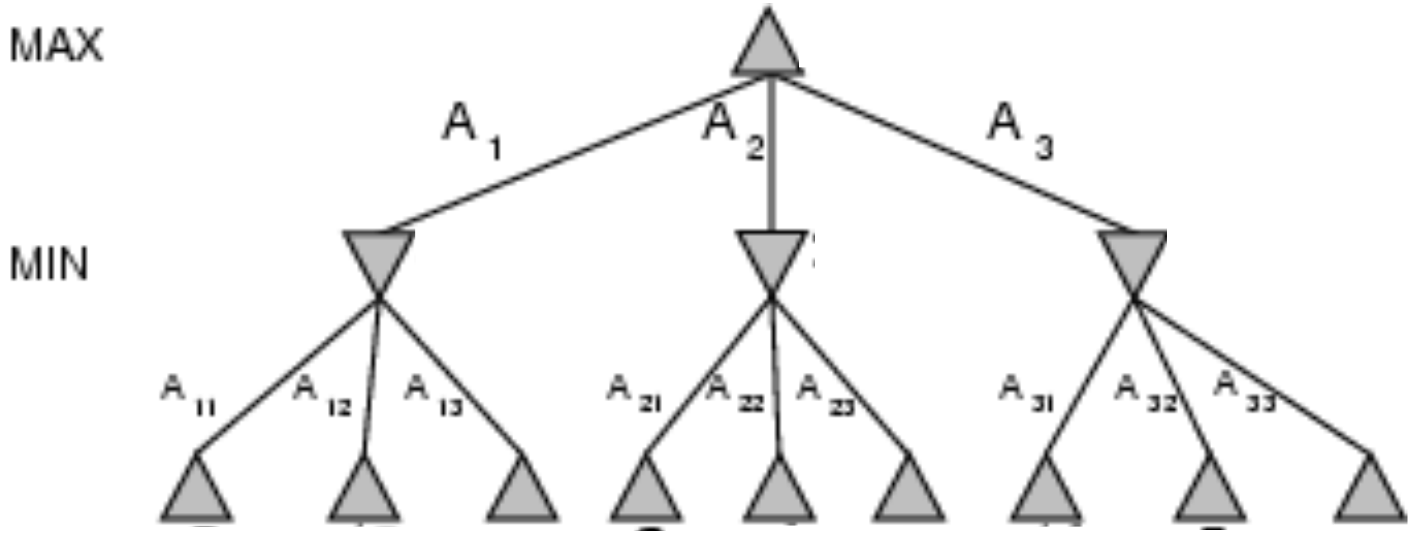
Several piles of sticks are given. We represent the configuration of the piles by a monotone sequence of integers, such as (1,3,5). A player may remove, in one turn, any number of sticks from one pile. Thus, (1,3,5) would become (1,1,3) if the player were to remove 4 sticks from the last pile. The player who takes the last stick loses.

- Represent the NIM game (1, 2, 2) as a game tree.



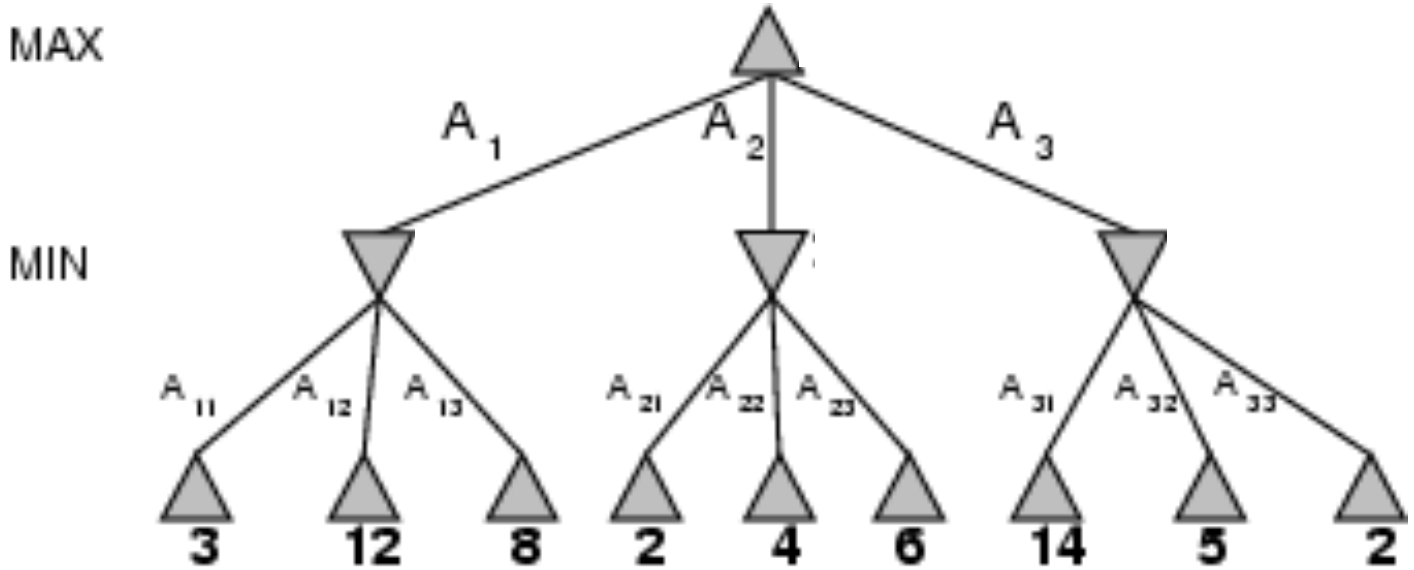
Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:



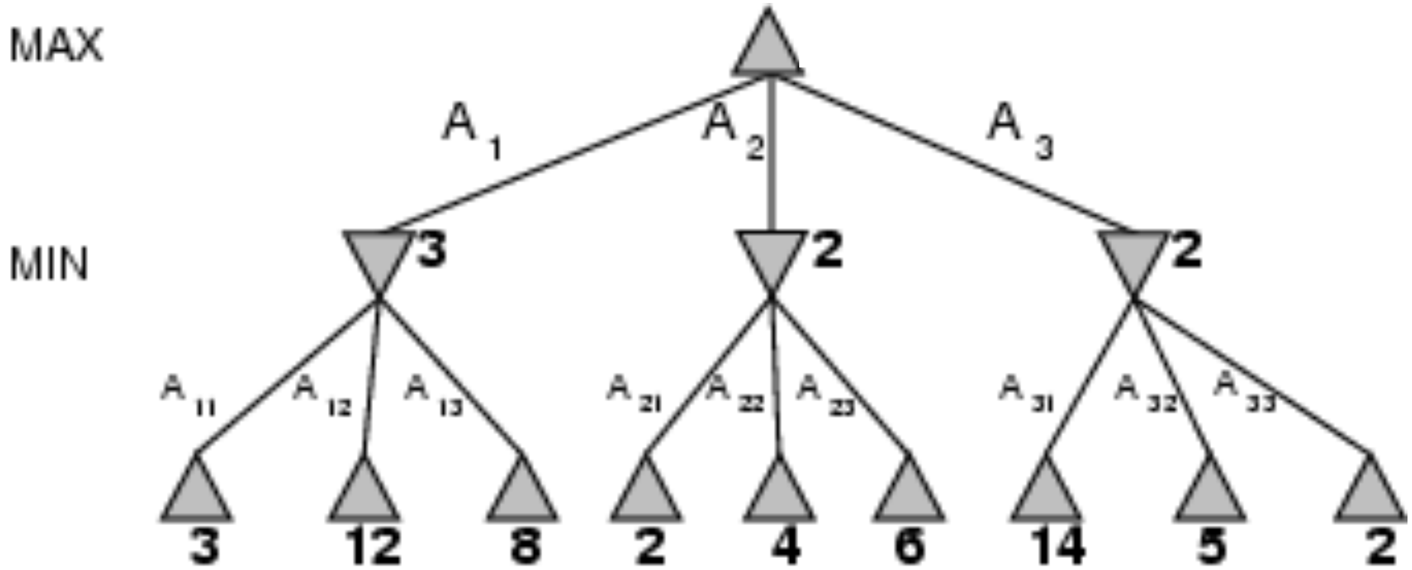
Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:



Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:



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
```

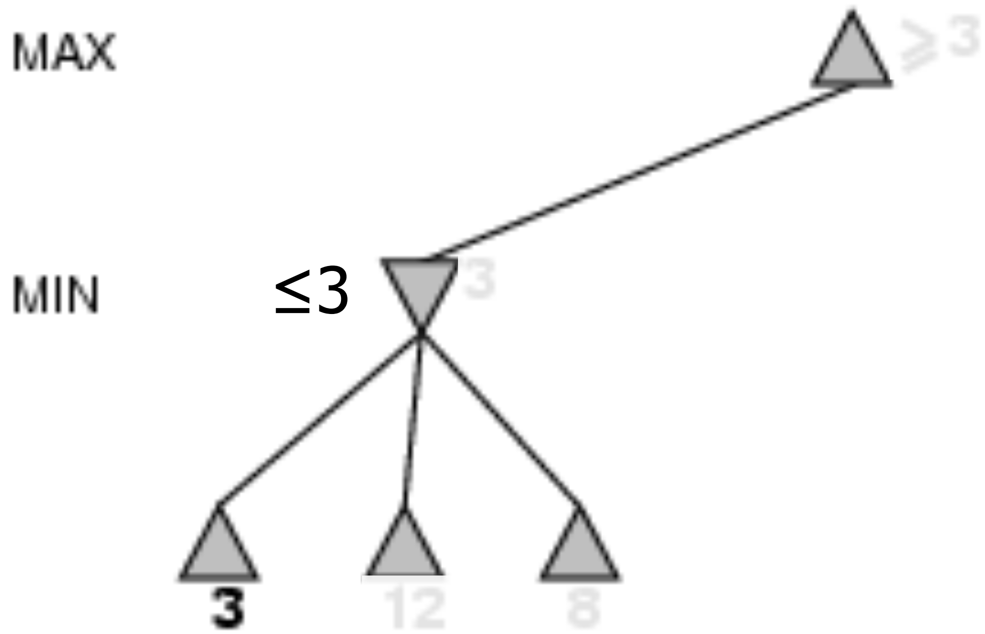
Properties of minimax

- Complete?
- Optimal?
- Time complexity?
- Space complexity?

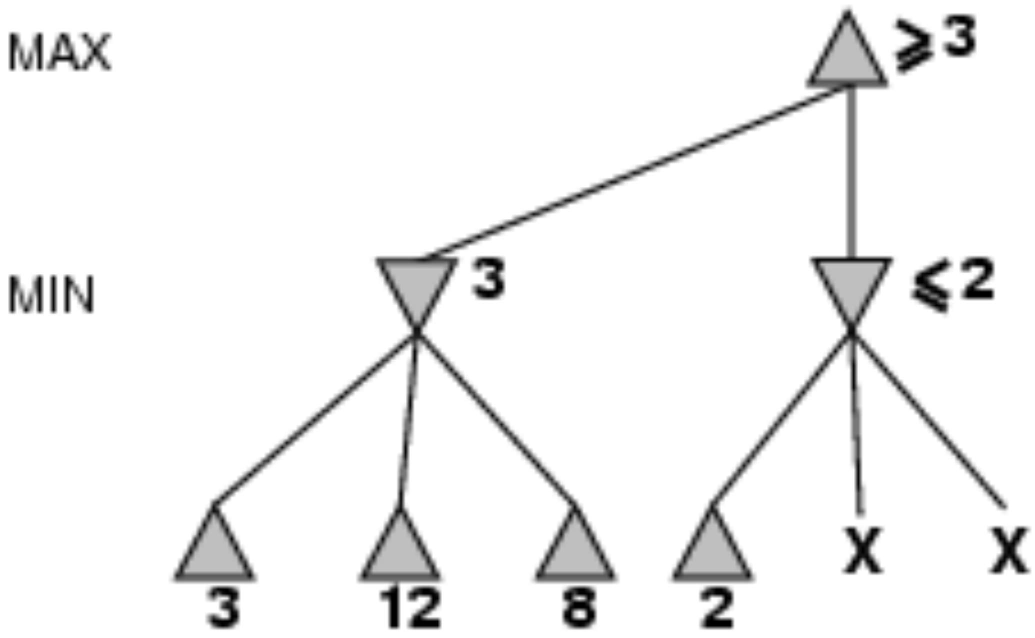
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
→ exact solution completely infeasible
- What can we do?

Pruning!

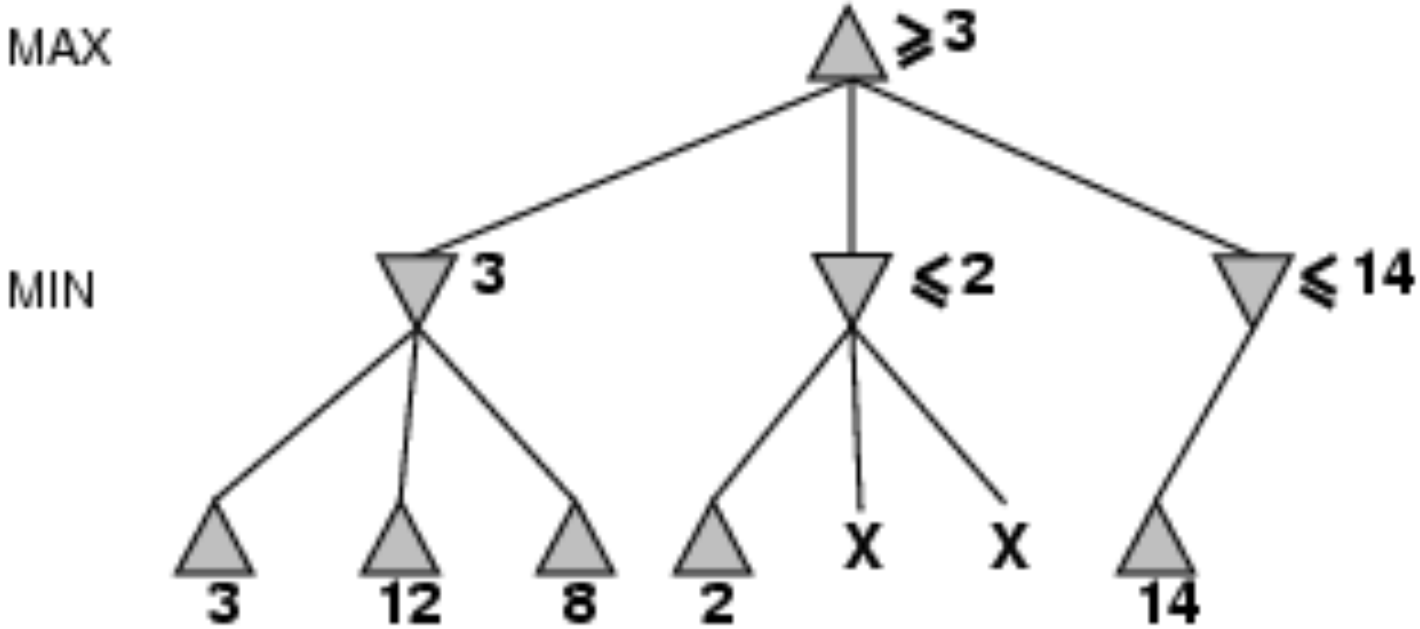
α - β pruning example



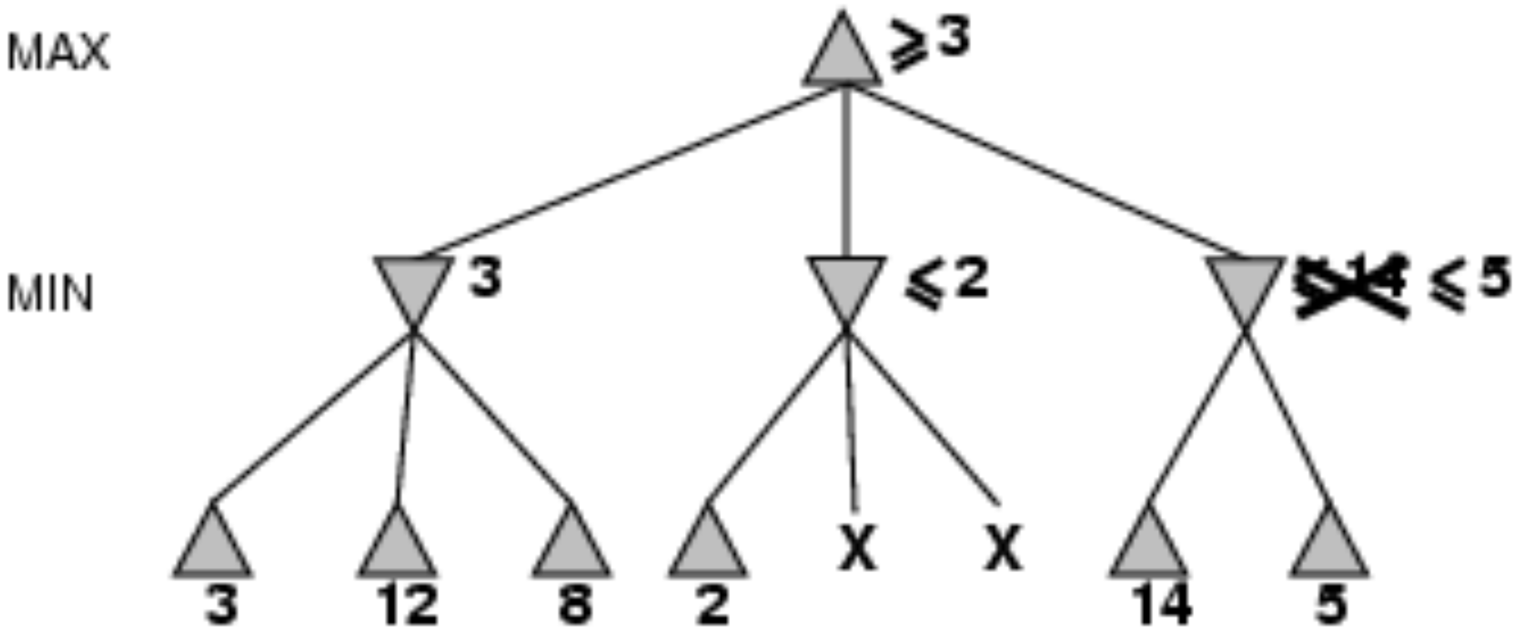
α - β pruning example



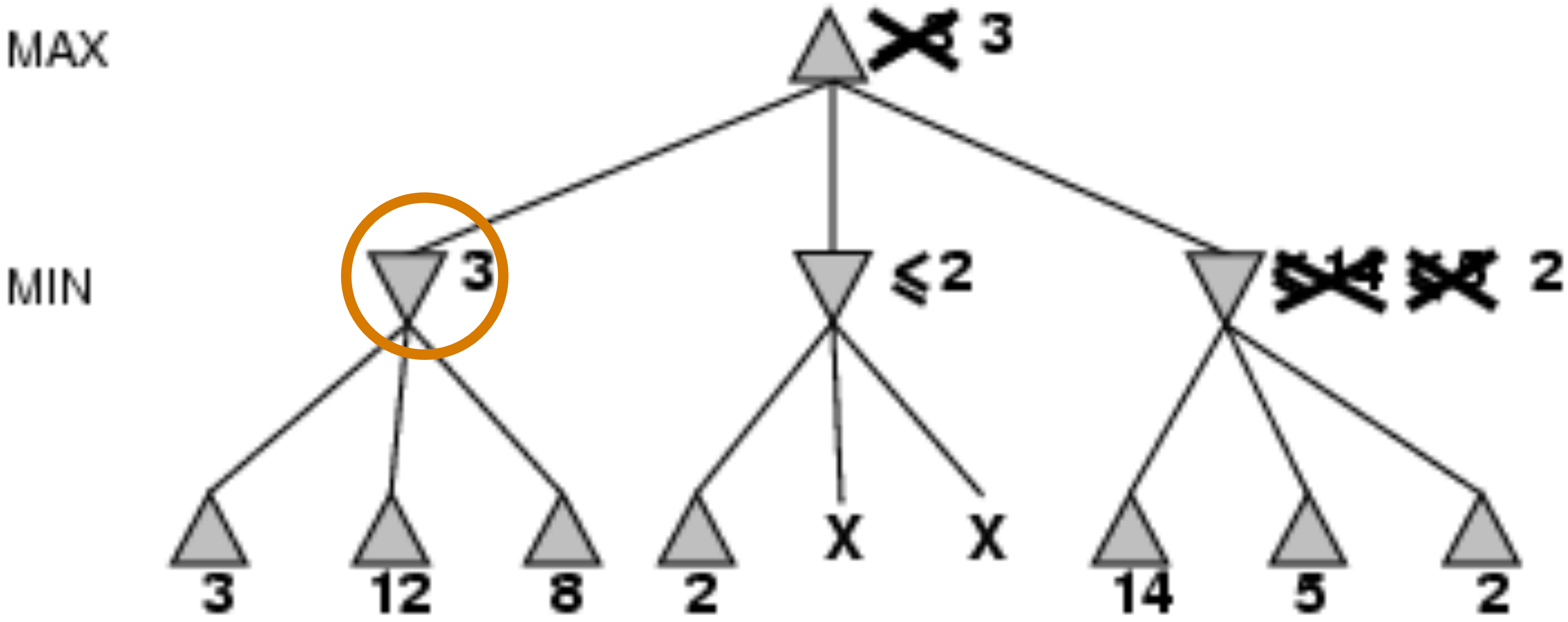
α - β pruning example



α - β pruning example



α - β pruning example



Properties of α - β

- Pruning does affect the final results

- Good move ordering improves effectiveness of pruning

- With "perfect ordering", time complexity =
  → doubles depth of search
  - What's the worse and average case time complexity?
  - Does it make sense then to have good heuristics for which nodes to expand first?

- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

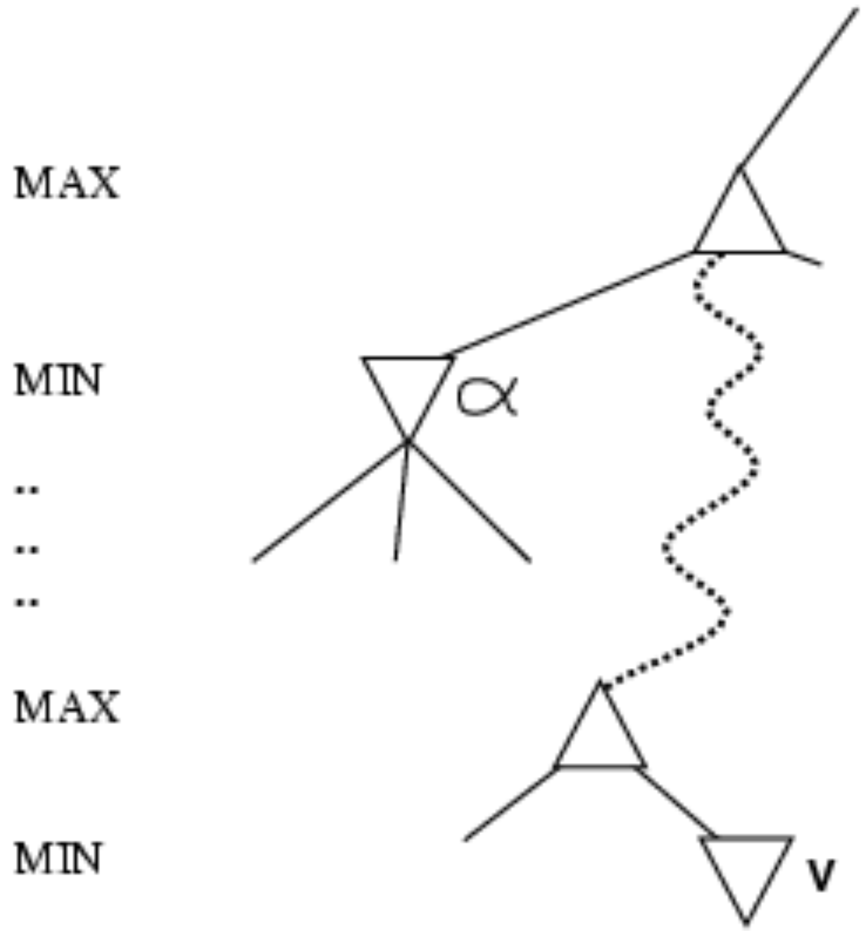
Why is it called α - β ?

- 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

Resource limits

The big problem is that the search space in typical games is very large.

Suppose we have 100 secs, explore 10^4 nodes/sec
→ 10^6 nodes per move

Standard approach:

- cutoff test:
  e.g., depth limit
- evaluation function
  = estimated desirability of position

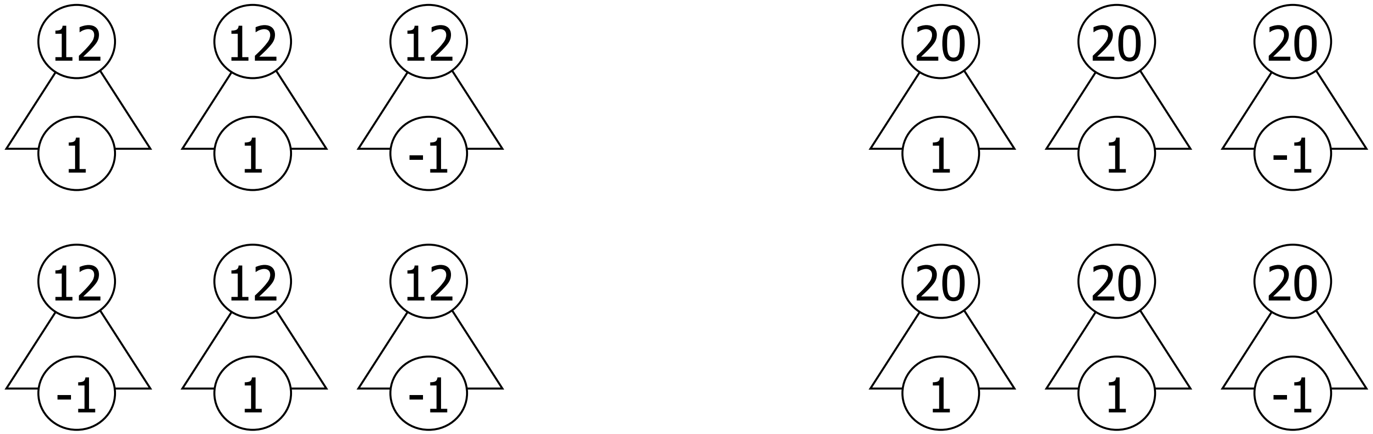
Evaluation functions

- 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.

Expected utility value



- A utility value may map to many states, each of which may lead to different terminal states
- Want utility values to model likelihood of better utility states.

Cutting off search

MinimaxCutoff is identical to *MinimaxValue* except

1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*

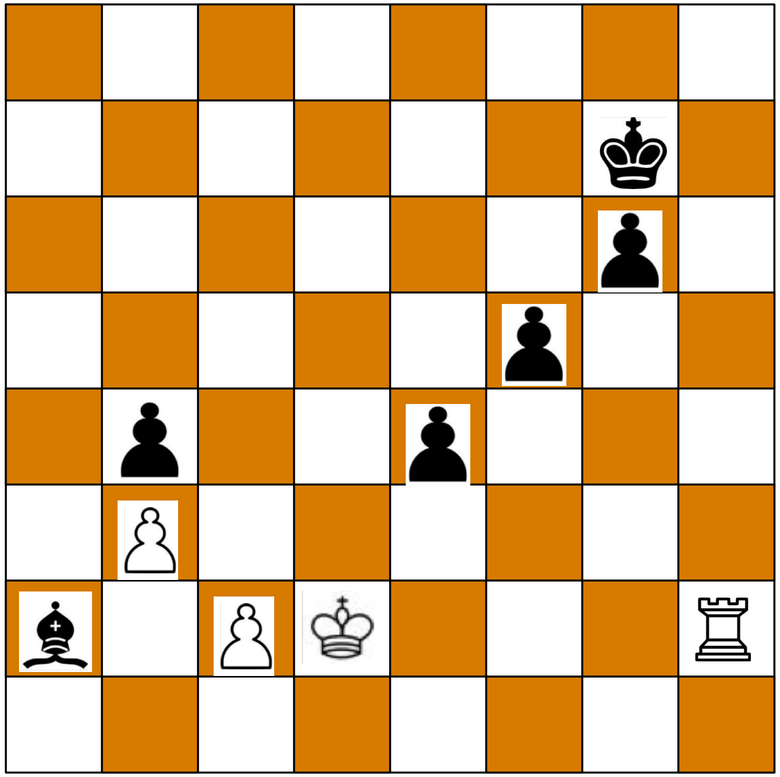
Does it work in practice?

PCs about 200m nodes per sec, b=35 → $2 \times 10^6 = 35^{m?}$
m = 5.2 plies

5-ply lookahead is still pretty easy to beat

- 4-ply ≈ human novice
- 8-ply ≈ human master
- 10-ply ≈ good PC program, with good pruning
- 12-ply ≈ Deep Blue, Gary Kasparov

Horizon Effect



Black will lose the bishop at some point forward

- An unavoidable damaging move is beyond the current search limit
- Add quiescence search
  - Evaluate moves that will make evaluation function fluctuate more wildly

The little black book

- Opening move search doesn't result in useful utility estimates
- Applying utility functions on end game scenarios may not solve the game
  - Western chess KRK end game

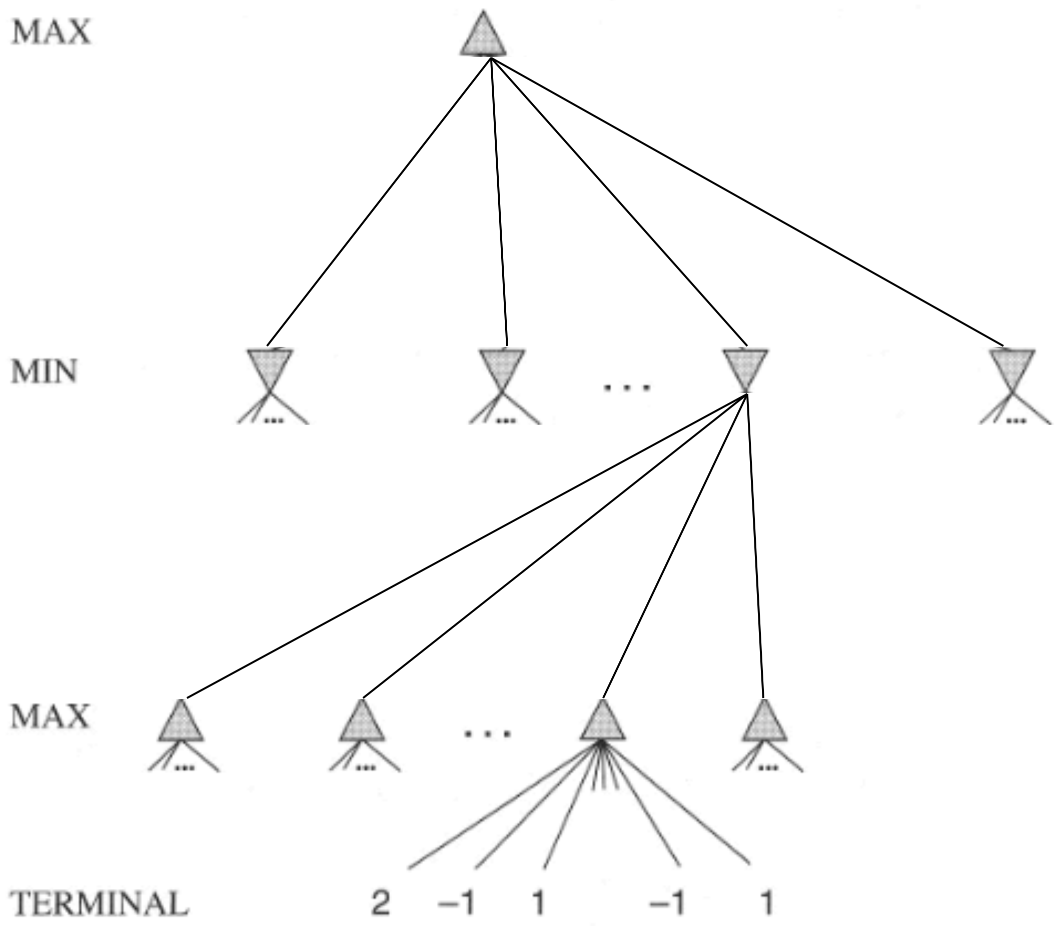
- Use a policy or lookup table (taken from previous game history)
- Similar to exploratory search (localization) encoding

Stochastic Games

- What about an element of chance?
- How do we deal with uncertainty?

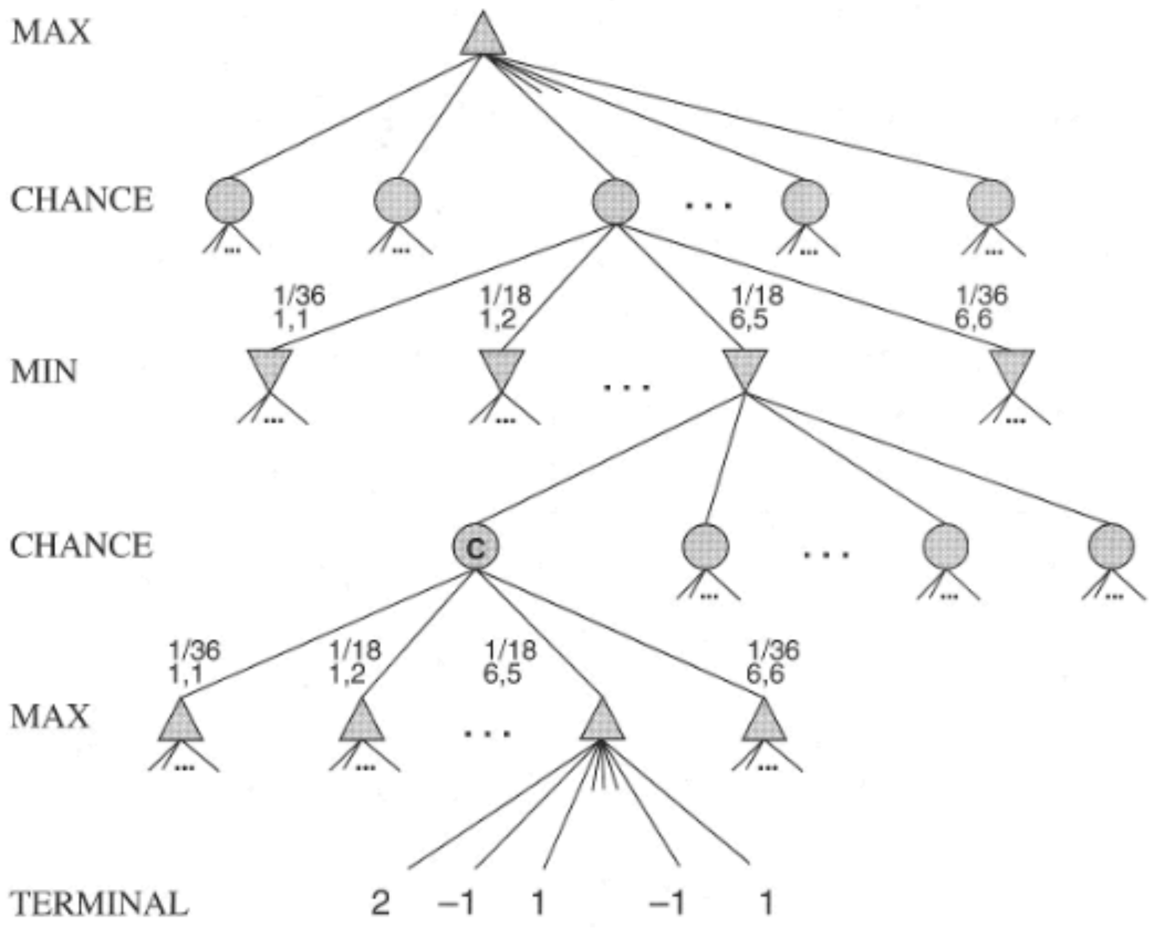
- Can we still use the minimax idea?

Adding chance layers



Calculate the expected value of a position

Adding chance layers



ExpectiMinimax: Calculate the expected value of a position

Deterministic games in practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.
- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- Othello: human champions refuse to compete against computers, who are too
- Go: human champions refuse to compete against computers, who are too In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

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