Intelligent Autonomous Agents

and Cognitive Robotics:

Adversarial Agents

Ralf Möller, Rainer Marrone
Hamburg University of Technology

Adversarial Agents

- In this chapter we cover **competitive environments**, in which the agents goals are in conflict, giving rise to adversarial search problems often known as games.
- Mathematical game theory, a branch of economics, views any multi-agent environment as a game, regardless of whether the agents are cooperative or competitive.

2

Multi-Agent Games

- Agents must ***anticipate*** what other agents do
- Criteria:
  - **Abstraction**: To describe a game we must capture every *relevant* aspect of the game.
  - **Accessible environments:** Such games are characterized by perfect information
  - **Search:** game-playing then consists of a search through possible game positions *with actions of other agents*
  - **Unpredictable opponent**: introduces **uncertainty** thus game-playing must deal with **contingency problems**

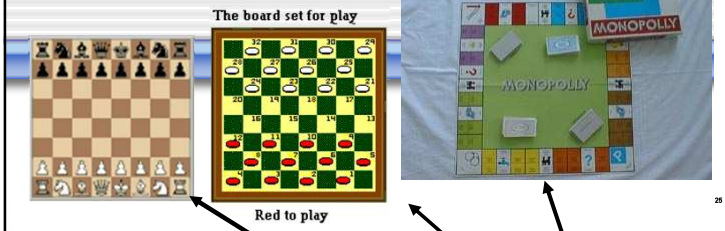
3

Two-player games

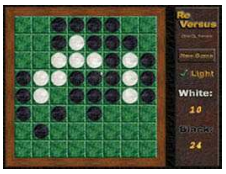
- A game formulated as a *search problem*:
  - *Initial state*: board position and turn
  - *Actions/Transition model*: definition of legal moves
  - *Terminal state*: conditions for when game is over
  - ***Utility function***:
    a numeric value that describes the outcome of the game. E.g., -1, 0, 1 for loss, draw, win (AKA **payoff function**)

4

Type of games

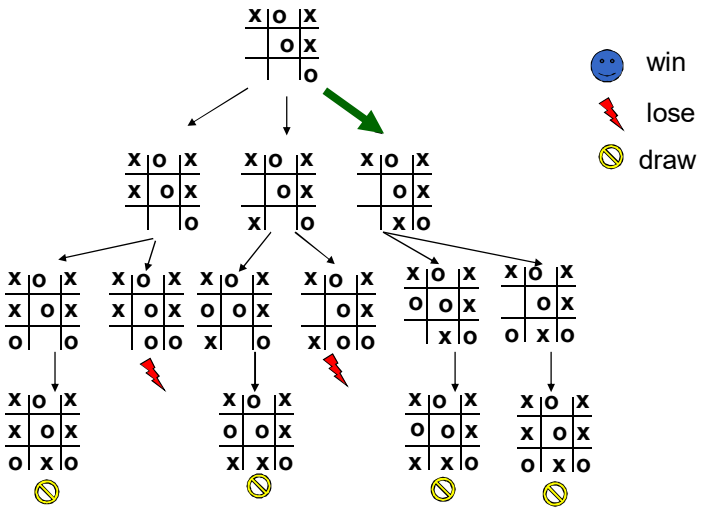


	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	Battleship	bridge, poker, scrabble



5

What is a good move?



6

The minimax algorithm

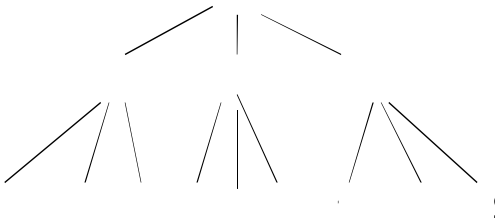
- Perfect play for deterministic environments with perfect information
- **Basic idea:** choose move with highest minimax value
  = best achievable payoff against **best play**
- **Algorithm:**
  1. Generate game tree completely
  2. Determine utility of each terminal state
  3. Propagate the utility values upward in the tree by applying MIN and MAX operators on the nodes in the current level
  4. At the root node use minimax decision to select the move with the max (of the min) utility value

7

Minimax algorithm

MAX

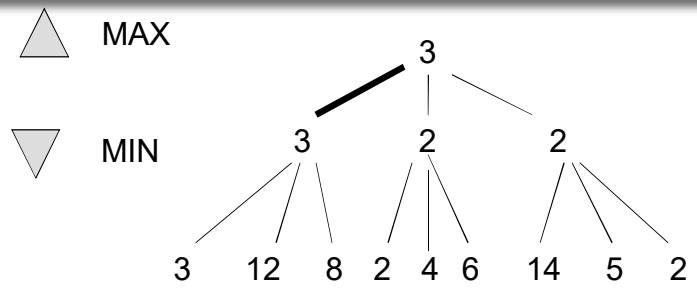
MIN



- Minimize opponent's chance
- Maximize your chance

8

Minimax



- Minimize opponent's chance
- Maximize your chance

$$\text{MINIMAX-VALUE}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in Successors(n)} \text{MINIMAX-VALUE}(s) & \text{if } n \text{ is a MIN node.} \end{cases}$$

9

Minimax: Recursive implementation

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

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

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

Complete: Yes, for finite state-space **Time complexity:** $O(b^m)$
Optimal: Yes, if winning is the goal **Space complexity:** O(bm) or O(m)

10

Game vs. search problem

- Unpredictable opponent → contingency plan (MINIMAX assumes best playing opponent)
- Time limits → cannot explore complete state space, approximate

- Pruning (McCarthy, 1956)
- Finite horizon, approximate (Zuse, 1945; Shannon 1950,…)

11

Searching for the next move

- **Complexity:** many games have a huge search space
  - **Chess:** $b = 35, m=100 \Rightarrow nodes = 35^{100}$ means more than 10^{154} in a search tree and more than 10^{40} nodes in a search graph. Take several millennia to compute moves. $35^{100} = 10^{log(35^{100})} = 10^{100*log(35)} = 10^{100*1,54} = 10^{154}$
- **Resource (e.g., time, memory) limit:** optimal solution not feasible/possible, thus must approximate

1. **Pruning:** makes the search more efficient by discarding portions of the search tree that cannot improve quality.
2. **Evaluation functions:** heuristics to evaluate utility of a state without exhaustive search.

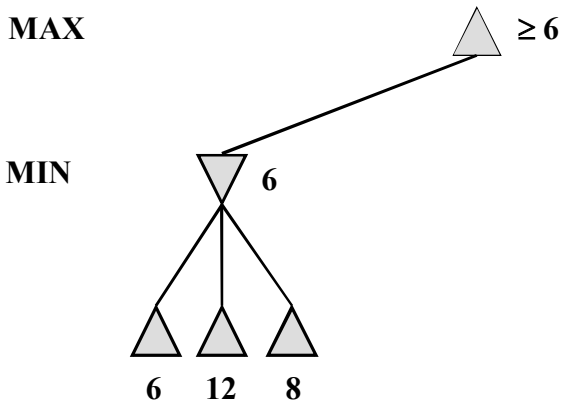
12

1. α-β pruning

- **α-β pruning:** the basic idea is to prune portions of the search tree that cannot improve the utility value of the *max* or *min* node, by just considering the values of nodes seen so far.
- Does it work? Yes, it roughly cuts the branching factor from b to √b resulting in double as far look-ahead than pure minimax.

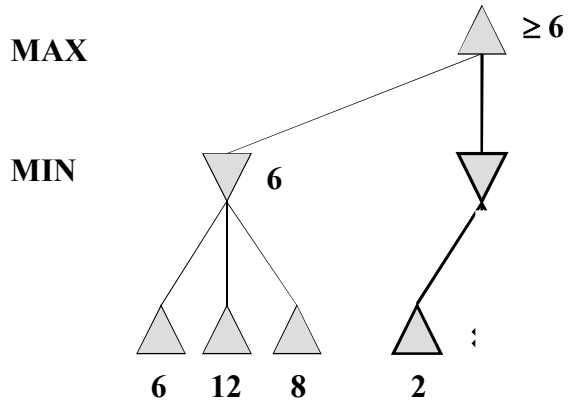
13

α-β pruning: example



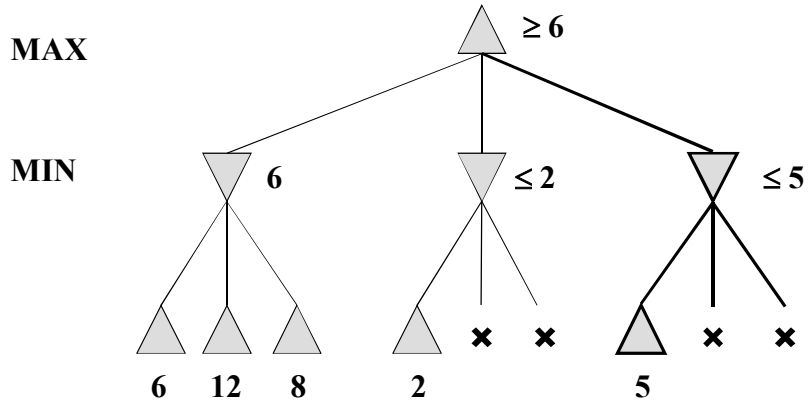
14

α-β pruning: example



15

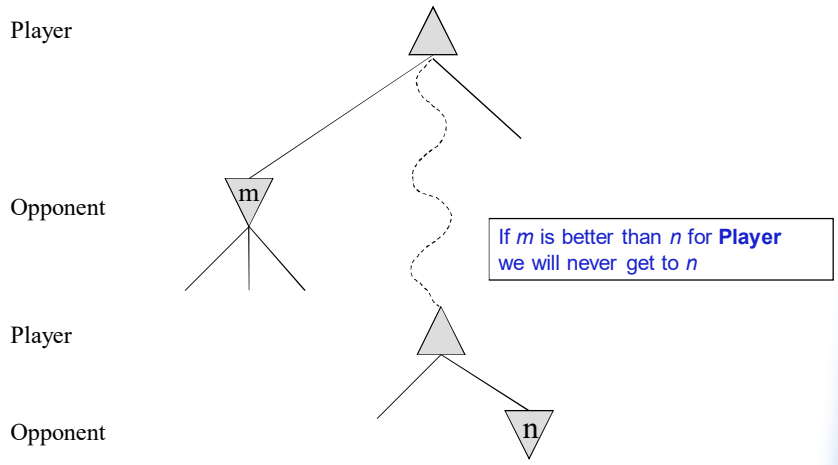
α-β pruning: example



MINIMAX(root) = max(min(6,12,8),min(2,a,b),min(5,b,d))
= max(6,z,y) where z=min(2,a,b)≤ 2 and y=min(5,b,d) ≤ 5
= 6

16

α-β pruning: general principle



17

More on the α-β algorithm

- Because minimax is depth-first, let's consider nodes along a given path in the tree. Then, as we go along this path, we keep track of:
  - α : the value of the best (i.e., highest-value) choice we have found so far at any choice point **along the path** for **MAX**
  - β : the value of the best (i.e., lowest-value) choice we have found so far at any choice point **along the path** for **MIN**

18

The α-β algorithm:

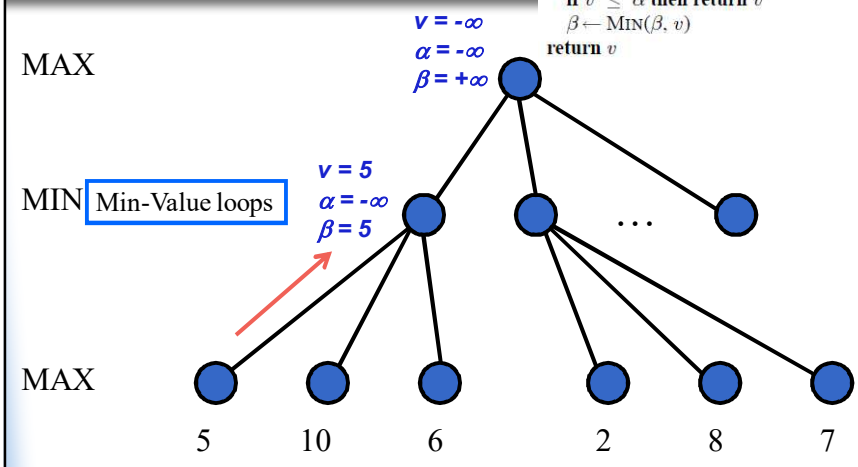
```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

19

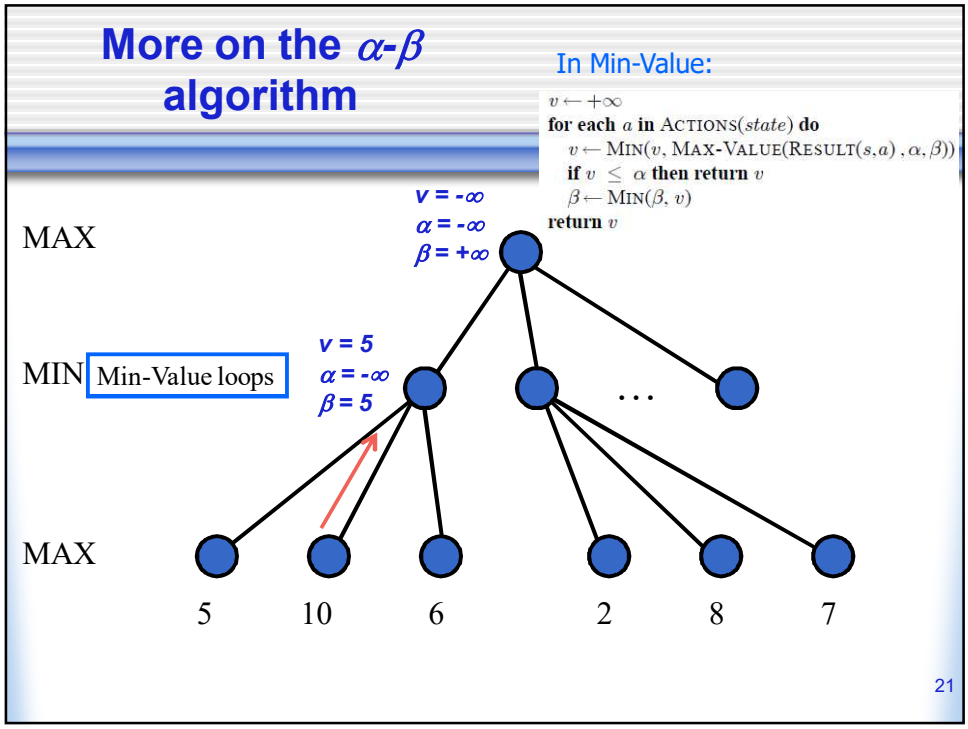
More on the α-β algorithm

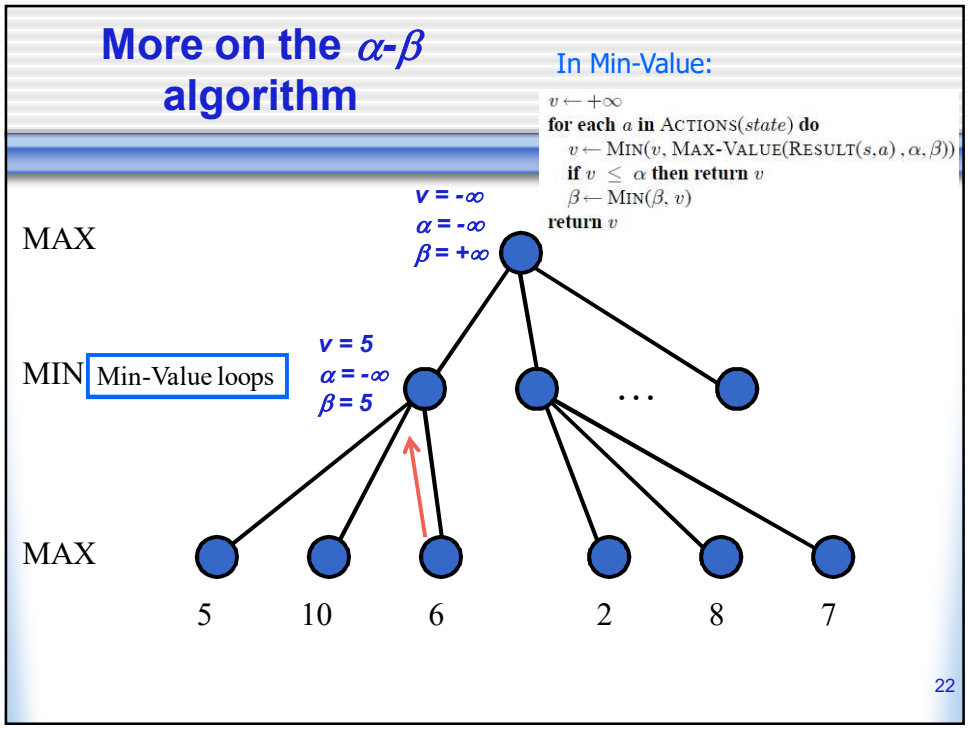
In Min-Value:

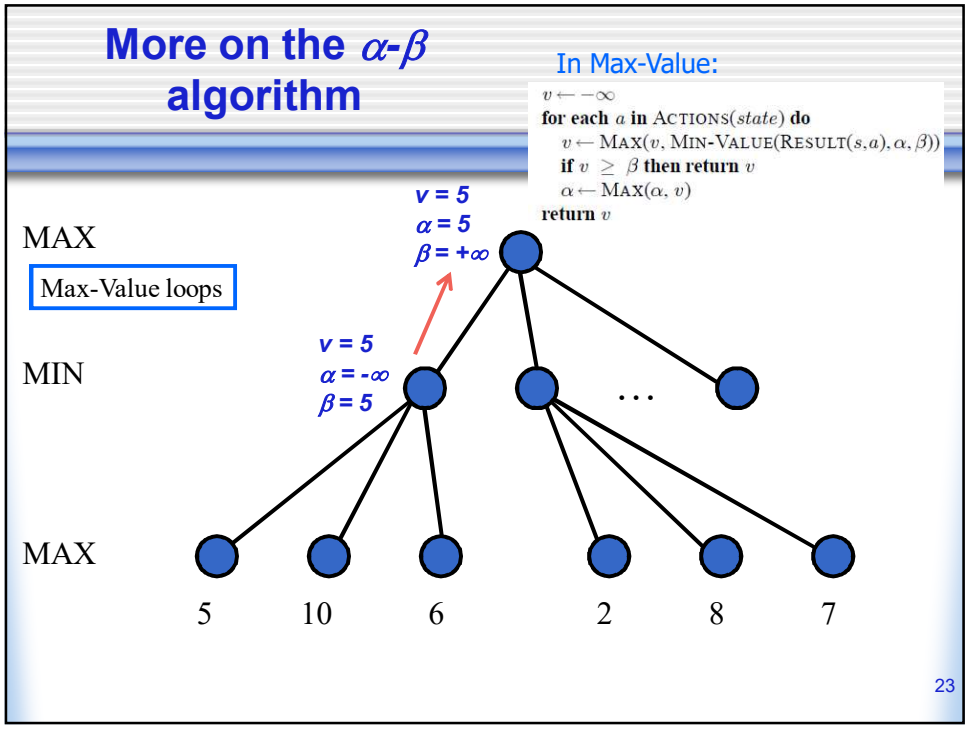
```
v ← +∞
for each a in ACTIONS(state) do
  v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
  if v ≤ α then return v
  β ← MIN(β, v)
return v
```

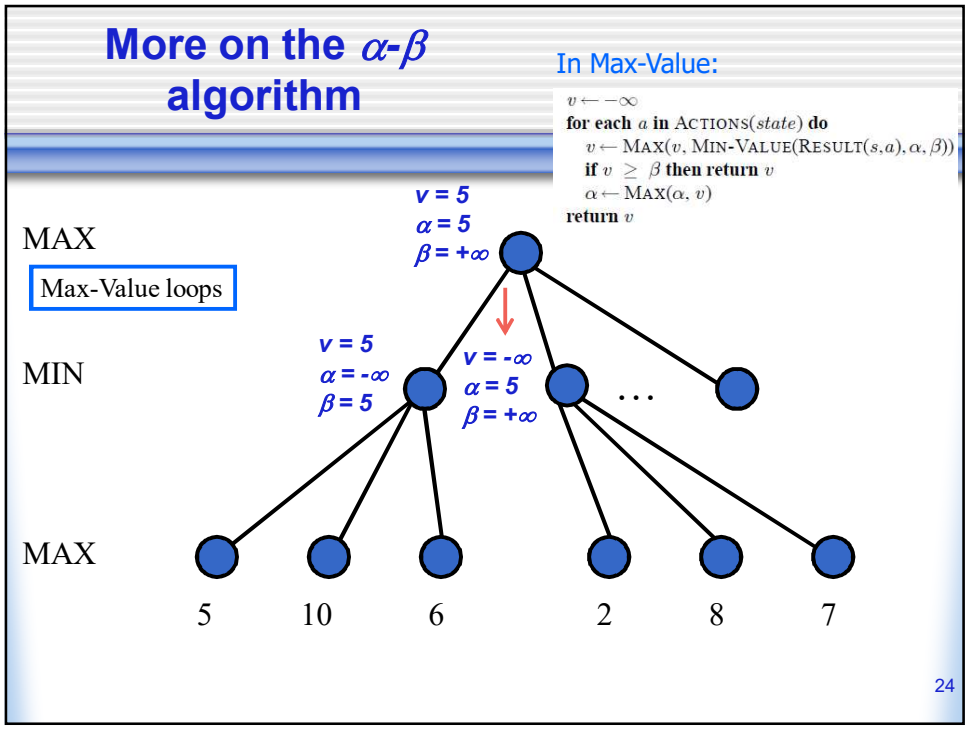


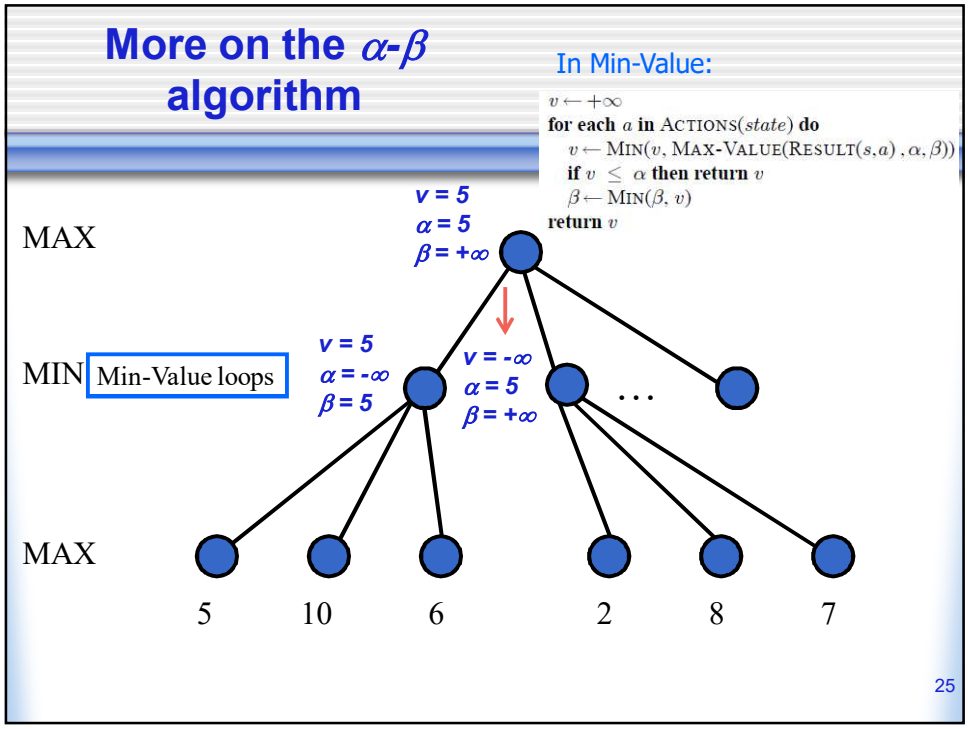
20

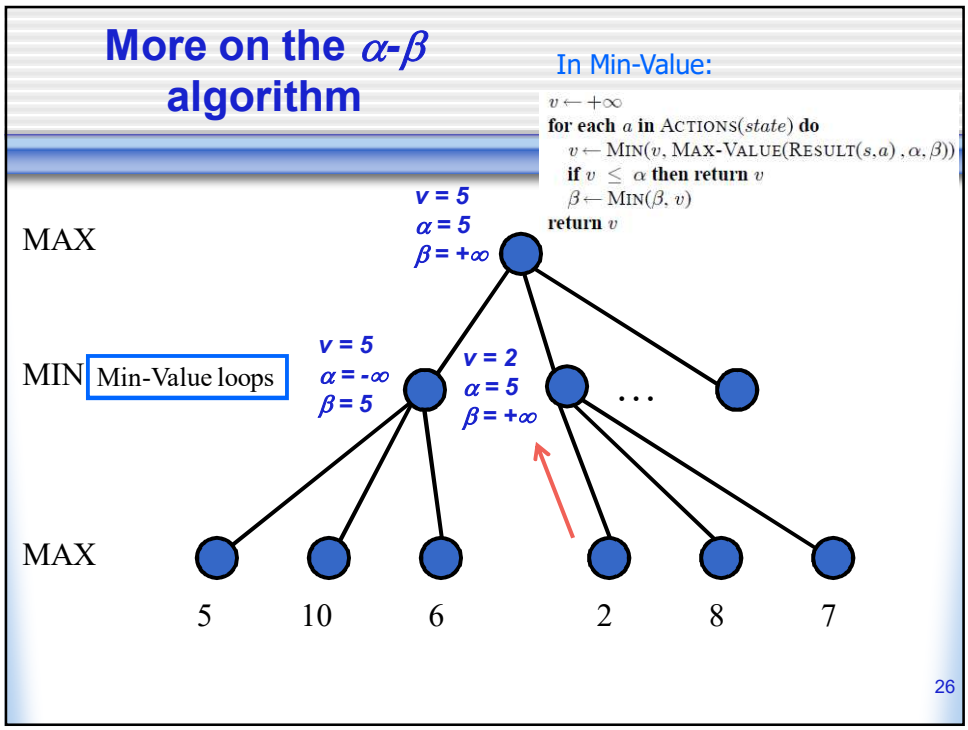

More on the α-β algorithm
In Min-Value:
v ← +∞
for each a in ACTIONS(state) do
v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
if v ≤ α then return v
β ← MIN(β, v)
return v
MAX
MIN
Min-Value loops
MAX
v = -∞
α = -∞
β = +∞
v = 5
α = -∞
β = 5
5
10
6
2
8
7
21

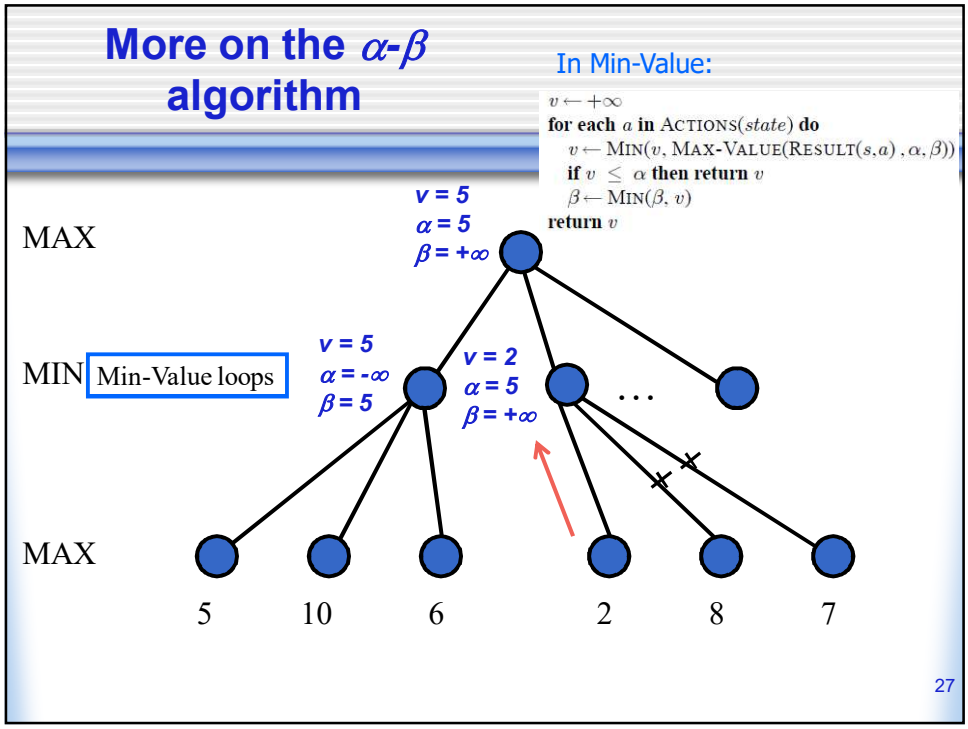

More on the α-β algorithm
In Min-Value:
v ← +∞
for each a in ACTIONS(state) do
v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
if v ≤ α then return v
β ← MIN(β, v)
return v
MAX
MIN
Min-Value loops
MAX
v = -∞
α = -∞
β = +∞
v = 5
α = -∞
β = 5
5
10
6
2
8
7
22

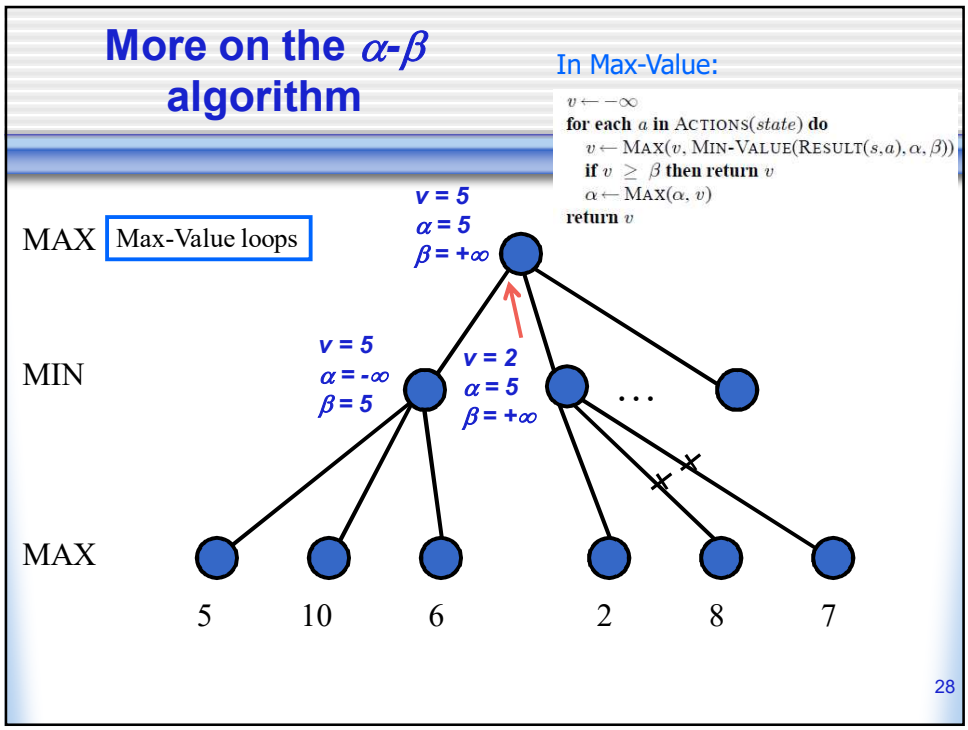

More on the α-β algorithm
In Max-Value:
v ← −∞
for each a in ACTIONS(state) do
v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
if v ≥ β then return v
α ← MAX(α, v)
return v
MAX
Max-Value loops
MIN
MAX
v = 5
α = 5
β = +∞
v = 5
α = -∞
β = 5
5
10
6
2
8
7
23


More on the α-β algorithm
In Max-Value:
v ← −∞
for each a in ACTIONS(state) do
v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
if v ≥ β then return v
α ← MAX(α, v)
return v
MAX
Max-Value loops
MIN
MAX
v = 5
α = 5
β = +∞
v = 5
α = -∞
β = 5
v = -∞
α = 5
β = +∞
5
10
6
2
8
7
24


More on the α-β algorithm
In Min-Value:
v ← +∞
for each a in ACTIONS(state) do
v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
if v ≤ α then return v
β ← MIN(β, v)
return v
MAX
MIN
Min-Value loops
MAX
v = 5
α = 5
β = +∞
v = 5
α = -∞
β = 5
v = -∞
α = 5
β = +∞
5
10
6
2
8
7
25


More on the α-β algorithm
In Min-Value:
v ← +∞
for each a in ACTIONS(state) do
v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
if v ≤ α then return v
β ← MIN(β, v)
return v
MAX
MIN
Min-Value loops
MAX
v = 5
α = 5
β = +∞
v = 5
α = -∞
β = 5
v = 2
α = 5
β = +∞
5
10
6
2
8
7
26


More on the α-β algorithm
In Min-Value:
v ← +∞
for each a in ACTIONS(state) do
v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
if v ≤ α then return v
β ← MIN(β, v)
return v
MAX
MIN
Min-Value loops
MAX
v = 5
α = 5
β = +∞
v = 5
α = -∞
β = 5
v = 2
α = 5
β = +∞
5
10
6
2
8
7
27


More on the α-β algorithm
In Max-Value:
v ← −∞
for each a in ACTIONS(state) do
v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
if v ≥ β then return v
α ← MAX(α, v)
return v
MAX
Max-Value loops
MIN
MAX
v = 5
α = 5
β = +∞
v = 5
α = -∞
β = 5
v = 2
α = 5
β = +∞
5
10
6
2
8
7
28

Properties of α-β

- Pruning does not affect the final result!!!
- Good move ordering improves effectiveness of pruning
- With *perfect ordering,* time complexity = $O(b^{m/2})$
  - doubles depth of search
  - need a heuristic how to order
  - can easily reach depth 8 => good chess
- A simple example of the value of reasoning about which computations are relevant (a form of *metareasoning*)

29

2. Move evaluation without complete search

- The minimax algorithm generates the entire game search space, whereas the alpha-beta algorithm allows us to prune large parts of it.
- Complete search is often too complex and impractical. alpha-beta is still DFS.
- **Evaluation function:** evaluates value of state using heuristics and cuts off search
- **New MINIMAX:**
  - CUTOFF-TEST: cutoff test to replace the termination condition (e.g., deadline, depth-limit, etc.)
  - EVAL: evaluation function to replace utility function (e.g., number of chess pieces taken)

30

Evaluation function

- The evaluation function should order the *terminal* states in the same way as the true utility function (a<b<c…).
- The computation must not take to long! Significant compared to minimax?
- For nonterminal states, the evaluation function should be strongly correlated with the actual chances of winning.

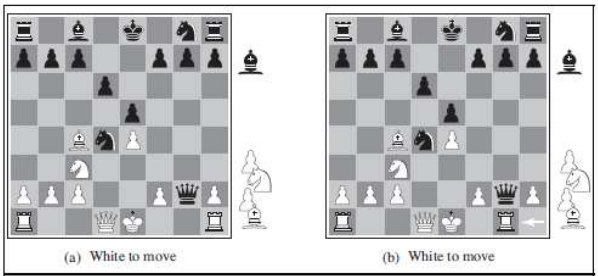
31

Evaluation functions

- Most calculate features – e.g., number of pawns
- From that we can form categories, equivalence classes.
- Any category represent states that win, lose or result in draws.
- If we know 72% lead to win (+1), 20% to loss (0), 8% drawn (1/2).
  Expected value:
- (0,72* +1) + (0,20* 0) + (0,08 * 1/2)= 0,76

32

Evaluation functions



(a) White to move (b) White to move

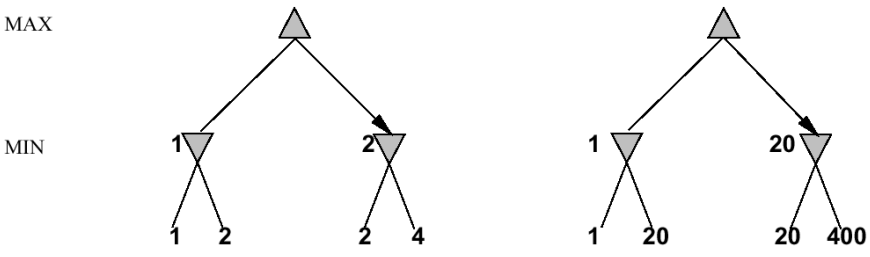
Figure 5.8 FILES: figures/chess-evaluation3.eps (Tue Nov 3 16:22:33 2009). Two chess positions that differ only in the position of the rook at lower right. In (a), Black has an advantage of a knight and two pawns, which should be enough to win the game. In (b), White will capture the queen, giving it an advantage that should be strong enough to win.

- **Weighted linear evaluation function:** to combine *n* heuristics
  $f = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$

E.g, w's could be the values of pieces (1 for pawn, 3 for bishop etc.)
f's could be the number of type of pieces on the board

33

Note: exact values do not matter



Behaviour is preserved under any *monotonic* transformation of EVAL

Only the order matters:
payoff in deterministic games acts as an *ordinal utility* function

34

With cutoff and eval

function MAX-VALUE($state, \alpha, \beta$) **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 CUTOFF-TEST(*state*, *depth*) **then return** EVAL(*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

35

Minimax with cutoff: viable algorithm?

MINIMAXCUTOFF is identical to MINIMAXVALUE except
1. TERMINAL? is replaced by CUTOFF?
2. UTILITY is replaced by EVAL

Does it work in practice?

$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

4-ply lookahead is a hopeless chess player!

4-ply ≈ human novice
8-ply ≈ typical PC, human master
12-ply ≈ Deep Blue, Kasparov

Assume we have 100 seconds, evaluate 10^4 nodes/s; can evaluate 10^6 nodes/move

36

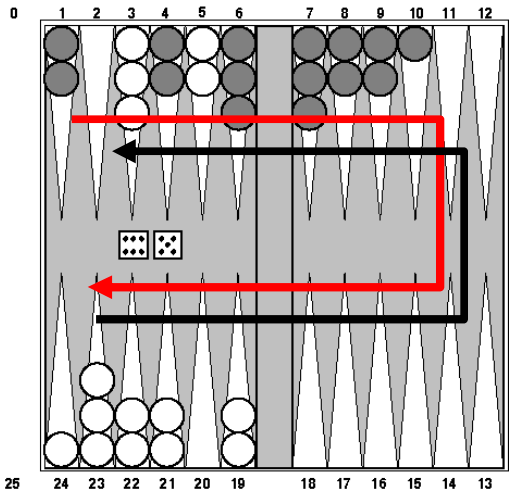
Other Cutoff methods

- Quiescent search
  apply eval only to positions that are quiescent, have no big change of value in the near future.
- Forward pruning
  considers not all moves in a concrete position.
  Beam search is one approach to forward pruning.
- ProbCut
  probabilistic alpha-beta with statistical prior knowledge

37

Games of chance

- *Backgammon is a two-player game with **uncertainty**.*
- *Players roll dices to determine what moves to make.*
- *White/red arrow has just rolled 5 and 6 and has four legal moves:*
  - *5-10, 5-11*
  - *5-11, 19-24*
  - *5-10, 10-16*
  - *5-11, 11-16*
- *Such games are good for exploring decision making in adversarial problems involving skill and luck.*



38

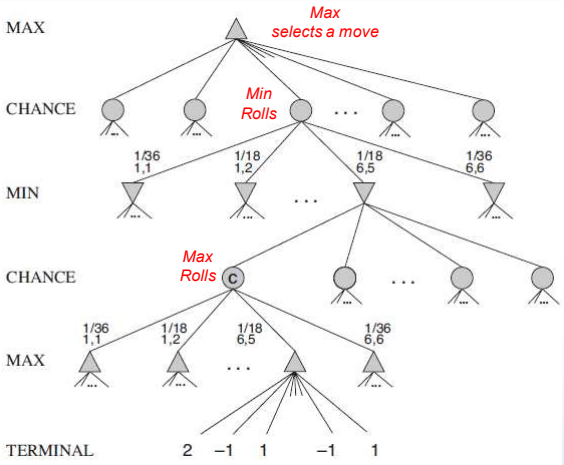
Game Trees with Chance Nodes

- Use minimax to compute values for MAX and MIN nodes
- Use **expected values** for chance nodes
- For chance nodes over a max node, as in C:

expectimax(C) = $\text{Sum}_i(P(d_i) * \text{maxvalue}(i))$

- For chance nodes over a min node:

expectimin(N) = $\text{Sum}_i(P(d_i) * \text{minvalue}(i))$



39

Algorithm for nondeterministic games

EXPECTIMINIMAX gives perfect play.

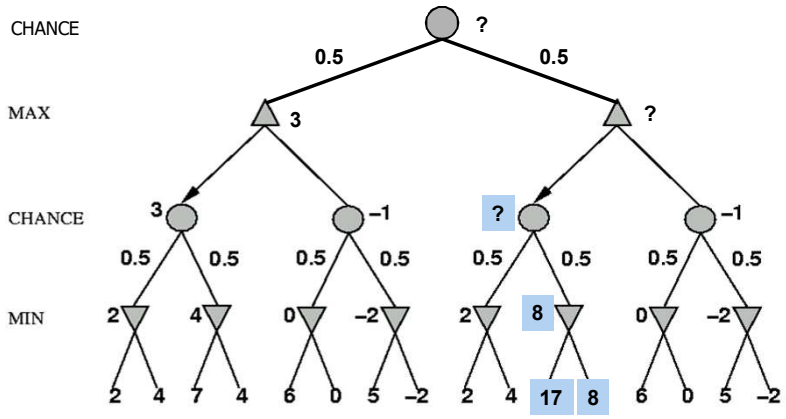
$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

A version of α-β is possible but only if leaf values are bounded. WHY??

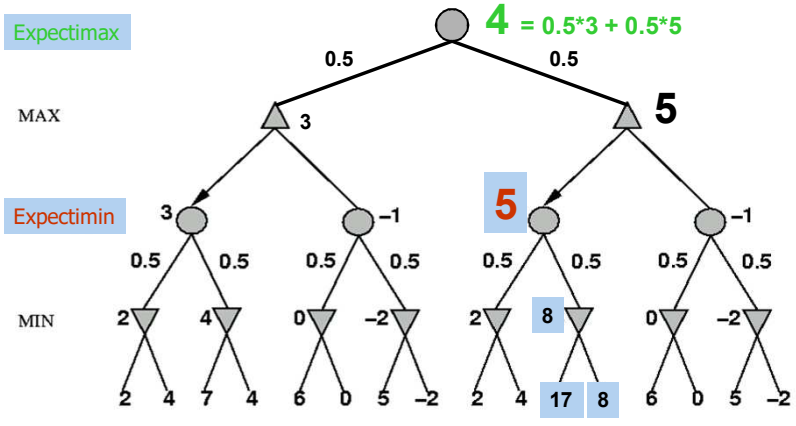
40

Nondeterministic games: the element of chance

expectimax and **expectimin**, expected values over all possible outcomes

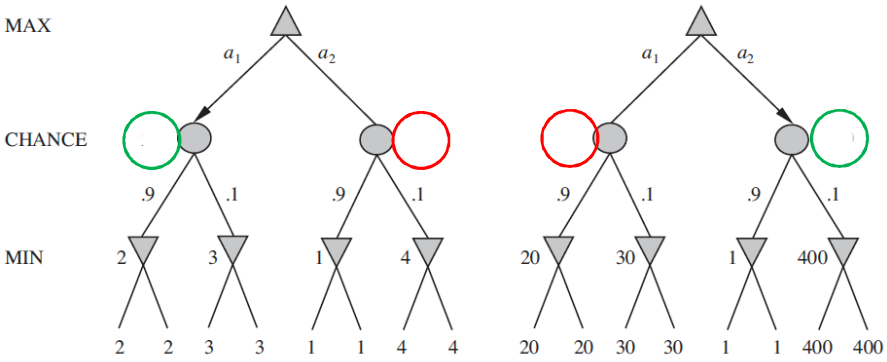


Nondeterministic games: the element of chance



Evaluation functions

Order-preserving transformation do not necessarily behave the same!



43

Games of imperfect information

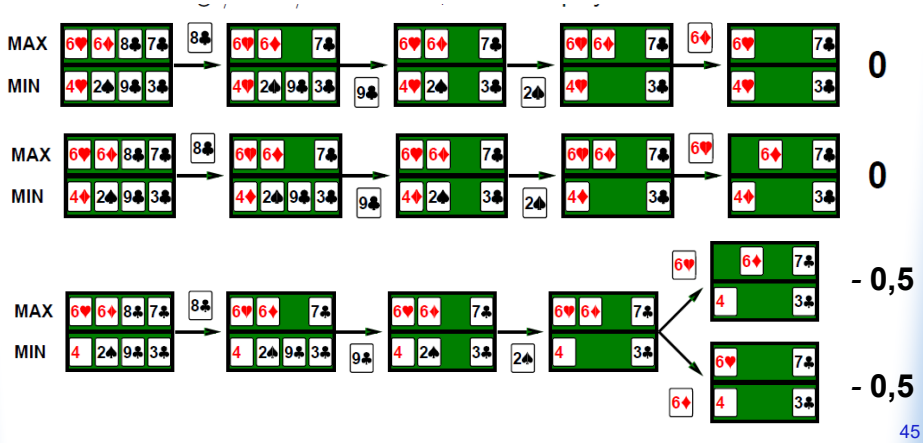
- E.g., card games, where opponent's initial cards are unknown
- Typically we can calculate a probability for each possible deal
- Seems just like having one big dice roll at the beginning of the game
- Idea: compute the minimax value of each action in each deal, then choose the action with highest expected value over all deals
- Special case: if an action is optimal for all deals, it's optimal.

- GIB, current best bridge program, approximates this idea by
  - generating 100 deals consistent with bidding information
  - picking the action that wins most tricks on average

44

Example

- Four card bridge, MAX to play first



45

Proper analysis

- Intuition that the value of an action is the average of its values in all actual states is *WRONG*
- With partial observability, value of an action depends on the *information state* or *belief state* the agent is in
- Can generate and search a tree of information states
- Leads to rational behaviors such as
  - Acting to obtain information
  - Signalling to one's partner
  - Acting randomly to minimize information disclosure

46

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
  - uncertainty constrains the assignment of values to states
  - optimal decisions depend on information state, not real state

47