Two Player Games

A4B33ZUI, LS 2017

Branislav Bošanský, Karel Horák, Ondřej Vaněk
{name.surname}@agents.fel.cvut.cz
Artificial Intelligence Center, Czech Technical University

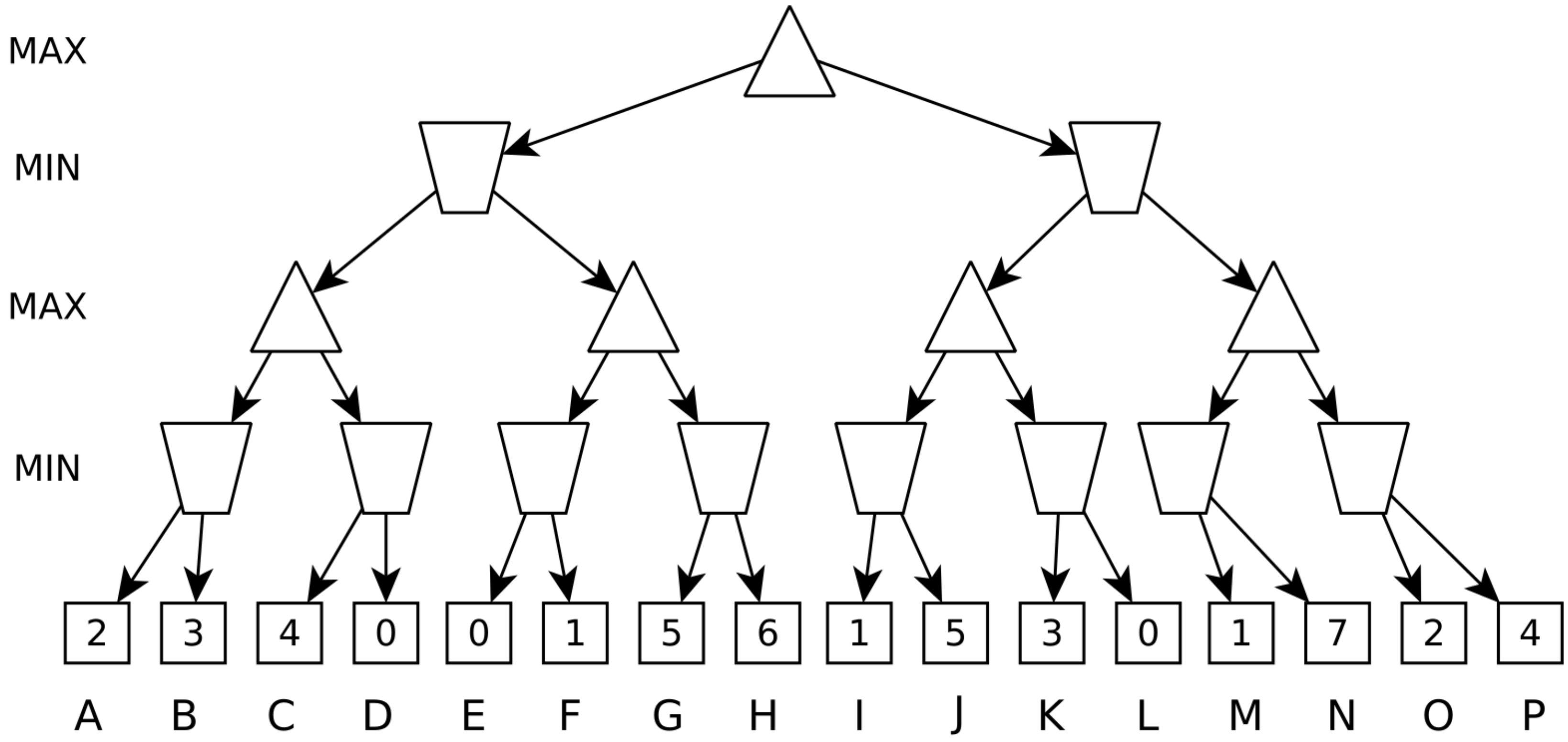
Minimax

- **function** minimax(node, depth, Player)
- **if** (depth = 0 or node is a terminal node) **return** evaluation value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- v := max(v, minimax(child, depth-1, switch(Player)))
-
- **return** v
- **else**
- **for each** child of node
- v := min(v, minimax(child, depth-1, switch(Player)))
-
- **return** v

Alpha-Beta Pruning

- **function** alphabeta(node, depth, α, β, Player)
- **if** (depth = 0 or node is a terminal node) **return** evaluation value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- v := max(v, alphabeta(child, depth-1, α, β, switch(Player)))
- α := max(α,v); **if** (β≤α) break
- **return** v
- **else**
- **for each** child of node
- v := min(v, alphabeta(child, depth-1, α, β, switch(Player)))
- β := min(β,v); **if** (β≤α) break
- **return** v

Game



Negamax

- **function** negamax(node, depth, α, β, Player)
- **if** (depth = 0 or node is a terminal node) **return** evaluation value of node
- **if** (Player = MaxPlayer)
- **for each** child of node
- v := max(v, -negamax(child, depth-1, -β, -α, switch(Player)))
- α := max(α,v); **if** (β≤α) break
- **return** v
- **else**
- **for each** child of node
- v := min(v, alphabeta(child, depth-1, α, β, switch(Player)))
- β := min(β,v); **if** (β≤α) break
- **return** v

NegaScout – Main Idea

- enhancement of the alpha-beta algorithm
- assumes some heuristic that determines move ordering
  - the algorithm assumes that the first action is the best one
  - after evaluating the first action, the algorithm checks whether the remaining actions are worse
  - the "test" is performed via null-window search
    - $[\alpha, \alpha+1]$
    - the algorithm needs to re-search, if the test fails (i.e., there might be a better outcome for the player when following the tested action)

NegaScout

function negascout(node, depth, α, β, Player)

- **if** ((depth = 0) or (node is a terminal node)) **return** eval(node)
- b := β
- **for each** child of node
- v := max(v,-negascout(child, depth-1, -b, -α, switch(Player))))
- **if** ((α < v) and (child is not the first child))
- v := max(v,-negascout(child, depth-1, -β, -α, switch(Player))))
- α := max(α, v)
- **if** (β≤α) break
- b := α + 1
- **return** v

NegaScout

function negascout(node, depth, α, β, Player)

```
•   if ((depth = 0) or (node is a terminal node)) return eval(node)
•   b := β
•   for each child of node
•       v := max(v,-negascout(child, depth-1, -b, -α, switch(Player))))
•       if (( α < v < β) and (child is not the first child))
•           v := max(v,-negascout(child, depth-1, -β, -v, switch(Player))))
•       α := max(α, v)
•      if (β≤α)  break
•      b := α + 1
•   return v
```

Alpha Beta and Negascout in Practice

- Extracting selected moves
- Cache for previous results (transposition tables)
- Iterative deepening (using previous results in game playing)
- Implementation of game states (bit operations, modifications have to be as quick as possible)

Alpha Beta and Negascout in Practice

- TEST on alpha beta and negascout
  - **3.4. and 4.4. 2017 on seminars**