

BÚSQUEDA ADVERSARIA II

ALAN REYES-FIGUEROA
INTELIGENCIA ARTIFICIAL

(AULA 26) 21.ABRIL.2025

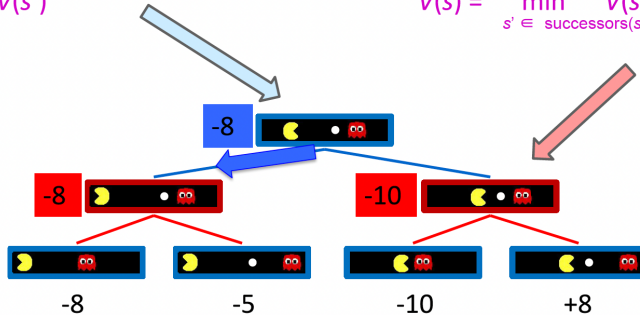
Minimax

MAX nodes: under Agent's control

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

MIN nodes: under Opponent's control

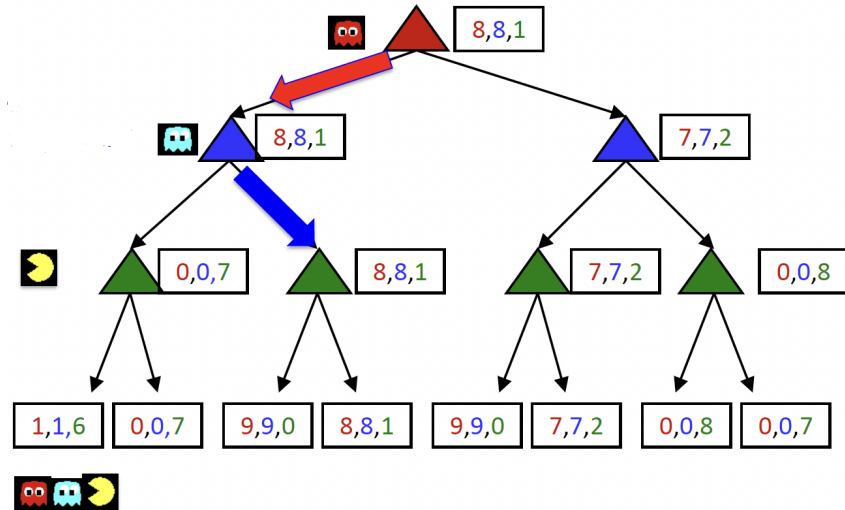
$$V(s) = \min_{s' \in \text{successors}(s)} V(s')$$



Terminal States:

$V(s) = \text{known}$

Minimax



Minimax

Implementación de Minimax

function **decision(s)** returns an action

return the action **a** in **Actions(s)** with the highest
minimax_value(Result(s,a))



function **minimax_value(s)** returns a value

if **Terminal-Test(s)** then return **Utility(s)**

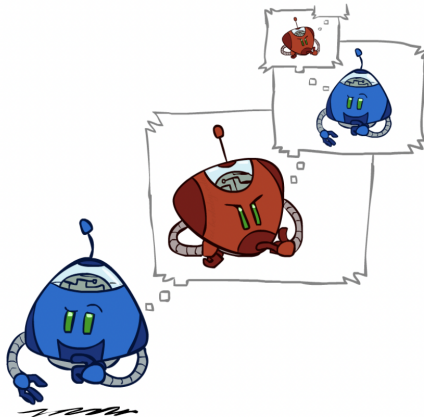
if **Player(s) = MAX** then return **max**_{a in Actions(s)} **minimax_value(Result(s,a))**

if **Player(s) = MIN** then return **min**_{a in Actions(s)} **minimax_value(Result(s,a))**

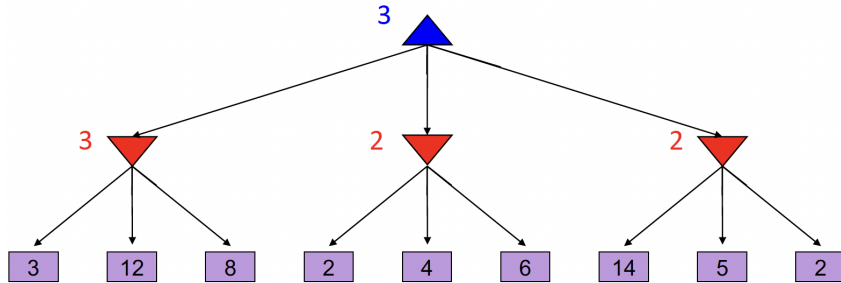
Minimax

Eficiencia del algoritmo Minimax

- How efficient is minimax?
 - Just like (exhaustive) DFS
 - Time: $O(b^m)$
 - Space: $O(bm)$
- Example: For chess, $b \approx 35$, $m \approx 100$
 - Exact solution is completely infeasible
 - Humans can't do this either, so how do we play chess?

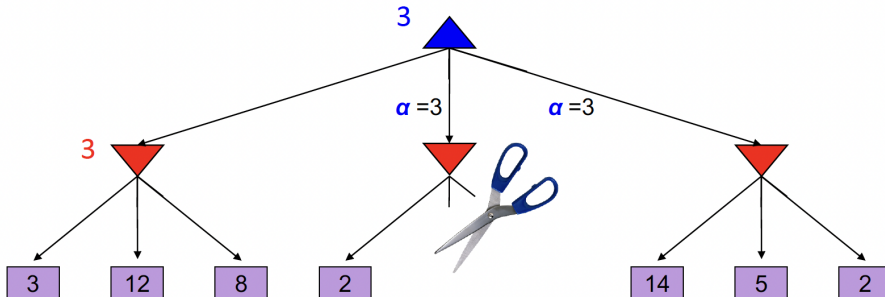


α - β Pruning



α - β Pruning

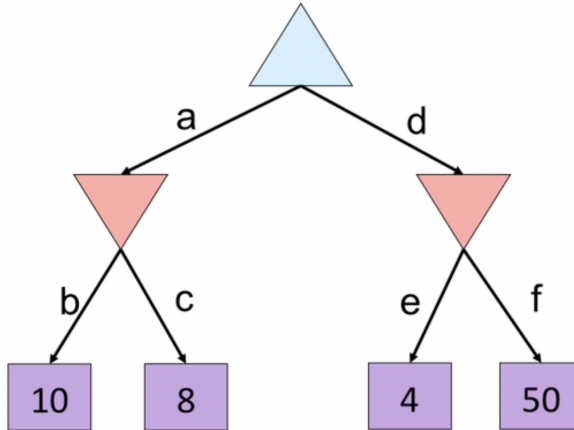
α = best option so far from any MAX node on this path



- ***The order of generation matters***: more pruning is possible if good moves come first

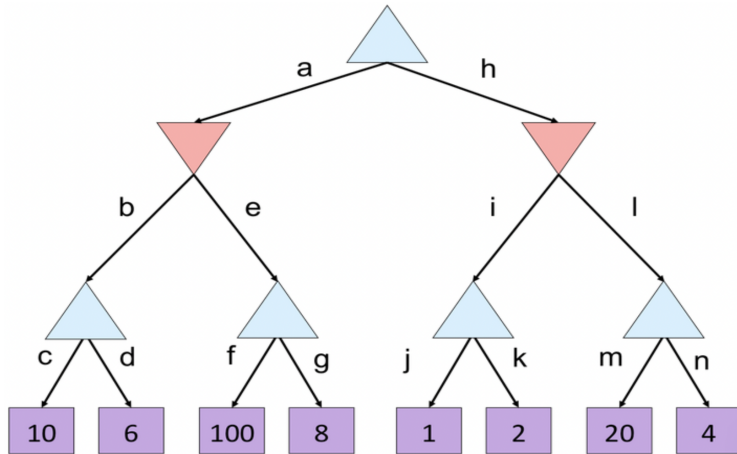
α - β Pruning

Ejercicio 1: Determinar cuáles nodos se descartan en la poda α - β .



α - β Pruning

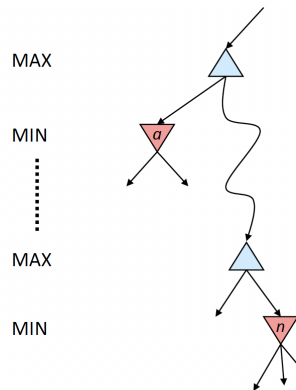
Ejercicio 2: Determinar cuáles nodos se descartan en la poda α - β .



α - β Pruning

Implementación de α - β Pruning

- General case (pruning children of MIN node)
 - We're computing the MIN-VALUE at some node n
 - We're looping over n 's children
 - n 's estimate of the childrens' min is dropping
 - Who cares about n 's value? MAX
 - Let α be the best value that MAX can get so far at any choice point along the current path from the root
 - If n becomes worse than α , MAX will avoid it, so we can prune n 's other children (it's already bad enough that it won't be played)
- Pruning children of MAX node is symmetric
 - Let β be the best value that MIN can get so far at any choice point along the current path from the root



α - β Pruning

Implementación de α - β Pruning

α : MAX's best option on path to root
 β : MIN's best option on path to root

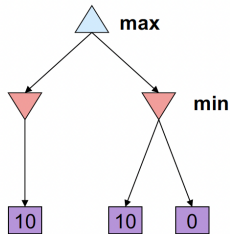
```
def max-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = -\infty$   
    for each successor of state:  
         $v = \max(v, \text{min-value}(\text{successor}, \alpha, \beta))$   
        if  $v \geq \beta$   
            return  $v$   
         $\alpha = \max(\alpha, v)$   
    return  $v$ 
```

```
def min-value(state,  $\alpha$ ,  $\beta$ ):  
    initialize  $v = +\infty$   
    for each successor of state:  
         $v = \min(v, \text{max-value}(\text{successor}, \alpha, \beta))$   
        if  $v \leq \alpha$   
            return  $v$   
         $\beta = \min(\beta, v)$   
    return  $v$ 
```

α - β Pruning

Propiedades de α - β Pruning

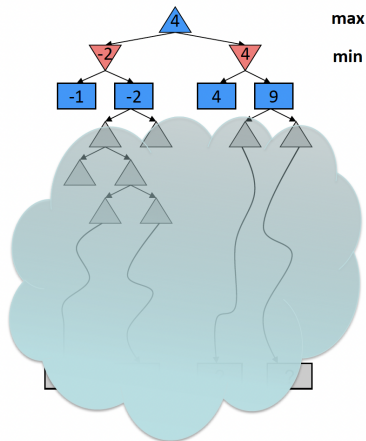
- Theorem: This pruning has **no effect** on minimax value computed for the root!
- Good child ordering improves effectiveness of pruning
 - Iterative deepening helps with this
- With “perfect ordering”:
 - Time complexity drops to $O(b^{m/2})$
 - Doubles solvable depth!
- This is a simple example of **metareasoning** (reasoning about reasoning)
- For chess: only 35^{50} instead of 35^{100} !! Yaaay!!!!!



Límite de Recursos

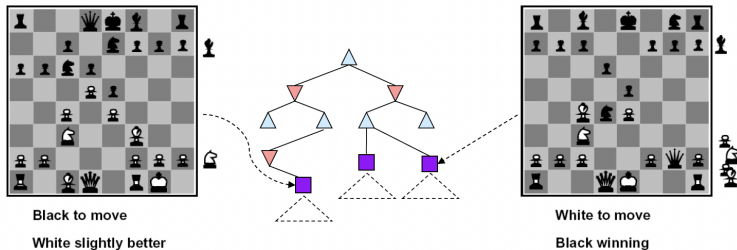
En la práctica no se hace una evaluación exhaustiva.

- Problem: In realistic games, cannot search to leaves!
- Solution (Shannon, 1950): Bounded lookahead
 - Search only to a preset **depth limit** or **horizon**
 - Use an **evaluation function** for non-terminal positions
- Guarantee of optimal play is gone
- Example:
 - Suppose we can explore 1M nodes per move
 - Chess with alpha-beta, $35^{(8/2)} \approx 1M$; depth 8 is quite good



Funciones de Utilidad

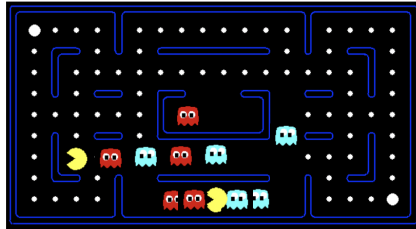
- Evaluation functions score non-terminals in depth-limited search



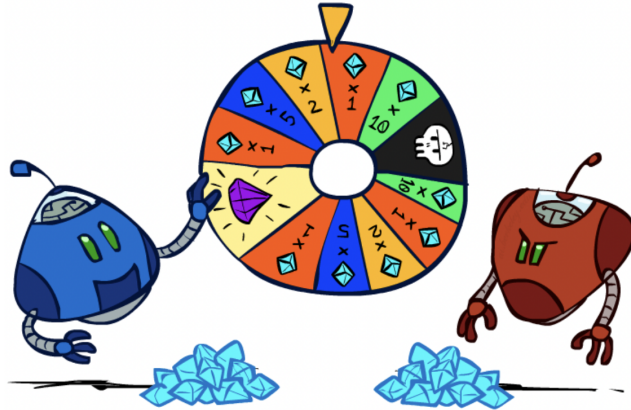
- Typically weighted linear sum of features:
 - $EVAL(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$
 - E.g., $w_1 = 9$, $f_1(s) = (\text{num white queens} - \text{num black queens})$, etc.
- Or a more complex nonlinear function (e.g., NN) trained by self-play RL
- Terminate search only in **quiescent** positions, i.e., no major changes expected in feature values

Funciones de Utilidad

Ejercicio: ¿Cómo diseñar funciones de utilidad para Pac-Man?

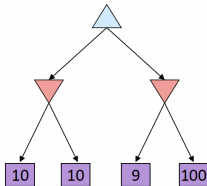
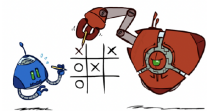


Juegos bajo incerteza

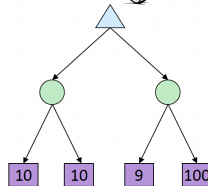


Expectimax y Expectiminimax

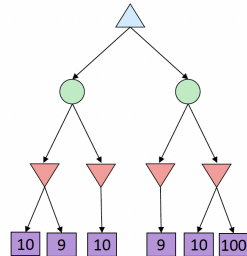
Chance outcomes in trees



Tictactoe, chess
Minimax



Tetris, investing
Expectimax



Backgammon, Monopoly
Expectiminimax

Expectimax

Implementación de Expectiminimax

function **decision(s)** returns an action

return the action **a** in **Actions(s)** with the highest
value(Result(s,a))



function **value(s)** returns a value

if **Terminal-Test(s)** then return **Utility(s)**

if **Player(s) = MAX** then return **max**_{a in Actions(s)} **value(Result(s,a))**

if **Player(s) = MIN** then return **min**_{a in Actions(s)} **value(Result(s,a))**

if **Player(s) = CHANCE** then return **sum**_{a in Actions(s)} **Pr(a) * value(Result(s,a))**

Expectiminimax

Ejemplo: Backgammon

- Dice rolls increase *b*: 21 possible rolls with 2 dice
 - Backgammon ≈ 20 legal moves
 - 4 plies = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$
- As depth increases, probability of reaching a given search node shrinks
 - Usefulness of search is diminished
 - Pruning is trickier...
- Historic AI: TDGammon (1997) uses depth-2 search + very good evaluation function + reinforcement learning: world-champion level play



Montecarlo Tree Search



Montecarlo Tree Search

