CS444

Nathan Sprague

September 10, 2012

Minimax!

$$\text{MINIMAX}(s, d) =$$

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

Minimax!

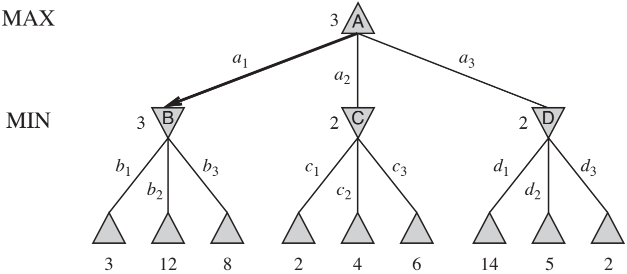
```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

```
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
```

Figure 5.3 An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state. The notation $\operatorname{argmax}_{a \in S} f(a)$ computes the element a of set S that has the maximum value of $f(a)$.

Minimax!



Sequential Rochambeau

- One game consists of three moves (plys)*.
- Player one (MAX) selects Paper, Rock or Scissors.
- Player two (MIN) selects one of the other options.
  - If MAX picked Paper, MIN may pick Rock or Scissors.
- MAX is then required to select the only remaining choice.
- Scoring is based on the last two picks:
  - Rock/Paper: 1/-1
  - Paper/Scissors: 2/-2
  - Scissors/Rock: 3/-3

∗ Really only two, since the third move is forced.

Your Job...

- Draw the game gree.
- Calculate the minimax value of each state.
- Describe the optimal policy of both players.

Alpha-Beta Pruning

```
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
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the

Alpha-Beta Pruning

```
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
```

From this state MAX can guarantee an outcome better than beta! MIN would never let us reach this state, so stop.

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the

Alpha-Beta Pruning

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  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*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

If this state is a new best-case for MAX, change alpha, so we can warn MIN-VALUE not to bother with anything worse.

---

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

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the

Alpha-Beta Pruning

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  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*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

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

From this state, MIN can guarantee an outcome below alpha! MAX would never this happen. Stop.

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the

Alpha-Beta Pruning

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  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*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

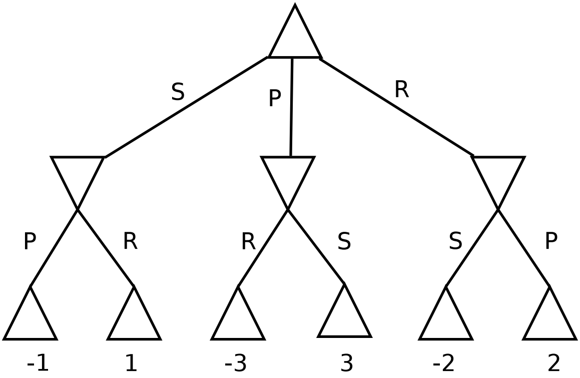
---

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

If this state is a new best-case for MIN, change beta, so we can warn MAX-VALUE not to bother with anything worse.

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the

Pruning Rochambeau



Which (if any) search tree nodes will not be visited if we use alpha-beta pruning?

Limited Time...

$$\text{H-MINIMAX}(s, d) =$$

$$= \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s, d) \\ \max_a \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

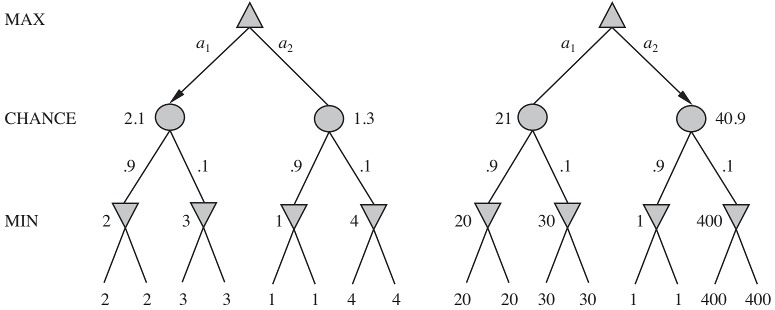
Games of Chance

- Can we apply minimax if there is an element of chance?
- Sort of...

$$\text{EXPECTIMINIMAX}(s, d) =$$

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

Games of Chance



Status of Games

Three categories:

- "Solved"
  - Sequential Rochambeau
  - tic-tac-toe
  - Checkers
- Best computer player is better than the best human player
  - Chess
  - Othello
- Best human players are better than the best computer players
  - Go (UCT- upper confidence bounds on trees)
  - Poker (?)