

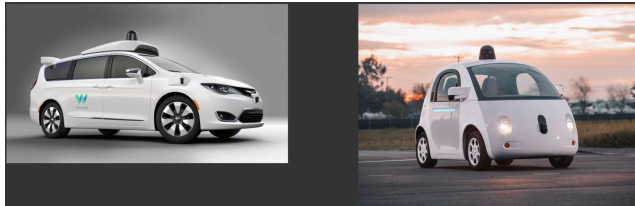
BÚSQUEDA ADVERSARIA

ALAN REYES-FIGUEROA
INTELIGENCIA ARTIFICIAL

(AULA 17) 15.MARZO.2024

Juegos Adversarios

Agentes interactuando con agentes



Agentes interactuando con personas



Juegos Adversarios

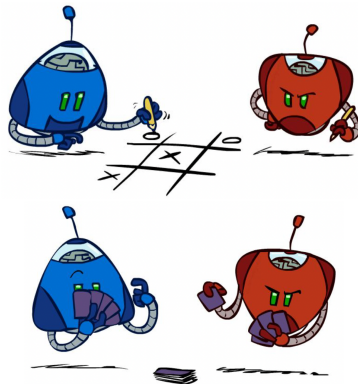
Modelamos estas situaciones a través de juegos.

Juego = ambiente en donde el número de agentes > 1 .

Existen varios tipos de juegos:

- ¿Determinista o estocástico?
- ¿Información perfecta (completamente observable) o imperfecta?
- ¿Dos, tres, o más jugadores? ¿Equipos o individuos?
- ¿Por turnos o jugadas simultáneas?
- ¿De suma cero?

Queremos algoritmos para calcular un plan de contingencia (**estrategia** o **política**) que recomienda un movimiento para cada posible eventualidad.

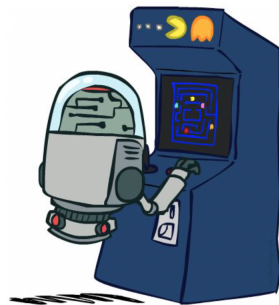


Juegos Adversarios

Los juegos estándar que se pueden modelar fácilmente son: deterministas, completamente observable, de 2 jugadores, por turnos, de suma cero.

Formulación de un juego:

- Conjunto de estados: S
- Estado inicial: $s_0 \in S$
- Jugadores: $P = \{p_i\}_{i=1}^N$, e i indica de quien es el turno actual
- Acciones: A_i (depende de cada jugador)
- Función de transición: $S \times A_i \rightarrow S$ (depende de p_i)
- Estados terminales: Test terminal $S \rightarrow \{T, F\}$
- Valores terminales: $U : S \times P \rightarrow \mathbb{R}$, es la **función de utilidad**. $U(s, p_i) = \text{Utility}_i(s)$, para el jugador p_i .

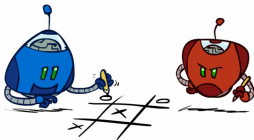


Para cada jugador, buscamos una **solución** o **política** $P_i : S \rightarrow A_i$.

Juegos Adversarios

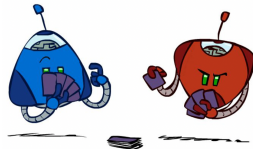
Juegos de suma cero:

- Los agentes tienen utilidades opuestas (valores de los resultados).
- Uno maximiza y el otro minimiza.
- Juego adversario, competencia pura.



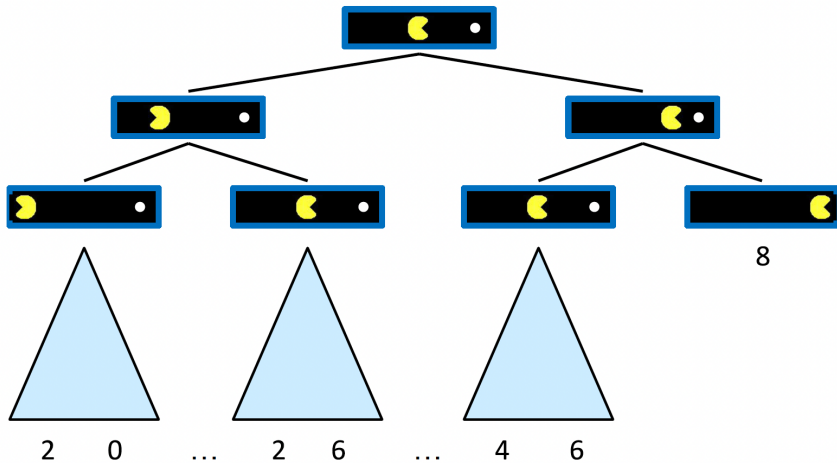
Juegos generales:

- Los agentes tienen utilidades independientes (valores de los resultados).
- Puede haber cooperación, indiferencia, competencia, u otras formas de juego.
- Más adelante hablaremos sobre juegos de suma distinta de cero.

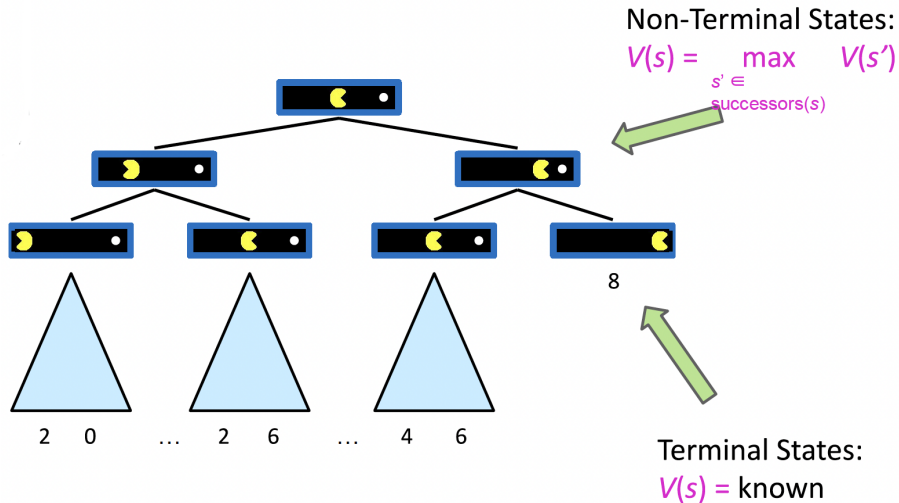


Ejemplos de Juegos

Juegos Adversarios



Juegos Adversarios



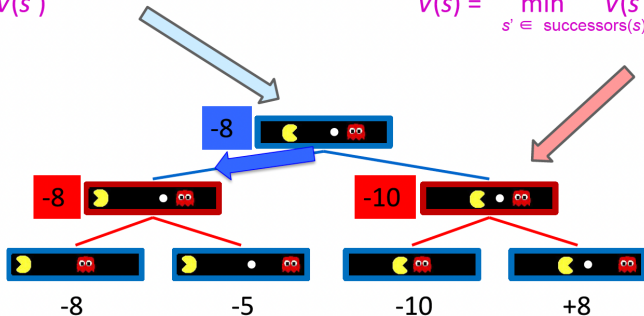
Juegos Adversarios

MAX nodes: under Agent's control

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

MIN nodes: under Opponent's control

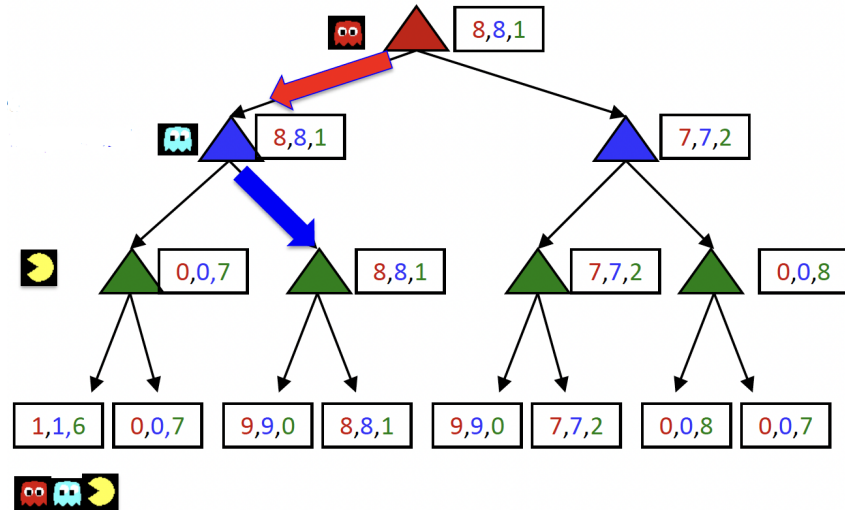
$$V(s) = \min_{s' \in \text{successors}(s)} V(s')$$



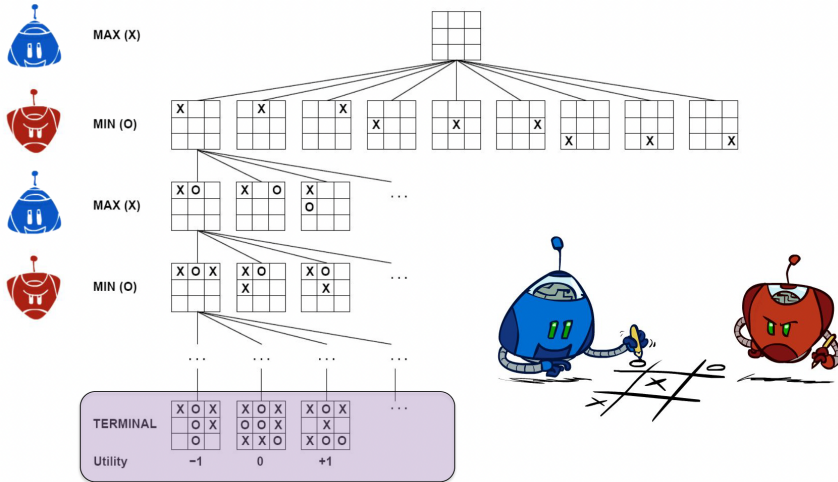
Terminal States:

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

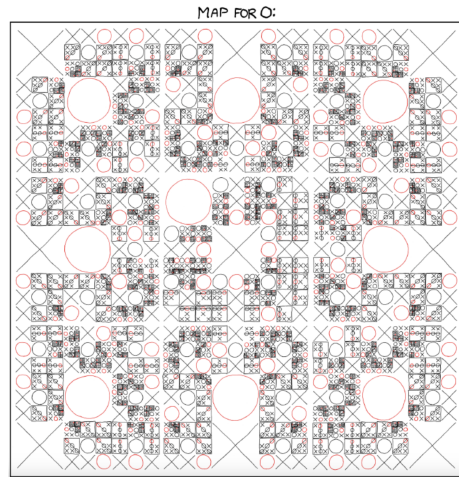
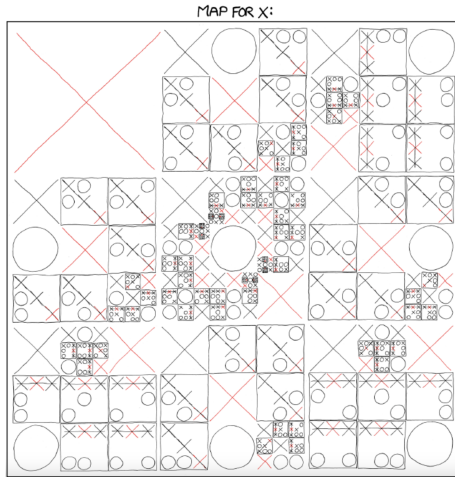
Caso General



Caso General



Caso General



MINIMAX IMPLEMENTATION

MAX Agent:

Function *max-value*(state)

$v \leftarrow -\infty$;

for each successor of state **do**

$v \leftarrow \max(v, \min - \text{value}(\text{successor}))$;

return v ;

end



MIN Agent:

Function *min-value*(state)

$v \leftarrow +\infty$;

for each successor of state **do**

$v \leftarrow \min(v, \max - \text{value}(\text{successor}))$;

return v ;

end

MINIMAX IMPLEMENTATION (DISPATCH)

Dispatch:

```
Function value(state)
  if state is a terminal state then
    | return the state's utility
  end
  if next agent is MAX then
    | return max-value(state)
  end
  if next agent is MIN then
    | return min-value(state)
  end
```

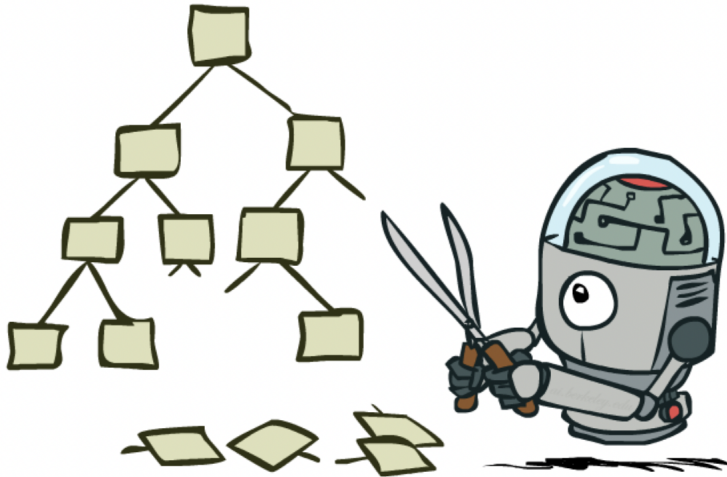
MAX Agent:

```
Function max-value(state)
   $v \leftarrow -\infty$  ;
  for each successor of state do
    |  $v \leftarrow \max(v, \text{min} - \text{value}(\text{successor}))$  ;
    | return v;
  end
```

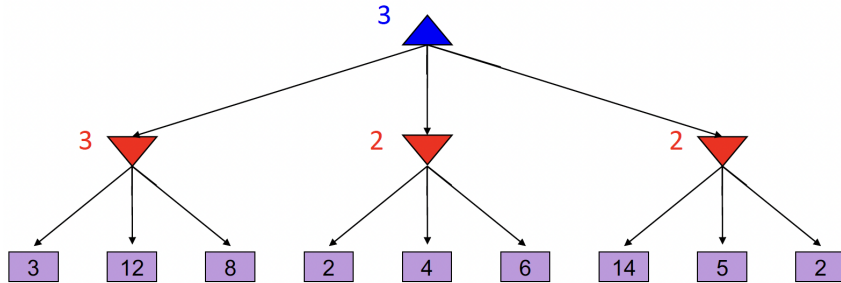
MIN Agent:

```
Function min-value(state)
   $v \leftarrow +\infty$  ;
  for each successor of state do
    |  $v \leftarrow \min(v, \text{max} - \text{value}(\text{successor}))$  ;
    | return v;
  end
```

$\alpha - \beta$ Pruning

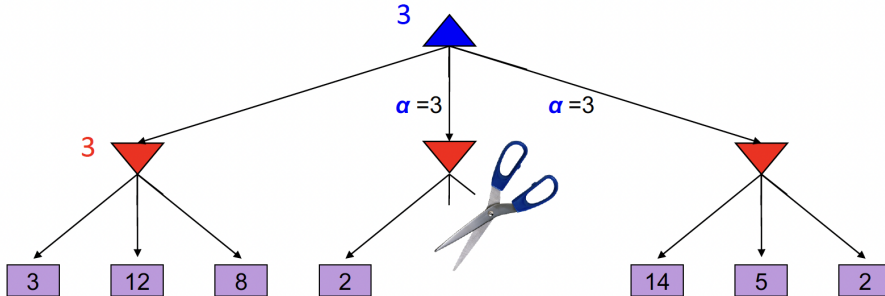


$\alpha - \beta$ Pruning



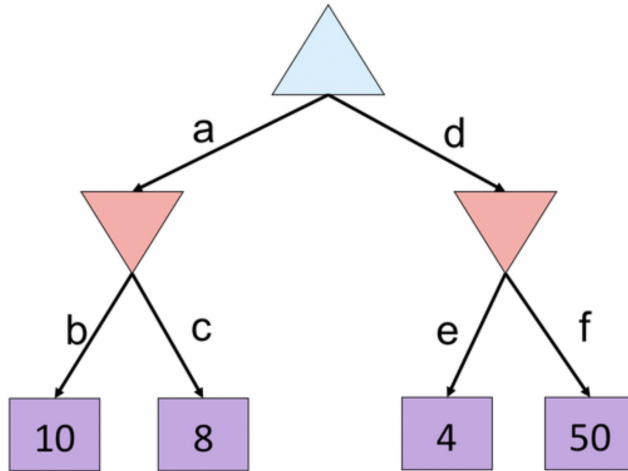
$\alpha - \beta$ Pruning

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

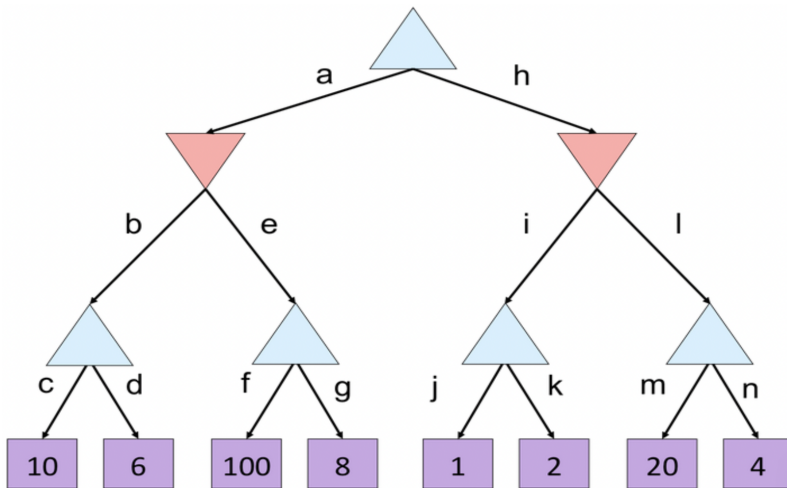


- ***The order of generation matters***: more pruning is possible if good moves come first

$\alpha - \beta$ Pruning



$\alpha - \beta$ Pruning



$\alpha - \beta$ Pruning

$\alpha - \beta$ IMPLEMENTATION

Dispatch:

α : MAX's best option on path to root

β : MIN's best option on path to root

MAX Agent:

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

MIN Agent:

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