Artificial Intelligence- Games

Amir Aavani Amir@Aavani.net
http://Amir.Aavani.net/CS/AI-86

27. November 2007

Outline

1. Games
2. Game playing strategies
   - minimax decisions
   - α-β pruning
3. Games that includes chance
4. Games of imperfect information

Games

1. Adversarial Search Problem: Competitive environments in which the agents' goals are in conflict.

Games

1. Adversarial Search Problem: Competitive environments in which the agents' goals are in conflict.
2. Examples: Chess, Checkers, Backgammon, ...

Games

1. Adversarial Search Problem: Competitive environments in which the agents' goals are in conflict.
2. Examples: Chess, Checkers, Backgammon, ...
3. Games in AI are usually:
    1. Deterministic.

Games

1. Adversarial Search Problem: Competitive environments in which the agents' goals are in conflict.
2. Examples: Chess, Checkers, Backgammon, ...
3. Games in AI are usually:
   1. Deterministic.
   2. Fully observable.

Games

1. Adversarial Search Problem: Competitive environments in which the agents' goals are in conflict.
2. Examples: Chess, Checkers, Backgammon, ...
3. Games in AI are usually:
   1. Deterministic.
   2. Fully observable.
   3. There are two agents whose actions must alternate.

Games

1. Adversarial Search Problem: Competitive environments in which the agents' goals are in conflict.
2. Examples: Chess, Checkers, Backgammon, ...
3. Games in AI are usually:
   1. Deterministic.
   2. Fully observable.
   3. There are two agents whose actions must alternate.
   4. Utility values at end of game are equal and opposite.

Games, Cont

1. State of games is easy to represent.

Games, Cont

1. State of games is easy to represent.
2. Actions are usually limited and simple.
3. Actions' effects are defined by precise rules.

Games, Cont

1. State of games is easy to represent.
2. Actions are usually limited and simple.
3. Actions' effects are defined by precise rules.
4. Games are (too) hard toy problem.
5. Games usually have a huge state space (chess: 35^{100})

Optimal Strategy

1. Complete game tree is usually complex.

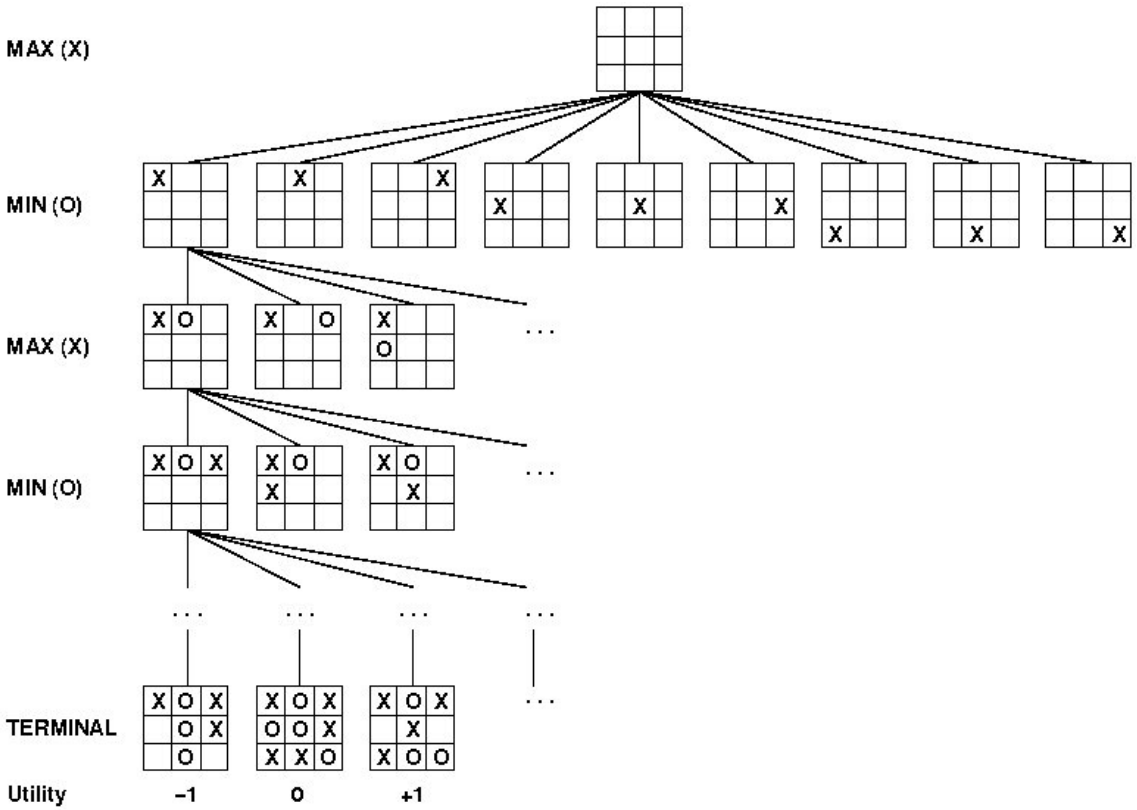
Optimal Strategy

1. Complete game tree is usually complex.
2. The max(min) player wants to maximize(minimize) the utility.

Optimal Strategy

1. Complete game tree is usually complex.
2. The max(min) player wants to maximize(minimize) the utility.
3. The minmax strategy is an optimal strategy for deterministic games with complete information.

Optimal Strategy, Cont



Optimal Strategy, Cont

MINMAX-VALUE (n)=

Optimal Strategy, Cont

MINMAX-VALUE (n)=

UTILITY (n) if n is a terminal state

Optimal Strategy, Cont

MINMAX-VALUE (n)=

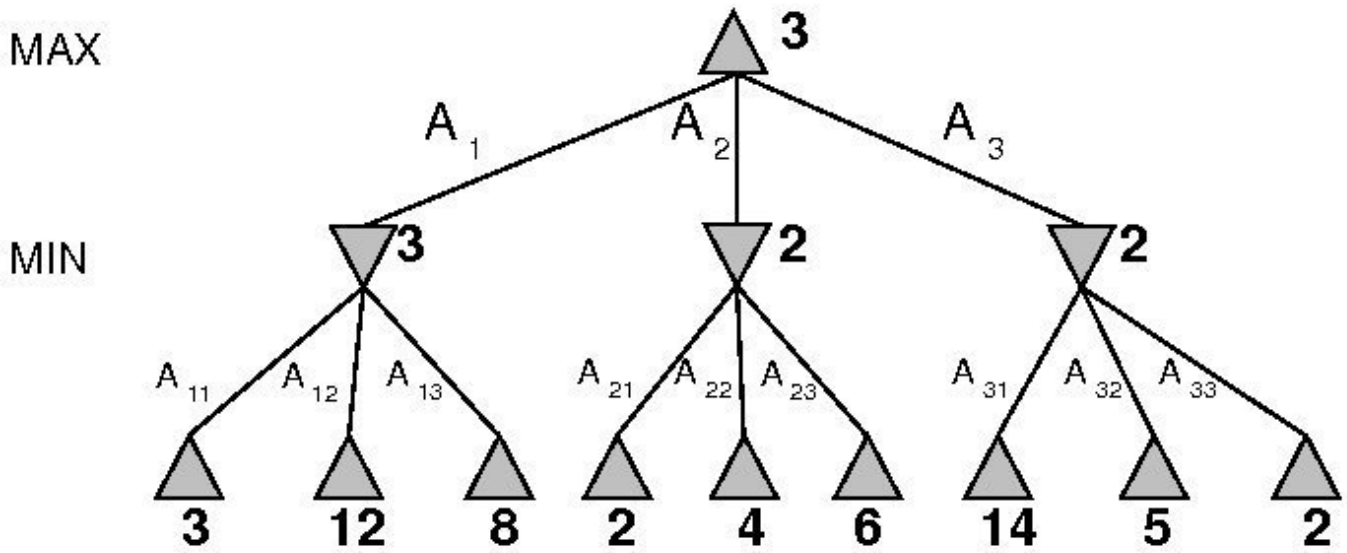
UTILITY (n)	if n is a terminal state
$max_{s \in Succ(n)}$MINMAX-VALUE (s)	if n is a max node

Optimal Strategy, Cont

MINMAX-VALUE (n)=

UTILITY (n)	if n is a terminal state
$max_{s \in Succ(n)}$MINMAX-VALUE (s)	if n is a max node
$min_{s \in Succ(n)}$MINMAX-VALUE (s)	if n is a min node

Optimal Strategy, Cont



Optimal Strategy, Cont

```
function ALPHA-BETA-DECISION (state) returns an action
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESU

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

Optimal Strategy, Cont

Complete:

Optimal Strategy, Cont

Complete: Yes, if tree is finite.

Optimal Strategy, Cont

Complete: Yes, if tree is finite.

Optimal:

Optimal Strategy, Cont

Complete: Yes, if tree is finite.

Optimal: Yes (Against optimal opponent), Otherwise ?

Optimal Strategy, Cont

Complete: Yes, if tree is finite.

Optimal: Yes (Against optimal opponent), Otherwise ?

Time complexity:

Optimal Strategy, Cont

Complete: Yes, if tree is finite.

Optimal: Yes (Against optimal opponent), Otherwise ?

Time complexity: b^m

Optimal Strategy, Cont

Complete: Yes, if tree is finite.

Optimal: Yes (Against optimal opponent), Otherwise ?

Time complexity: b^m

Space complexity:

Optimal Strategy, Cont

Complete: Yes, if tree is finite.

Optimal: Yes (Against optimal opponent), Otherwise ?

Time complexity: b^m

Space complexity: bm

Optimal Strategy, Cont

1. The idea can be extended for multi-player games.

Optimal Strategy, Cont

1. The idea can be extended for multi-player games.
2. Minmax is an optimal strategy but needs exponential time.

Optimal Strategy, Cont

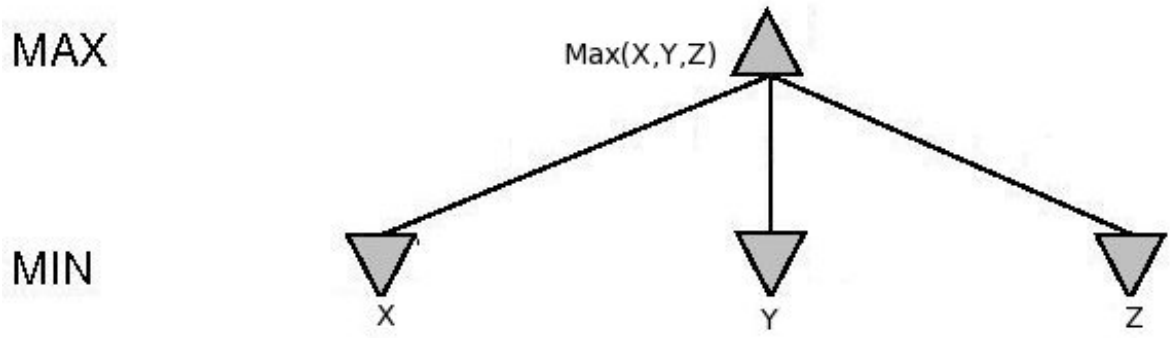
1. The idea can be extended for multi-player games.
2. Minmax is an optimal strategy but needs exponential time.
3. Is it necessary to visit all nodes in tree?

Optimal Strategy, Cont

1. The idea can be extended for multi-player games.
2. Minmax is an optimal strategy but needs exponential time.
3. Is it necessary to visit all nodes in tree?
4. Can the tree be pruned?

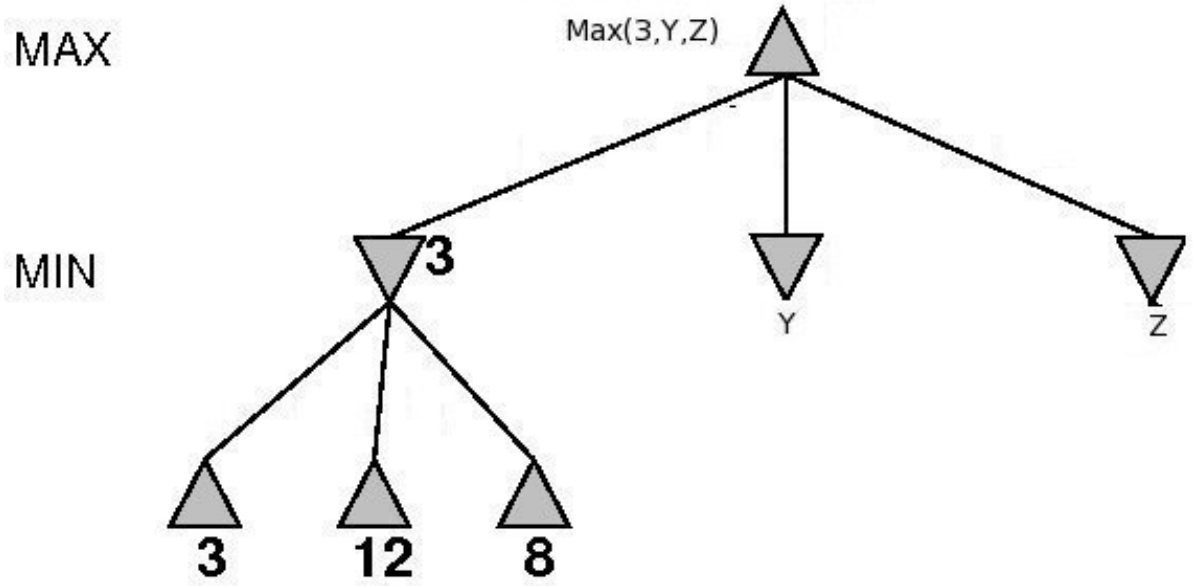
Alpha-Beta Pruning

To calculate minmax root we should exand all its children.



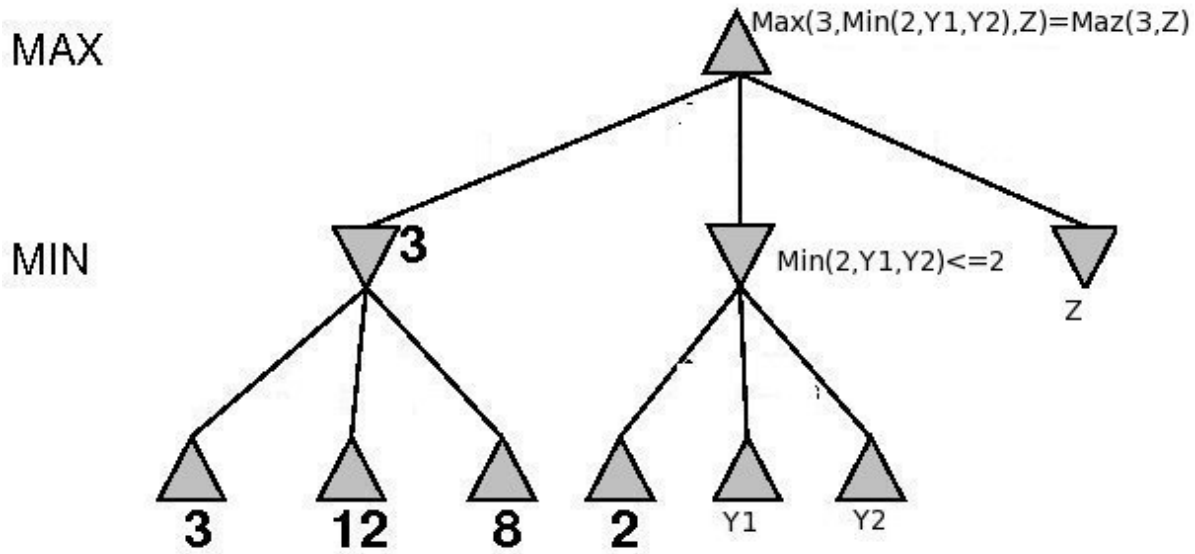
Alpha-Beta Pruning, Cont

To calculate minmax(X) we should exand all its children.

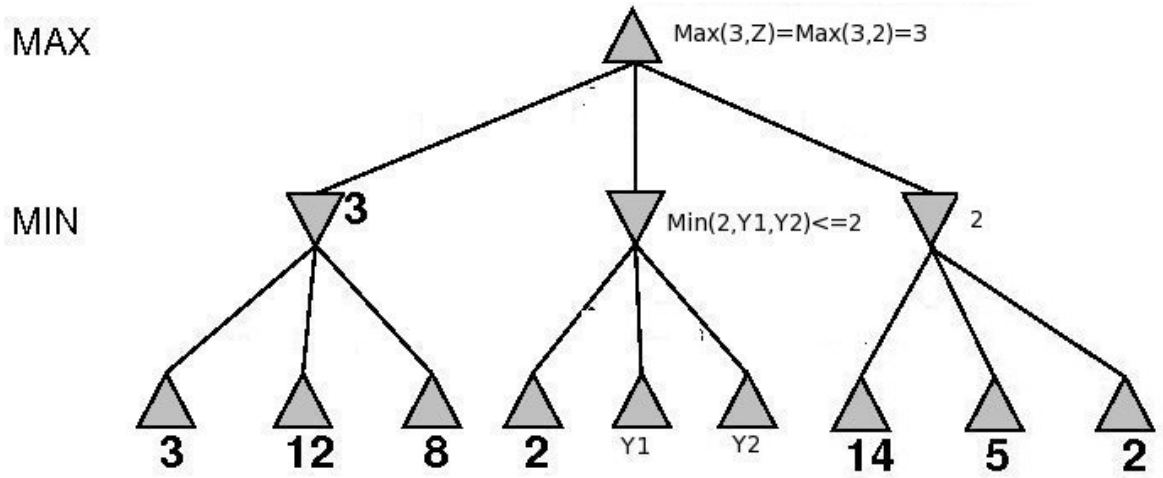


Alpha-Beta Pruning, Cont

To calculate minmax(Y), knowing minmax(X), only the first child is visited.



Alpha-Beta Pruning, Cont



Alpha-Beta Pruning, Cont

1. Alpha-Beta pruning can be applied to trees of any depth.

Alpha-Beta Pruning, Cont

1. Alpha-Beta pruning can be applied to trees of any depth.
2. Alpha-Beta pruning can be prune a subtree.

Alpha-Beta Pruning, Cont

1. Alpha-Beta pruning can be applied to trees of any depth.
2. Alpha-Beta pruning can be prune a subtree.
3. $\alpha =$ the value of best(i.e., highest value) choice we have found so far at any choice point along the path for MAX.

Alpha-Beta Pruning, Cont

1. Alpha-Beta pruning can be applied to trees of any depth.
2. Alpha-Beta pruning can be prune a subtree.
3. $\alpha =$ the value of best(i.e., highest value) choice we have found so far at any choice point along the path for MAX.
4. $\beta =$ the value of best(i.e., lowest value) choice we have found so far at any choice point along the path for MIN.

Alpha-Beta Pruning, Cont

1. Alpha-Beta pruning can be applied to trees of any depth.
2. Alpha-Beta pruning can be prune a subtree.
3. $\alpha =$ the value of best(i.e., highest value) choice we have found so far at any choice point along the path for MAX.
4. $\beta =$ the value of best(i.e., lowest value) choice we have found so far at any choice point along the path for MIN.
5. Alpha-Beta updates the value of α and β as it goes along.

Alpha-Beta Pruning, Cont

1. The performance is depend on order of childern (successor function).

Alpha-Beta Pruning, Cont

1. The performance is depend on order of childern (successor function).
2. In best case, alpha-beta pruning needs to explore $b^{\frac{m}{2}}$ nodes instead of b^m nodes.

Alpha-Beta Pruning, Cont

1. The performance is depend on order of childern (successor function).
2. In best case, alpha-beta pruning needs to explore $b^{\frac{m}{2}}$ nodes instead of b^m nodes.
3. The effective branching factor becomes $\sqrt{b}$ instead of b.
4. For chess, 6 instead of 35.

Alpha-Beta Pruning, Cont

```
function ALPHA-BETA-DECISION (state) returns an action
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESU

function MAX-VALUE (state,a,b) returns a utility value
  if TERMINAL-TEST (state) then return UTILITY (state)
  v= -Infty
  for a, s in SUCCESSORS(state) do
    v= MAX(v, MIN-VALUE(s,a,b))
    if b<= v then return v
    a=MAX(a,v)
  return v

function MIN-VALUE (state) returns a utility value
    ??
```

Alpha-Beta Pruning, Cont

1. Pruning does not effect final result.

Alpha-Beta Pruning, Cont

1. Pruning does not effect final result.
2. Good move ordering improves effectiveness of pruning.

Alpha-Beta Pruning, Cont

1. Pruning does not effect final result.
2. Good move ordering improves effectiveness of pruning.
3. With the best ordering, time complexity= $O(b^{\frac{m}{2}})$.

Alpha-Beta Pruning, Cont

1. Pruning does not effect final result.
2. Good move ordering improves effectiveness of pruning.
3. With the best ordering, time complexity= $O(b^{\frac{m}{2}})$.
4. In chess, 35^{50} is still too big!.

Real-Time Decision Making

1. Modify minmax in the following ways:
   - Utility func. is replaced by a heuristic **evaluation function**. It try to estimate the desirability of state.

Real-Time Decision Making

1. Modify minmax in the following ways:
   - Utility func. is replaced by a heuristic **evaluation function**. It try to estimate the desirability of state.
   - Terminal test is replaced by **cutoff test** that decides when to apply EVAL.

Evaluation functions

1. are used by humans.

Evaluation functions

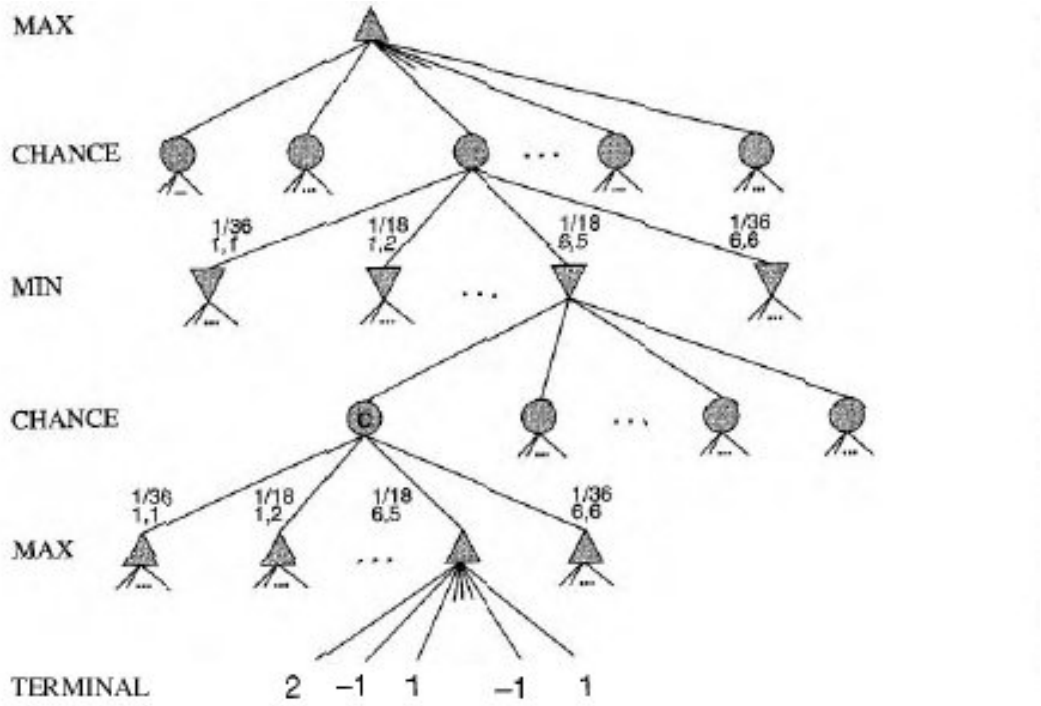
1. are used by humans.
2. can be in form of $Eval(s) = w_1 f_1(s) + w_1 f_1(s) + \cdots + w_n f_n(s)$.
   e.g. $w_1 = 5$ and
   $f_1(s)$=(number of white rooks)- (number of black rooks)

Nondeterministic games

1. Some games include an element of chance, dice, card shuffling, etc.

Nondeterministic games

1. Some games include an element of chance, dice, card shuffling, etc.
2. The game tree for this games are:



Nondeterministic games, Cont

1. EXPECTMINMAX (s)

Nondeterministic games, Cont

1. EXPECTMINMAX (s)

   $=$ s is MAX node $\Rightarrow$ return max(EXPECTMINMAX(succ(s))

Nondeterministic games, Cont

1. EXPECTMINMAX (s)
   = s is MAX node $\Rightarrow$ return max(EXPECTMINMAX(succ(s))
   = s is MIN node $\Rightarrow$ return min(EXPECTMINMAX(succ(s))

Nondeterministic games, Cont

1. EXPECTMINMAX (s)
   - = s is MAX node $\Rightarrow$ return max(EXPECTMINMAX(succ(s))
   - = s is MIN node $\Rightarrow$ return min(EXPECTMINMAX(succ(s))
   - = s is chance node $\Rightarrow$ return avr(EXPECTMINMAX(succ(s))

Nondeterministic games, Cont

1. EXPECTMINMAX (s)
   - = s is MAX node $\Rightarrow$ return max(EXPECTMINMAX(succ(s))
   - = s is MIN node $\Rightarrow$ return min(EXPECTMINMAX(succ(s))
   - = s is chance node $\Rightarrow$ return avr(EXPECTMINMAX(succ(s))
2. Time Complexity: $O(b^m n^m)$.

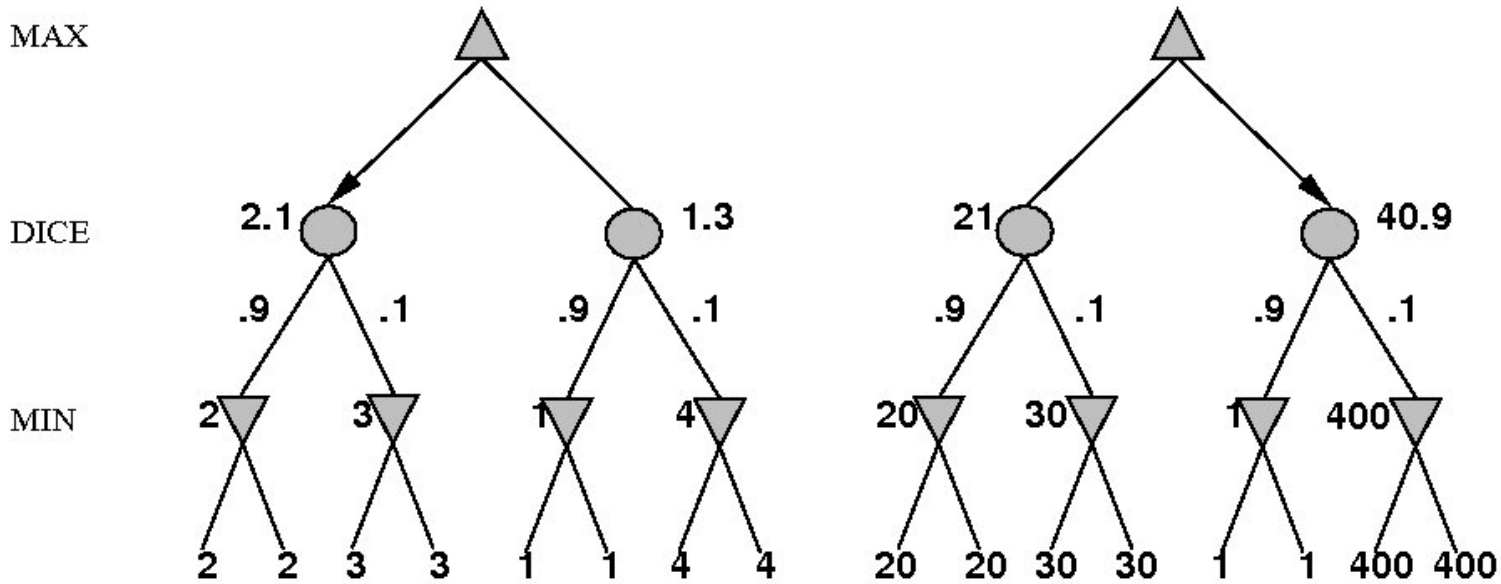
Nondeterministic games, Cont

1. EXPECTMINMAX (s)
   = s is MAX node $\Rightarrow$ return max(EXPECTMINMAX(succ(s))
   = s is MIN node $\Rightarrow$ return min(EXPECTMINMAX(succ(s))
   = s is chance node $\Rightarrow$ return avr(EXPECTMINMAX(succ(s))
2. Time Complexity: $O(b^m n^m)$.
3. Alpha-beta pruning can be used.

Nondeterministic games, Cont

1. EXPECTMINMAX (s)
   = s is MAX node $\Rightarrow$ return max(EXPECTMINMAX(succ(s))
   = s is MIN node $\Rightarrow$ return min(EXPECTMINMAX(succ(s))
   = s is chance node $\Rightarrow$ return avr(EXPECTMINMAX(succ(s))
2. Time Complexity: $O(b^m n^m)$.
3. Alpha-beta pruning can be used.
4. Eval function can be used here.

Nondeterministic games, Cont



Nondeterministic games, Cont

1. Backgommon: 20 legal moves.
2. Throwing two dices has 21 outcomes.

Nondeterministic games, Cont

1. Backgommon: 20 legal moves.
2. Throwing two dices has 21 outcomes.
3. Thinking in depth 4: $20 * (21 * 20)^3 \approx 1.2 * 10^9$

Nondeterministic games, Cont

1. Backgommon: 20 legal moves.
2. Throwing two dices has 21 outcomes.
3. Thinking in depth 4: $20 * (21 * 20)^3 \approx 1.2 * 10^9$
4. TDGammon uses depth-2 search+ very good Eval $\approx$ world-champion level.

Games of imperfect information

1. E.g., Card games, where opponent's initial cards are unknown.

Games of imperfect information

1. E.g., Card games, where opponent's initial cards are unknown.
2. A prob. for each opponent possible deal can be calculated.

Games of imperfect information

1. E.g., Card games, where opponent's initial cards are unknown.
2. A prob. for each opponent possible deal can be calculated.
3. Then, compute the minmax value of each action in each deal.