Artificial Intelligence 1: game playing

Lecturer: Tom Lenaerts

Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA)

Université Libre de Bruxelles

Outline

- What are games?
- Optimal decisions in games
  - **Which strategy leads to success?**
- α-β pruning
- Games of imperfect information
- Games that include an element of chance

What are and why study games?

- Games are a form of *multi-agent environment*
  - **What do other agents do and how do they affect our success?**
  - **Cooperative vs. competitive multi-agent environments.**
  - **Competitive multi-agent environments give rise to adversarial problems a.k.a. *games***
- Why study games?
  - **Fun; historically entertaining**
  - **Interesting subject of study because they are hard**
  - **Easy to represent and agents restricted to small number of actions**

Relation of Games to Search

- Search – no adversary
  - **Solution is (heuristic) method for finding goal**
  - **Heuristics and CSP techniques can find *optimal* solution**
  - **Evaluation function: estimate of cost from start to goal through given node**
  - **Examples: path planning, scheduling activities**
- Games – adversary
  - **Solution is strategy (strategy specifies move for every possible opponent reply).**
  - **Time limits force an *approximate* solution**
  - **Evaluation function: evaluate "goodness" of game position**
  - **Examples: chess, checkers, Othello, backgammon**

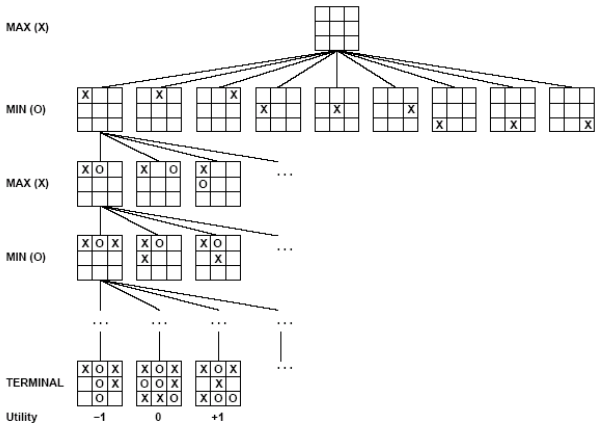
Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information		bridge, poker, scrabble nuclear war

Game setup

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over. Winner gets award, looser gets penalty.
- Games as search:
  - **Initial state: e.g. board configuration of chess**
  - **Successor function: list of (move,state) pairs specifying legal moves.**
  - **Terminal test: Is the game finished?**
  - **Utility function: Gives numerical value of terminal states. E.g. win (+1), loose (-1) and draw (0) in tic-tac-toe (next)**
- MAX uses search tree to determine next move.

Partial Game Tree for Tic-Tac-Toe



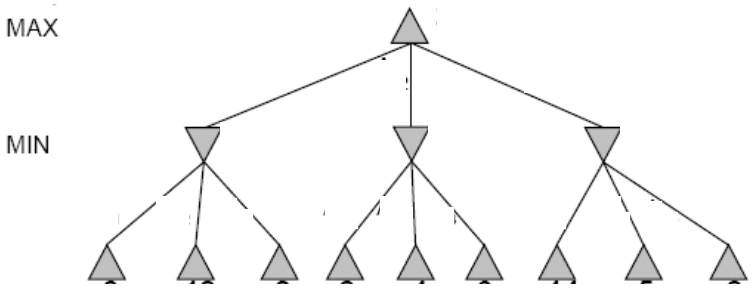
Optimal strategies

- Find the contingent *strategy* for MAX assuming an infallible MIN opponent.
- Assumption: Both players play optimally !!
- Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

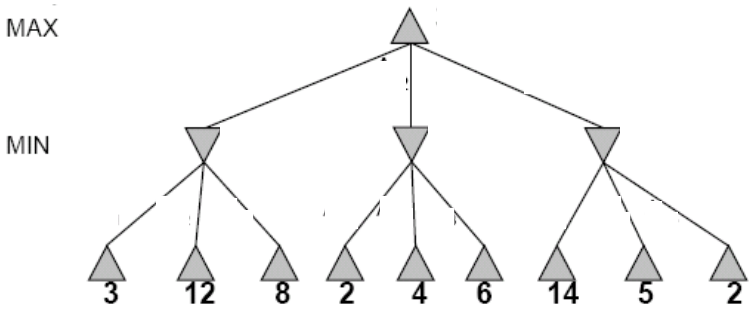
MINIMAX-VALUE(n)=

UTILITY(n)	If n is a terminal
$\max_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a max node
$\min_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a max node

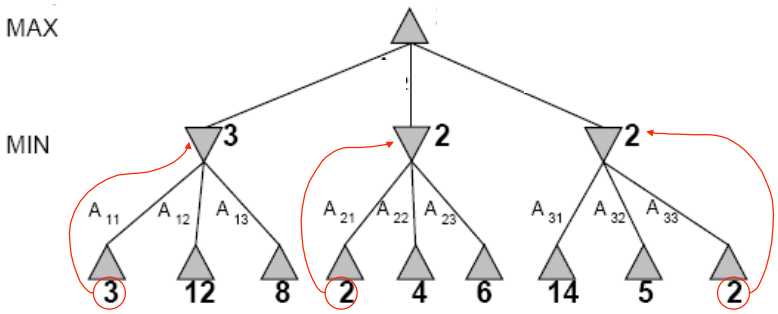
Two-Ply Game Tree



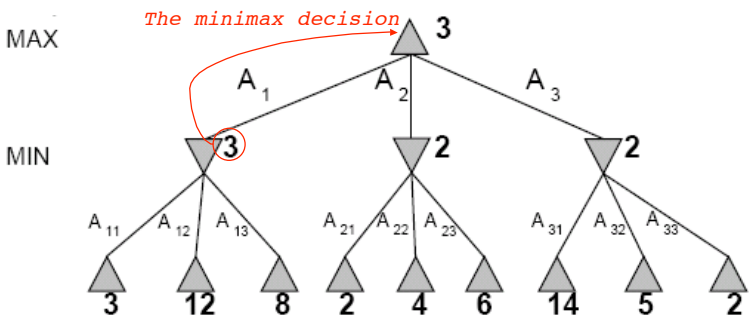
Two-Ply Game Tree



Two-Ply Game Tree



Two-Ply Game Tree



Minimax maximizes the worst-case outcome for max.

What if MIN does not play optimally?

- Definition of optimal play for MAX assumes MIN plays optimally: maximizes worst-case outcome for MAX.
- But if MIN does not play optimally, MAX will do even better. [Can be proved.]

Minimax Algorithm

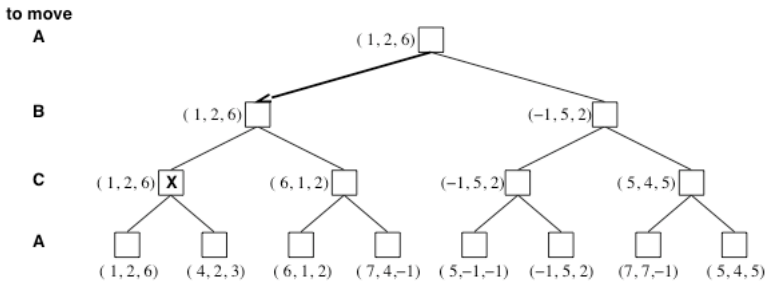
```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
  v←MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v
---------------------------------------------------------
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) 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

Criterion	Minimax
Complete?	Yes ☺
Time	$O(b^m)$ ☹
Space	O(bm) ☺
Optimal?	Yes ☹

Multiplayer games

- Games allow more than two players
- Single minimax values becomes vector

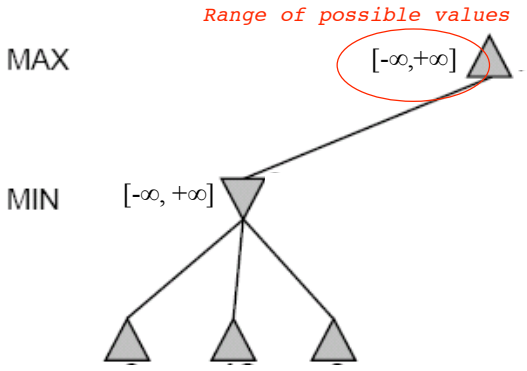


Problem of minimax search

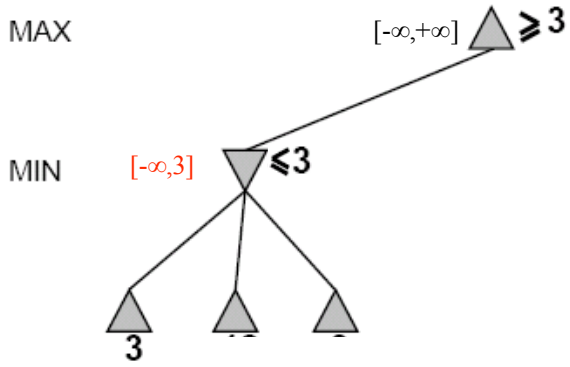
- Number of games states is exponential to the number of moves.
  - **Solution: Do not examine very node**
  - **==> Alpha-beta pruning**
    - Alpha = value of best choice found so far at any choice point along the MAX path
    - Beta = value of best choice found so far at any choice point along the MIN path
- Revisit example …

Alpha-Beta Example

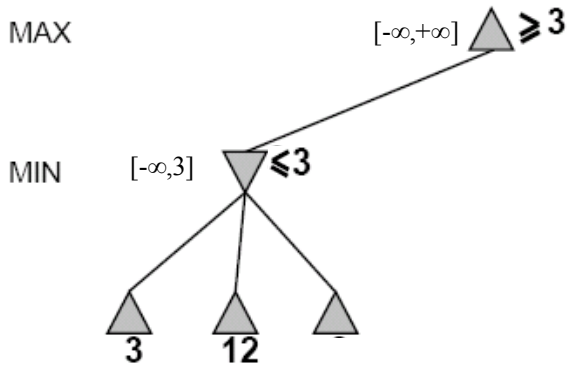
Do DF-search until first leaf



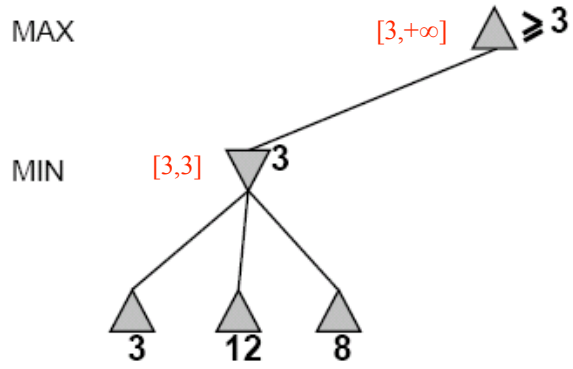
Alpha-Beta Example (continued)



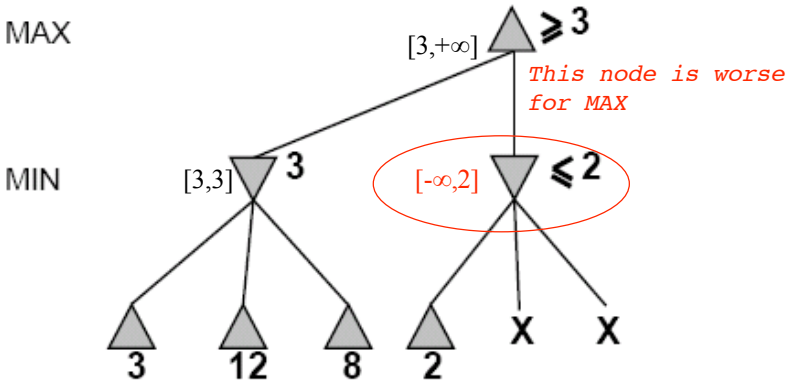
Alpha-Beta Example (continued)



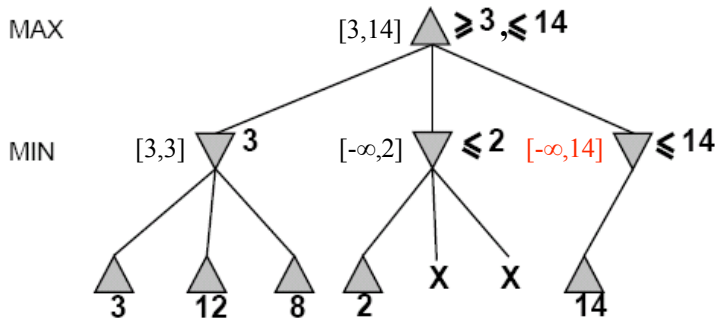
Alpha-Beta Example (continued)



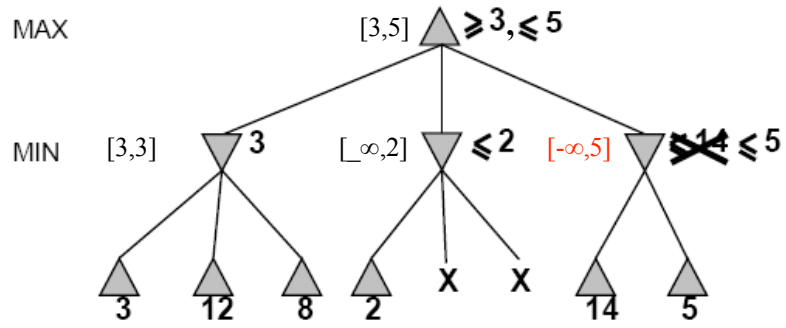
Alpha-Beta Example (continued)



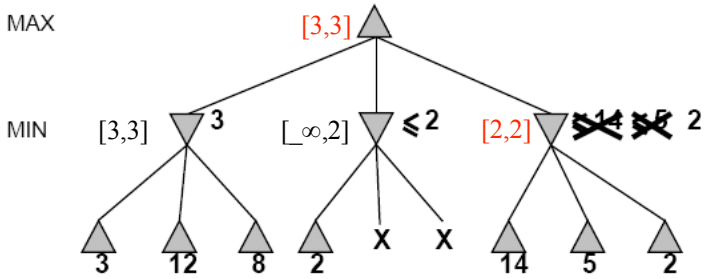
Alpha-Beta Example (continued)



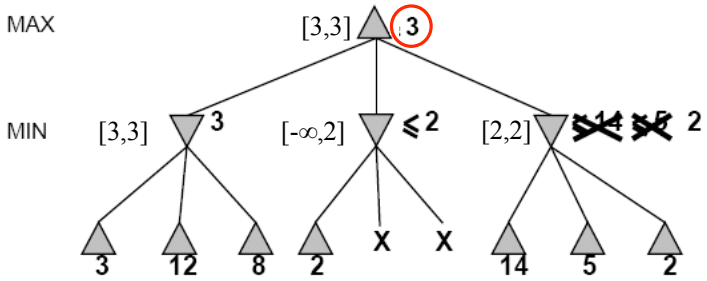
Alpha-Beta Example (continued)



Alpha-Beta Example (continued)



Alpha-Beta Example (continued)



Alpha-Beta Algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return the *action* in SUCCESSORS(*state*) with value *v*

---

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

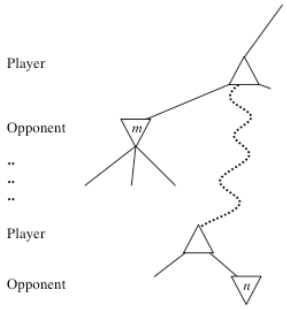
Alpha-Beta Algorithm

---

function MIN-VALUE(*state*, α , β) **returns** *a utility value*
  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

General alpha-beta pruning

- Consider a node *n* somewhere in the tree
- If player has a better choice at
  - **Parent node of n**
  - **Or any choice point further up**
- *n* will **never** be reached in actual play.
- Hence when enough is known about *n*, it can be pruned.



Final Comments about Alpha-Beta Pruning

- Pruning does not affect final results
- Entire subtrees can be pruned.
- Good move *ordering* improves effectiveness of pruning
- With "perfect ordering," time complexity is $O(b^{m/2})$
  - **Branching factor of sqrt(b) !!**
  - **Alpha-beta pruning can look twice as far as minimax in the same amount of time**
- Repeated states are again possible.
  - **Store them in memory = transposition table**

Games of imperfect information

- Minimax and alpha-beta pruning require to much leaf-node evaluations.
- May be impractical within a reasonable amount of time.
- SHANNON (1950):
  - **Cut off search earlier (replace TERMINAL-TEST by CUTOFF-TEST)**
  - **Apply heuristic evaluation function EVAL (replacing utility function of alpha-beta)**

Cutting off search

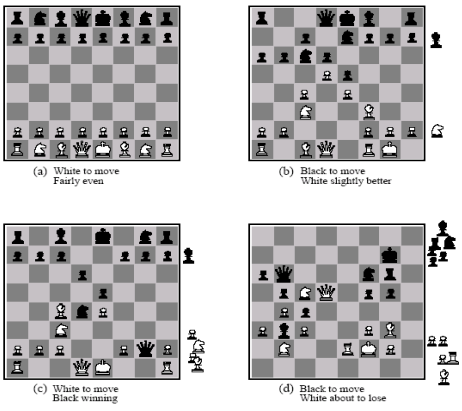
- Change:
  - **if TERMINAL-TEST(*state*) then return UTILITY(*state*)**

  into
  - **if CUTOFF-TEST(*state*,*depth*) then return EVAL(*state*)**
- Introduces a fixed-depth limit *depth*
  - **Is selected in so that the amount of time will not exceed what the rules of the game allow.**
- When cuttoff occurs, the evaluation is performed.

Heuristic EVAL

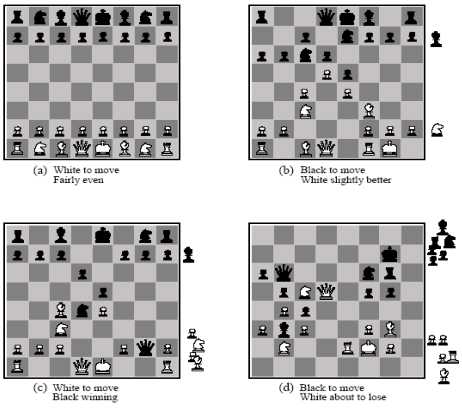
- Idea: produce an estimate of the expected utility of the game from a given position.
- Performance depends on quality of EVAL.
- Requirements:
  - **EVAL should order terminal-nodes in the same way as UTILITY.**
  - **Computation may not take too long.**
  - **For non-terminal states the EVAL should be strongly correlated with the actual chance of winning.**
- Only useful for quienscent (no wild swings in value in near future) states

Heuristic EVAL example



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

Heuristic EVAL example

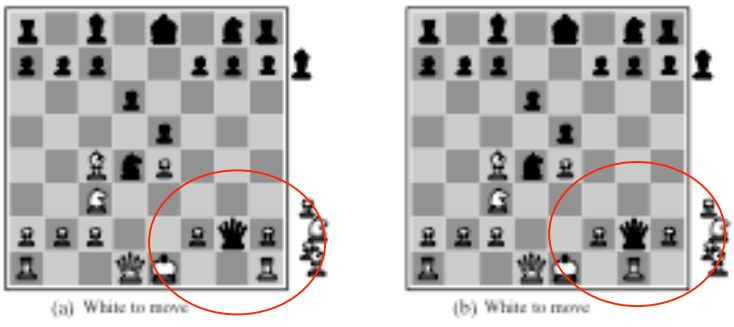


Addition assumes independence

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

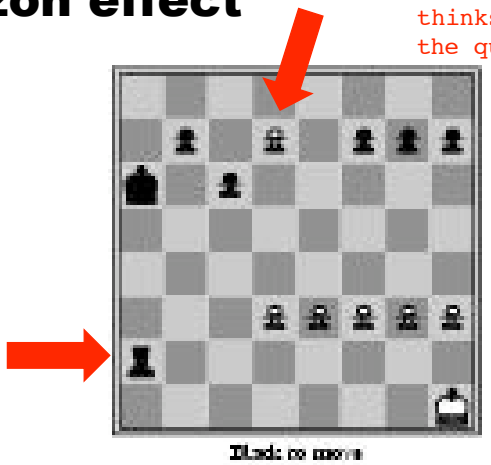
Heuristic difficulties

Heuristic counts pieces won

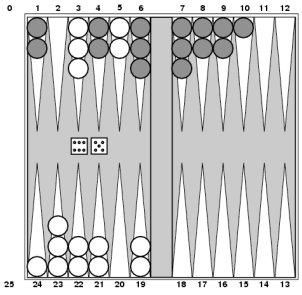


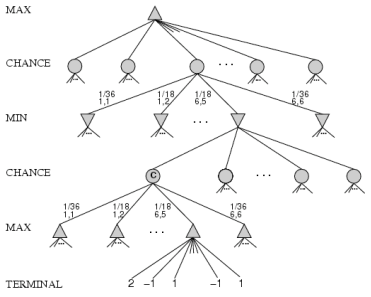
Horizon effect

Fixed depth search thinks it can avoid the queening move



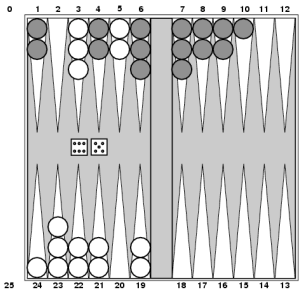
Games that include chance

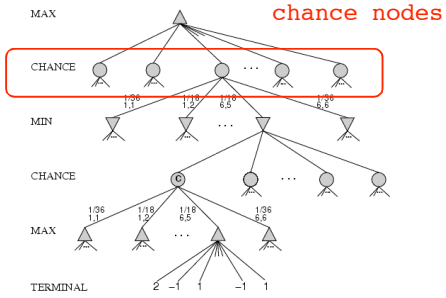




- Possible moves (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)

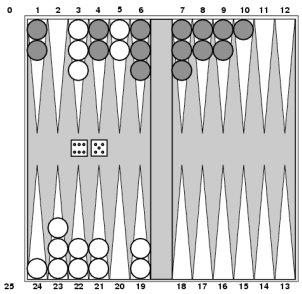
Games that include chance

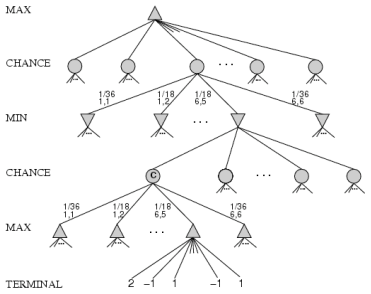




- Possible moves (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)
- [1,1], [6,6] chance 1/36, all other chance 1/18

Games that include chance





- [1,1], [6,6] chance 1/36, all other chance 1/18
- Can not calculate definite minimax value, only *expected* value

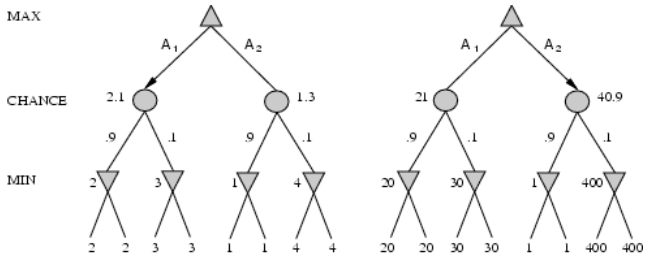
Expected minimax value

EXPECTED-MINIMAX-VALUE(n)=

UTILITY(n)	If n is a terminal
$\max_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a max node
$\min_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a max node
$\sum_{s \in successors(n)} P(s)$. EXPECTEDMINIMAX(s)	If n is a chance node

These equations can be backed-up recursively all the way to the root of the game tree.

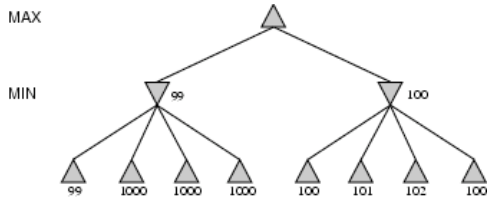
Position evaluation with chance nodes



- Left, A1 wins
- Right A2 wins
- Outcome of evaluation function may not change when values are scaled differently.
- Behavior is preserved only by a *positive linear* transformation of EVAL.

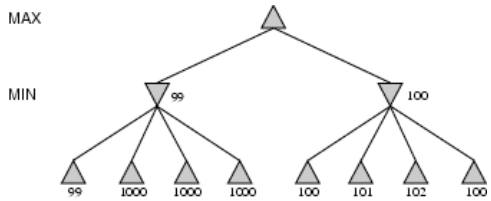
Discussion

- Examine section on state-of-the-art games yourself
- Minimax assumes right tree is better than left, yet …
  - **Return probability distribution over possible values**
  - **Yet expensive calculation**



Discussion

- Utility of node expansion
  - **Only expand those nodes which lead to significanlty better moves**
- Both suggestions require meta-reasoning



Summary

- Games are fun (and dangerous)
- They illustrate several important points about AI
  - **Perfection is unattainable -> approximation**
  - **Good idea what to think about**
  - **Uncertainty constrains the assignment of values to states**
- Games are to AI as grand prix racing is to automobile design.