

# *Inteligencia Artificial*

## Búsqueda adversarial y Árboles de juegos

Aitziber Atutxa

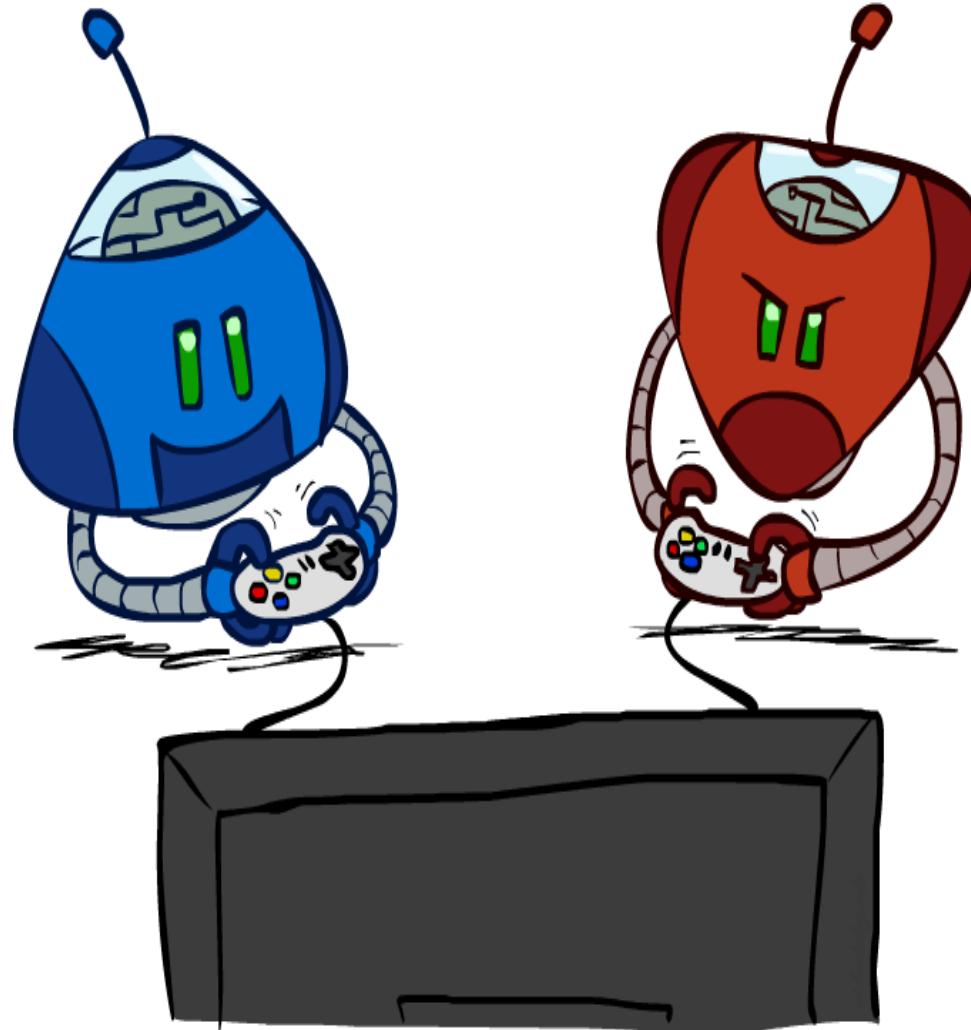
[transparencias de Koldo Gojenola y Ekaitz Jauregi adaptadas de Berkeley: Dan Klein, Pieter Abbeel]



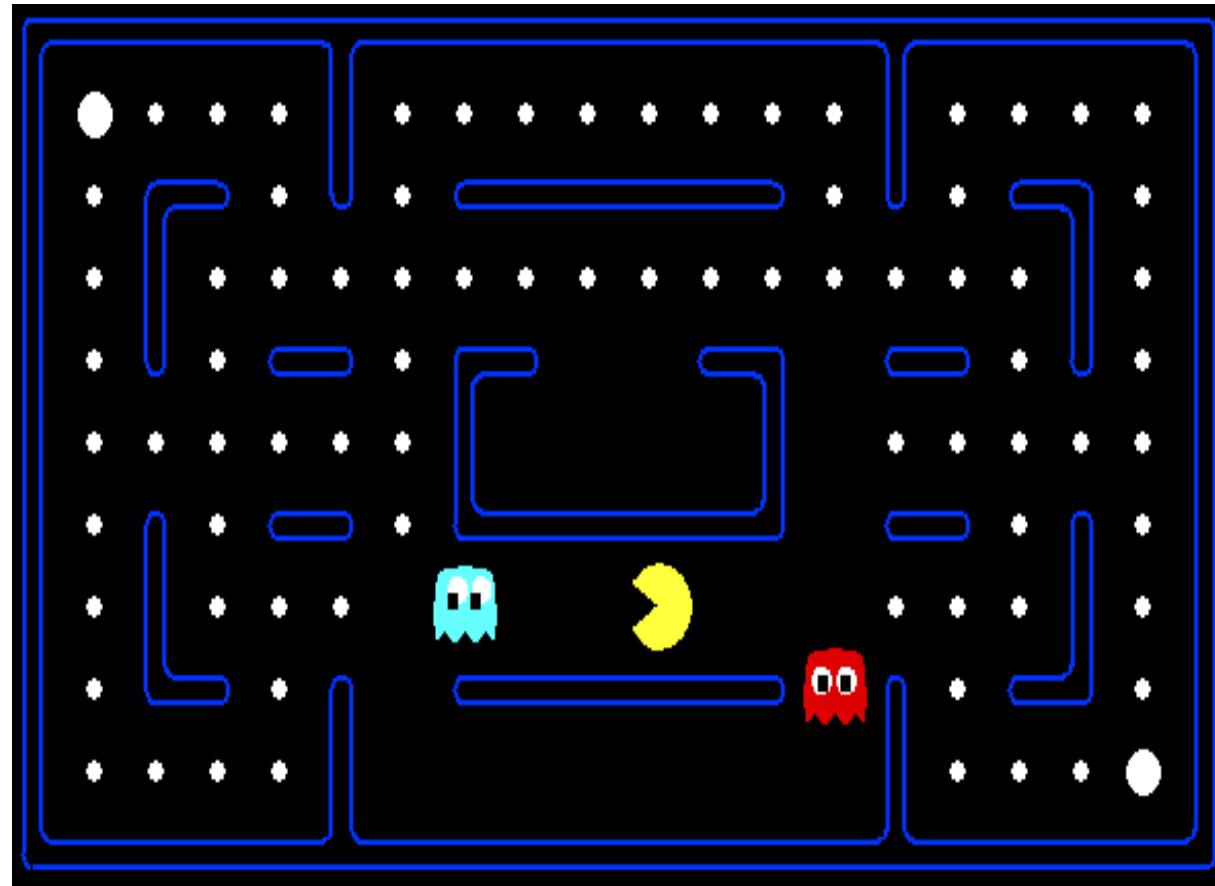
Universidad  
del País Vasco

Euskal Herriko  
Unibertsitatea

# Búsqueda adversarial y Árboles de juegos

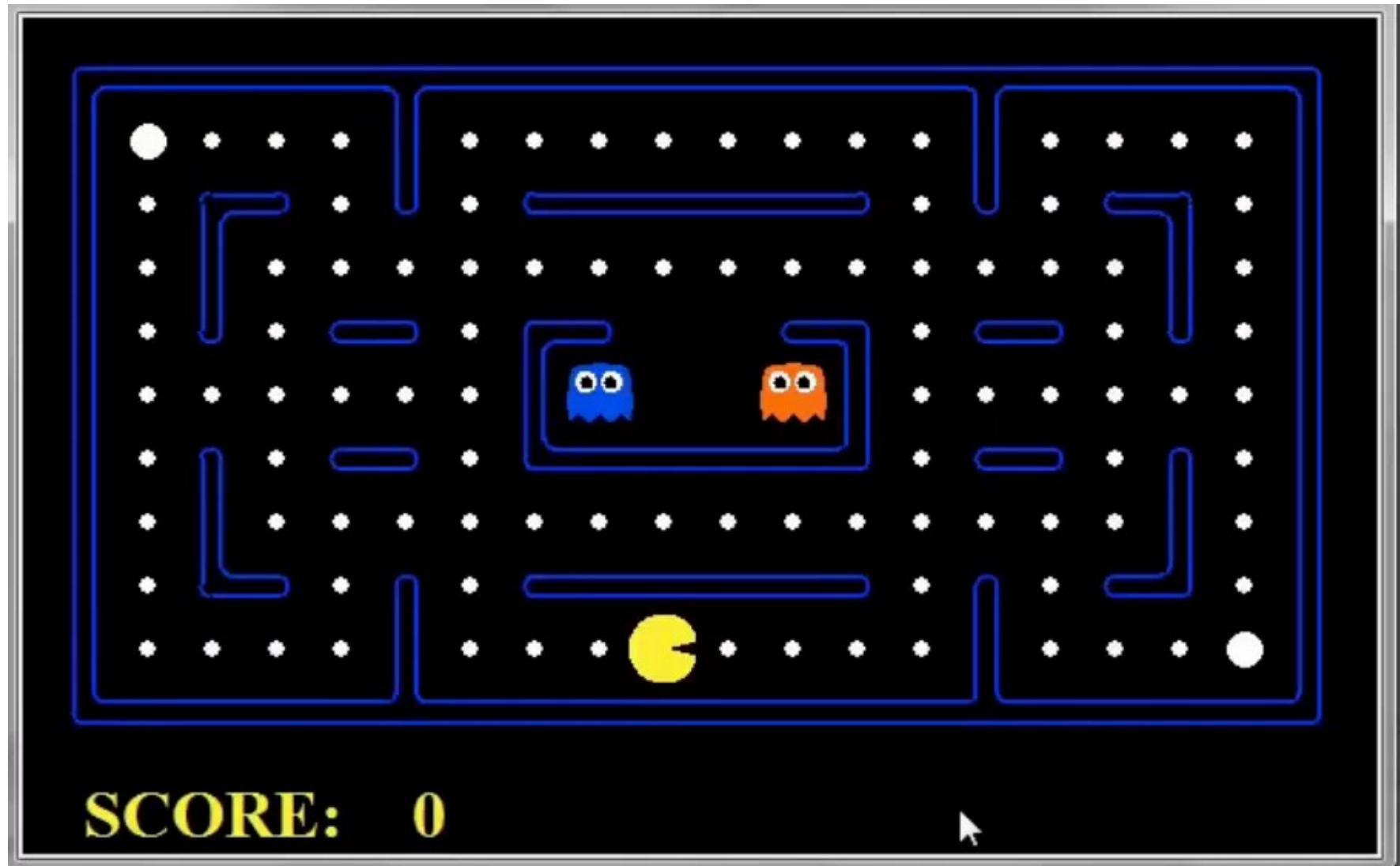


# Comportamiento y computación

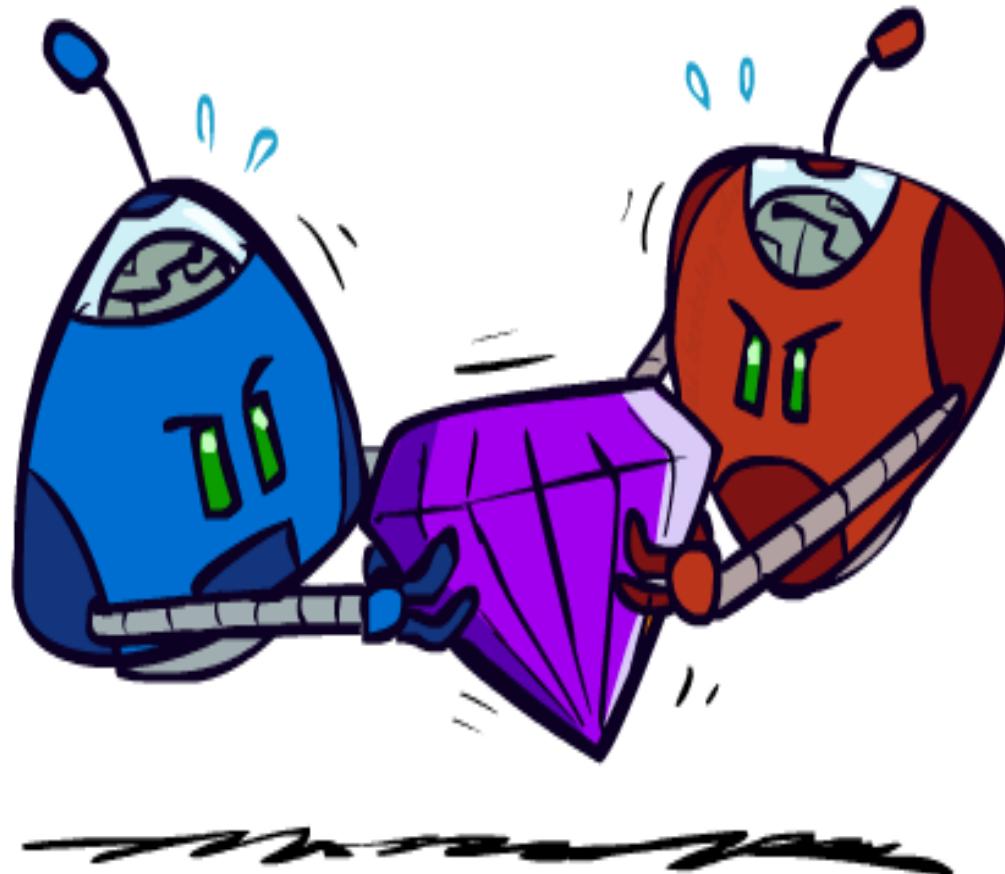


[Demo: mystery pacman (L6D1)]

# Video Demo Mystery Pacman

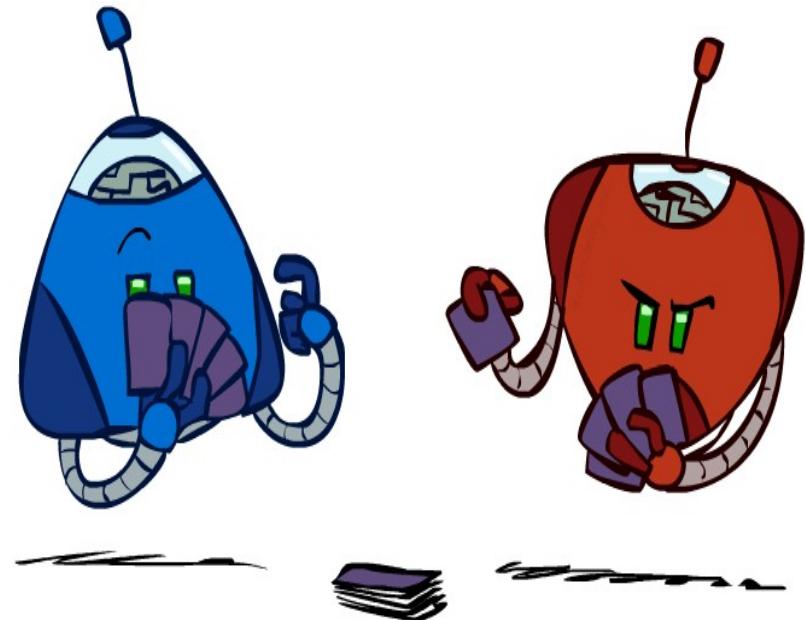


# Juegos Adversariales



# Tipos de juegos

- ¡Hay muchas clases de juegos!
- A tener en cuenta:
  - ¿Determinístico o estocástico?
  - ¿Uno, dos o más jugadores?
  - ¿Suma cero?
  - ¿Información perfecta (podemos ver el estado)?
- Queremos algoritmos para calcular una estrategia (política o policy) que recomiende un movimiento desde cada estado

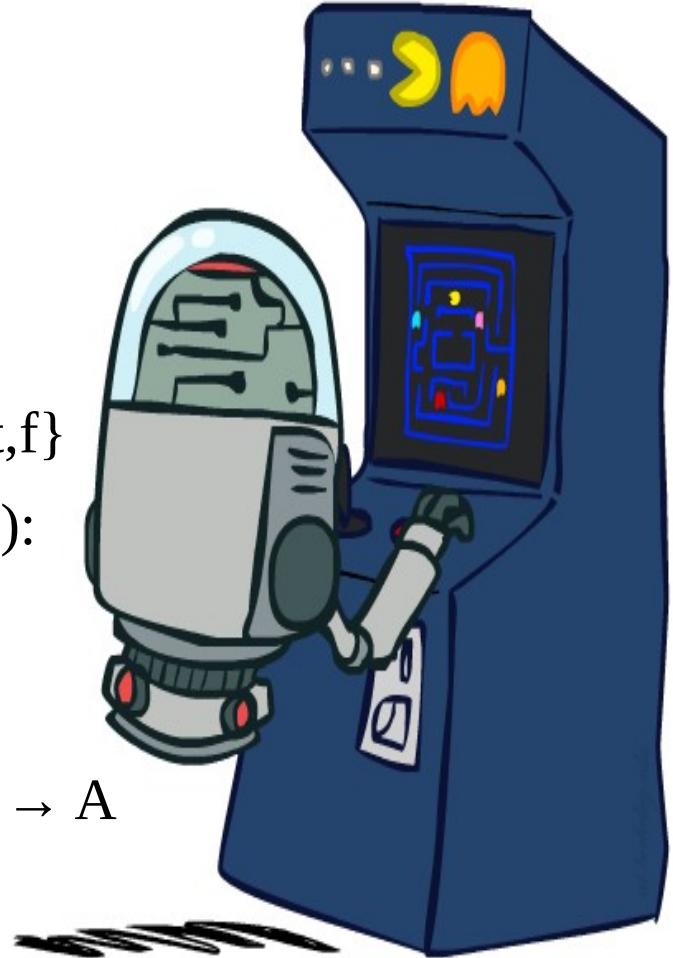


# Juegos determinísticos

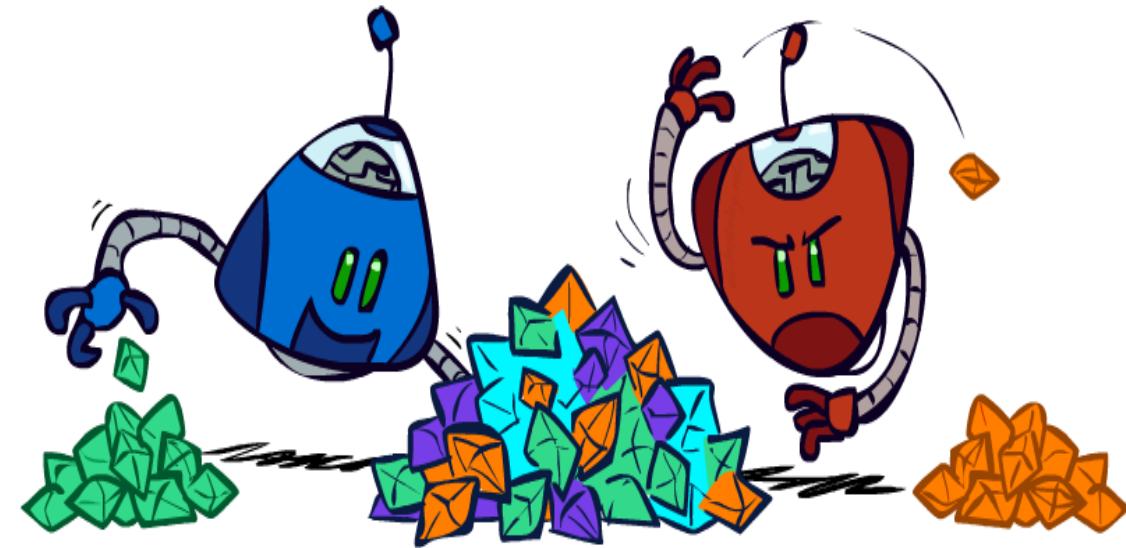
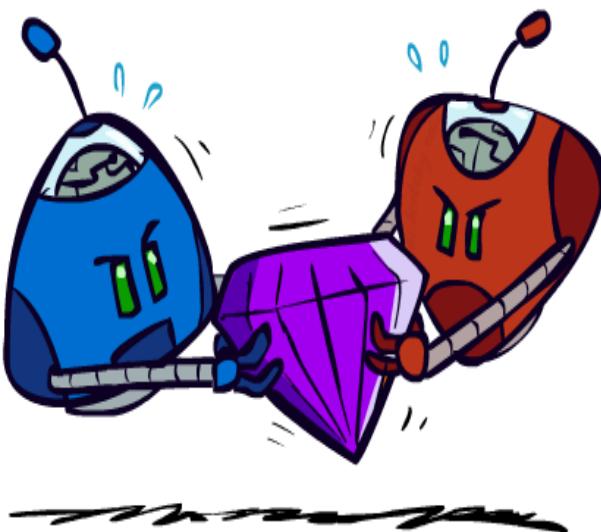
➤ Muchas formalizaciones posibles, una es:

- Estados:  $S$  (inicio en  $s_0$ )
- Jugadores:  $P=\{1\dots N\}$  (normalmente a turnos)
- Acciones:  $A$  (puede depender del jugador / estado)
- Función de transición:  $S \times A \rightarrow S$
- Test de terminación (estado objetivo o final):  $S \rightarrow \{t,f\}$
- Función de utilidad para terminal (Terminal Utilities):
- ¿Qué valor tiene este estado final
- para un jugador?  $S \times P \rightarrow R$

➤ La solución para un jugador es una política (**policy**):  $S \rightarrow A$



# Juegos de suma cero



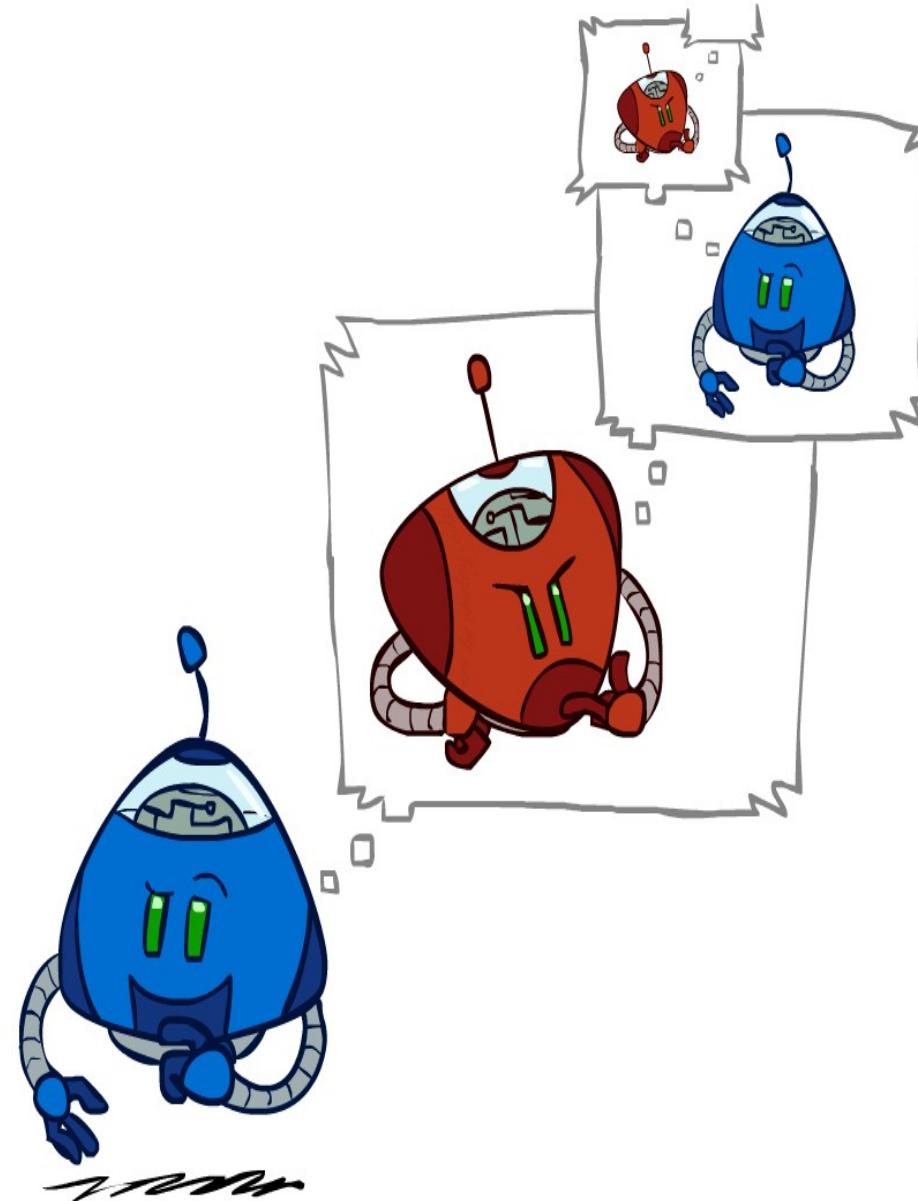
## ➤ Juegos de suma cero

- Los agentes tienen utilidades opuestas (valores)
- Podemos pensar en un único valor que uno maximiza y el otro minimiza
- Adversarial, competición pura

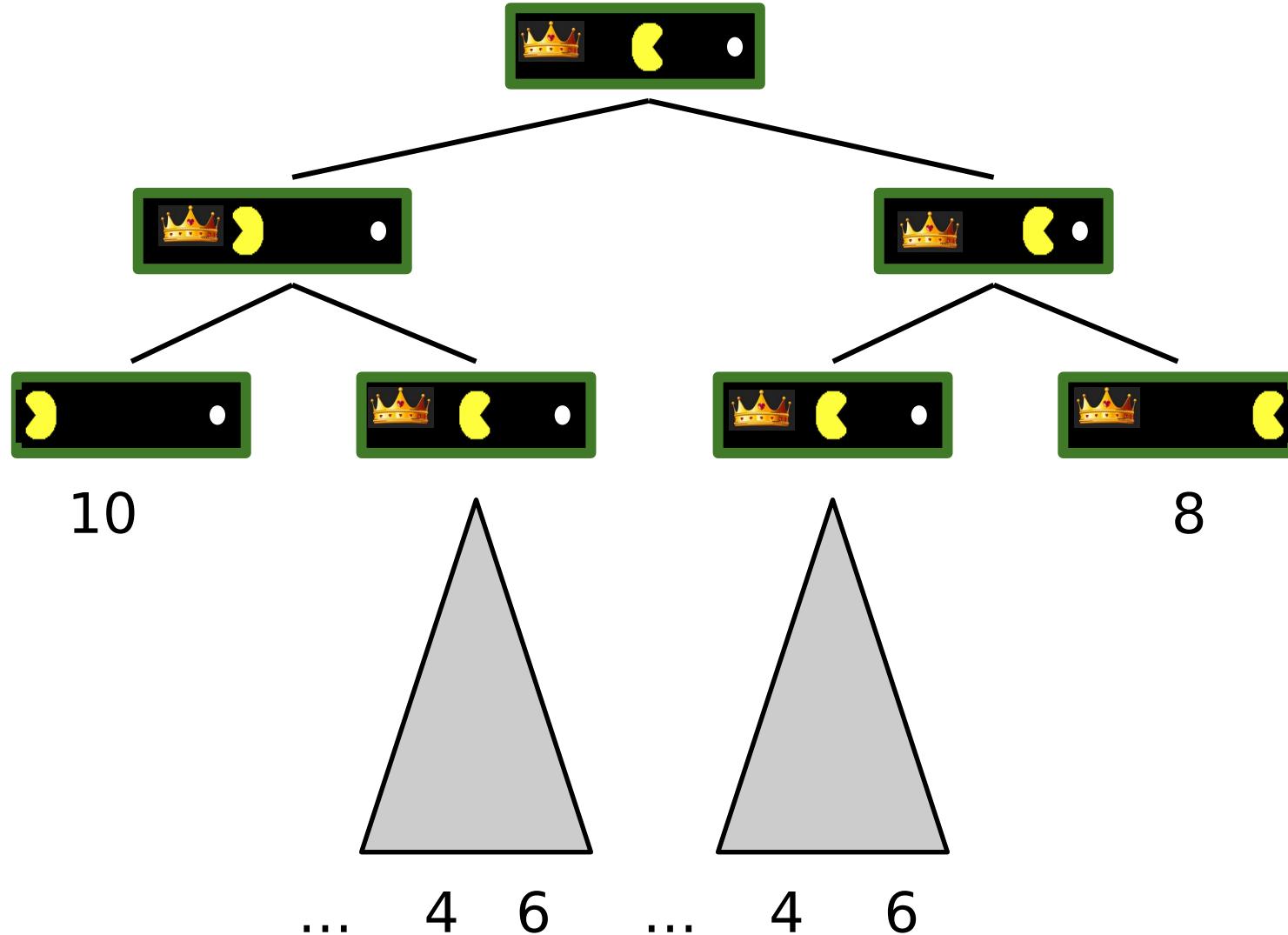
## ➤ Juegos Generales

- Los agentes tienen utilidades independientes (valores)
- Cooperación, indiferencia, competición, y más, todo es posible
- Más después

# Búsqueda Adversarial



# Árboles con un solo Agente

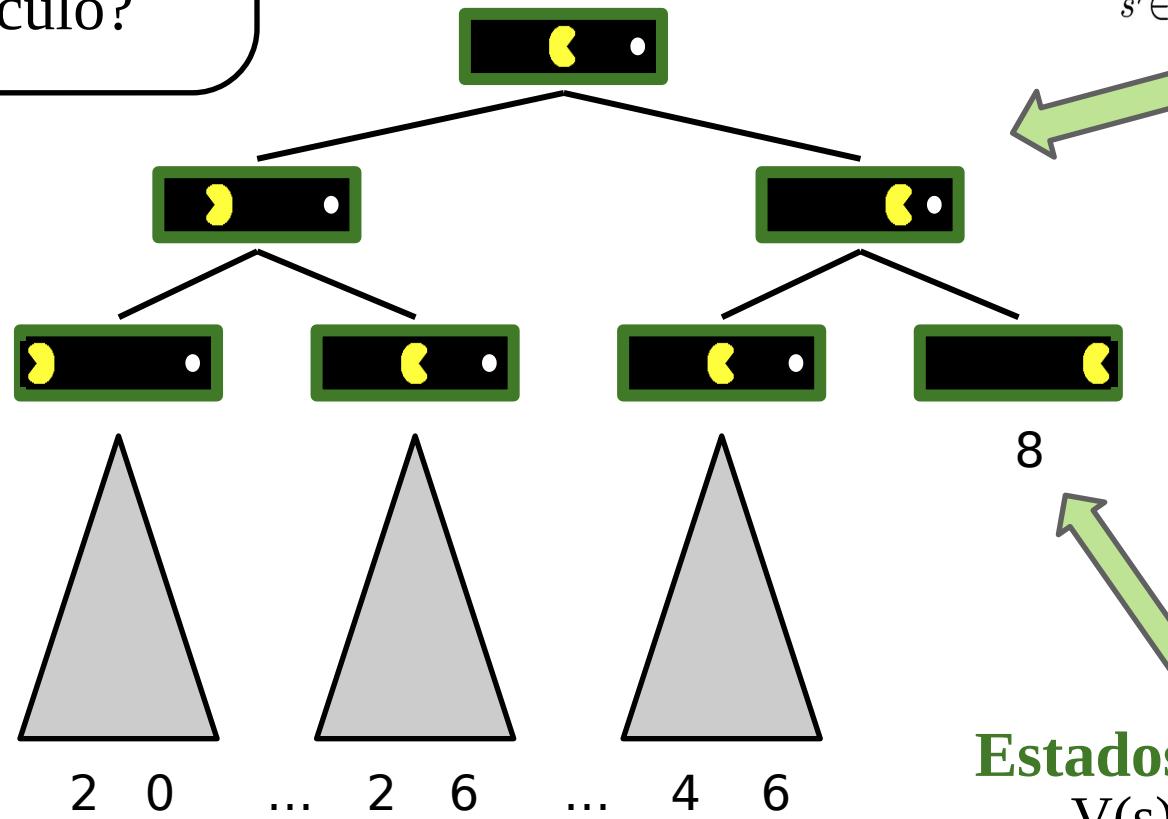


# Valor de un estado

Valor de un estado: El mejor resultado (utility) desde ese estado.  
¿Cómo lo calculo?

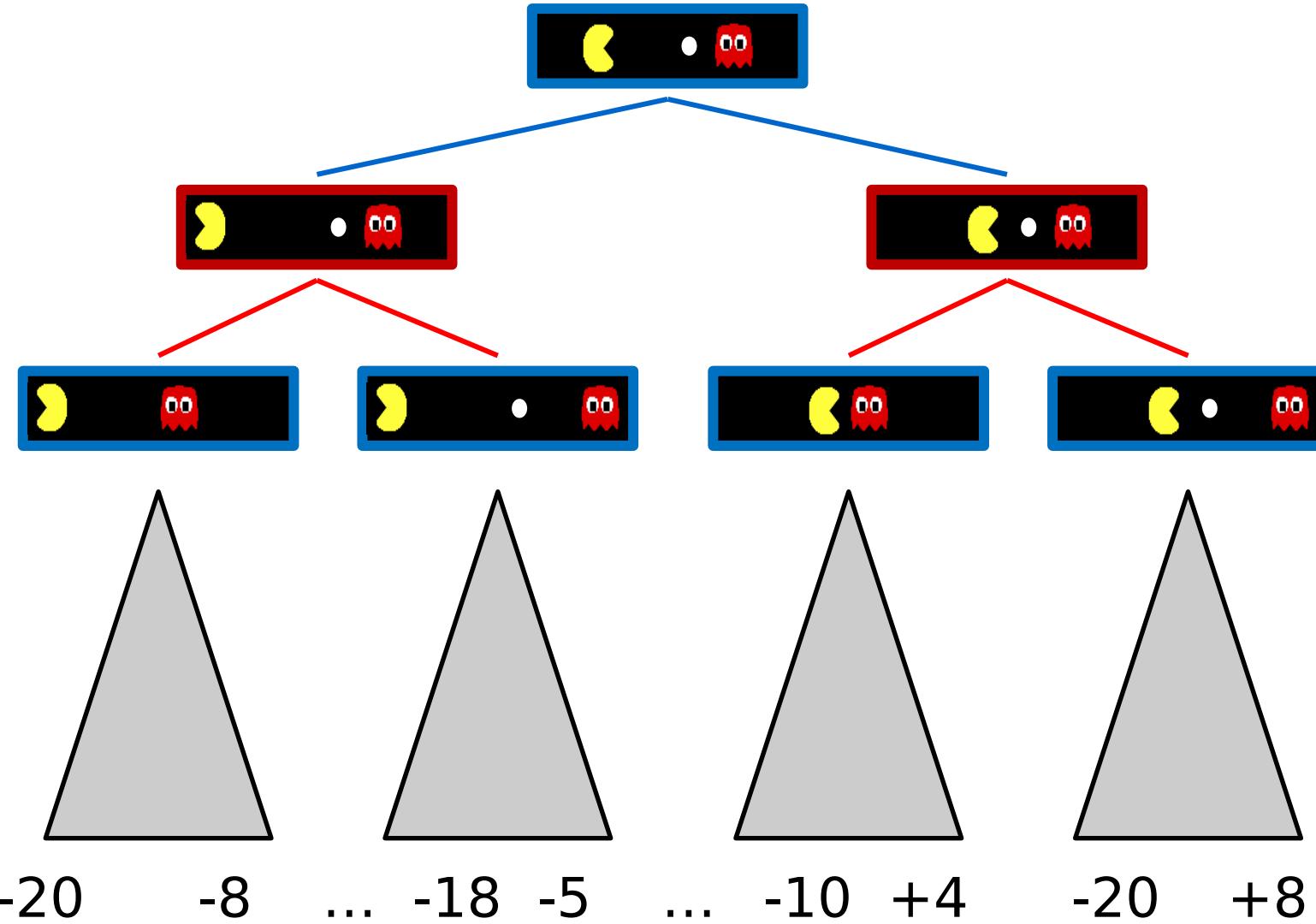
Estados no terminales:

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



Estados terminales:  
 $V(s) = \text{conocido}$

# Árboles de estados Adversariales



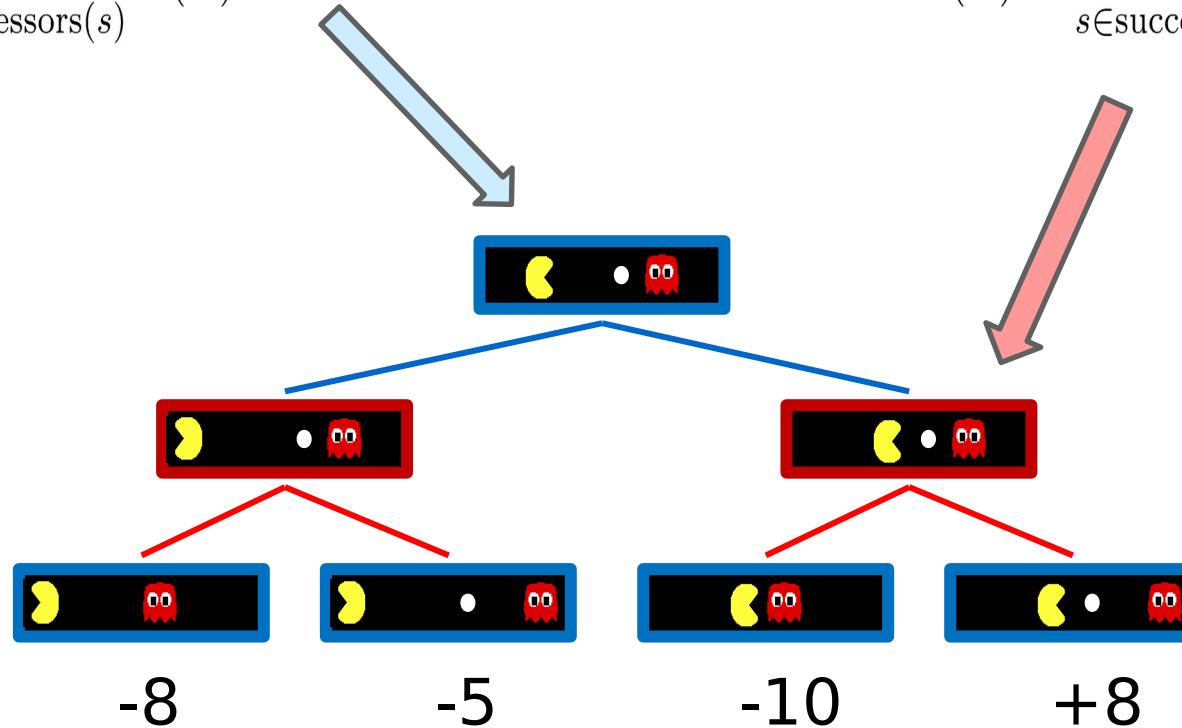
# Valores Minimax

Estados bajo el control del agente:

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

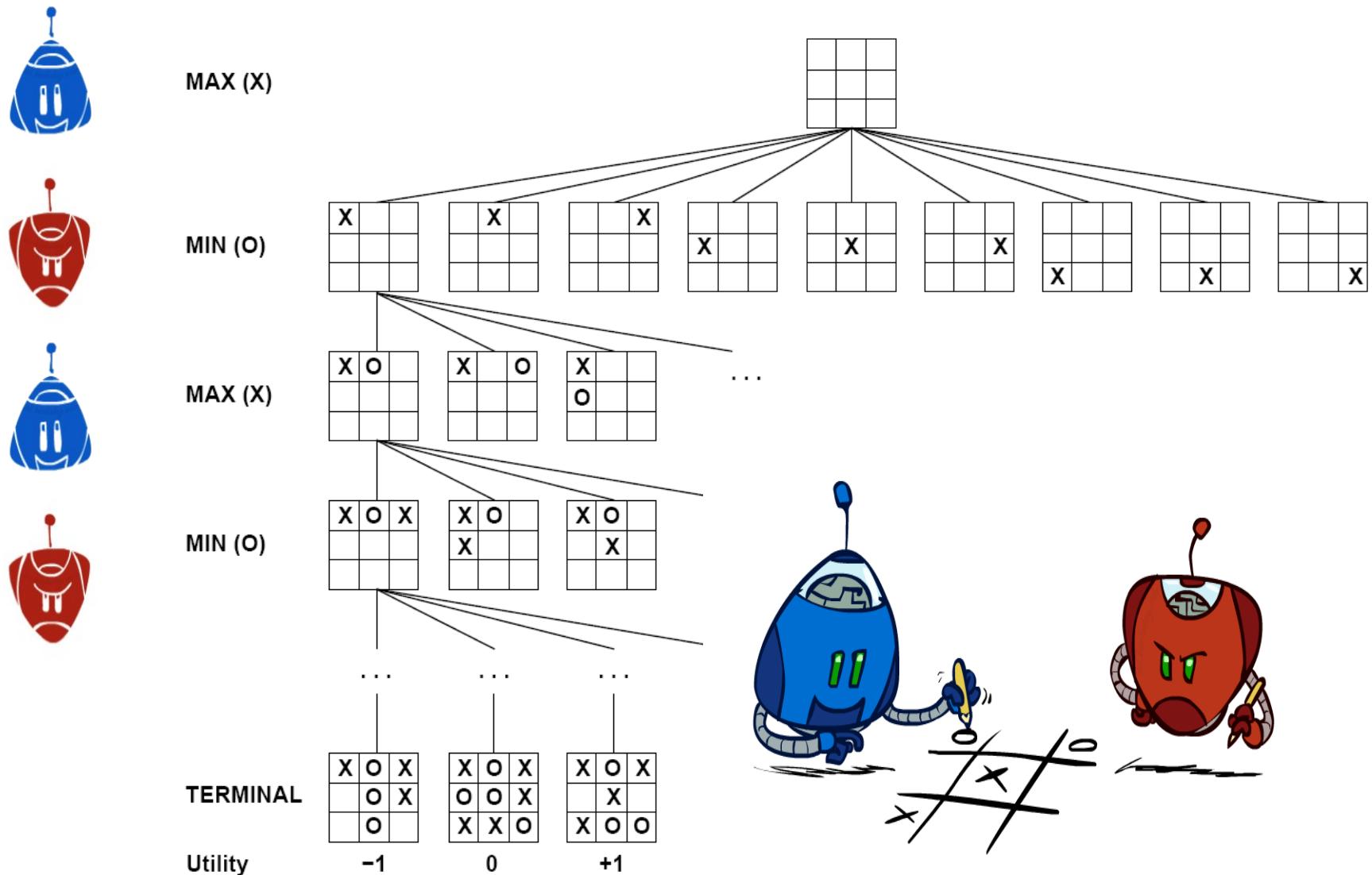
Estados bajo el control del oponente:

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



Estados terminales:  
 $V(s) = \text{conocido}$

# Árbol de juegos de Tic-Tac-Toe



# Búsqueda Adversarial (Minimax)

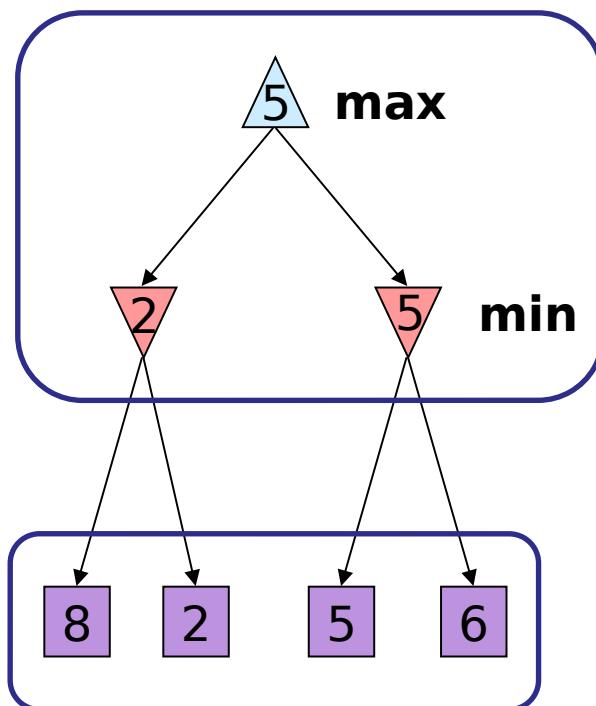
## ➤ Juegos determinísticos, de suma cero:

- Tic-tac-toe, ajedrez, damas
- Un jugador maximiza el resultado
- El otro minimiza el resultado

## ➤ Búsqueda Minimax:

- Árbol de búsqueda en un espacio de estados
- Los jugadores alternan turnos
- Se calcula el **valor minimax** de cada nodo: el máximo resultado (utility) posible contra un adversario racional (óptimo)

**Valores Minimax:**  
calculados **recursivamente**



**Valores terminales:**  
parte del juego

# Implementación de Minimax

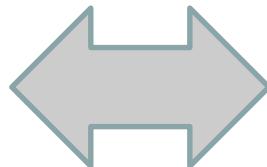
```
def max-value(state):
```

    initialize v = -∞

    for each successor of state:

        v = max(v, min-value(successor))

    return v



```
def min-value(state):
```

    initialize v = +∞

    for each successor of state:

        v = min(v, max-value(successor))

    return v

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

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

El valor v empieza con  
-∞ y va aumentando

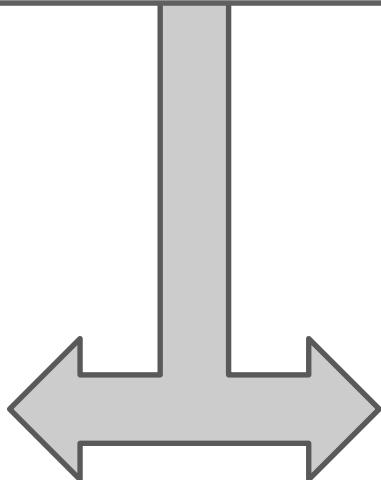
El valor v empieza con  
+∞ y va disminuyendo

# Implementación de Minimax (Dispatch)

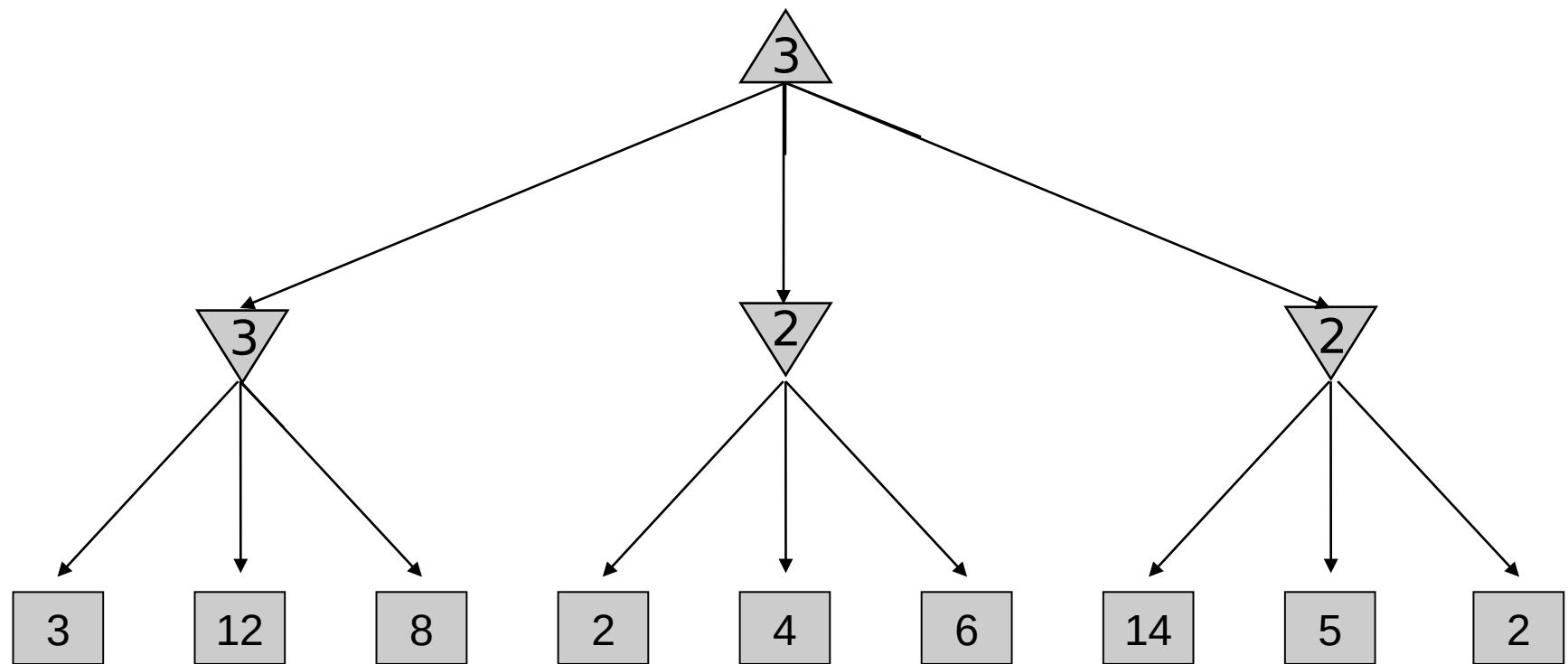
```
def value(state):  
    if the state is a terminal state: return the state's utility  
    if the next agent is MAX: return max-value(state)  
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):  
    initialize v = -∞  
    for each successor of state:  
        v = max(v,  
                 value(successor))  
    return v
```

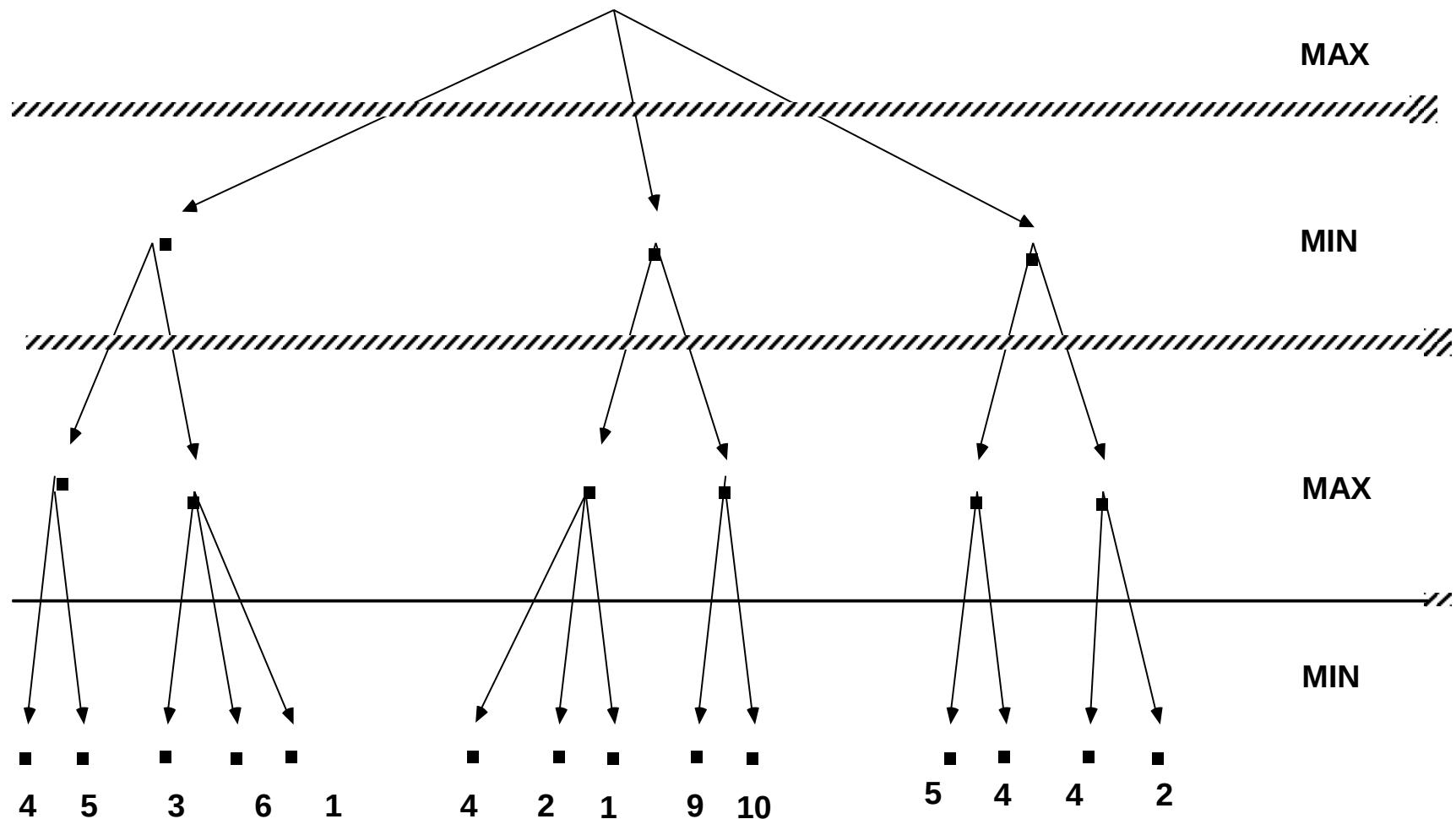
```
def min-value(state):  
    initialize v = +∞  
    for each successor of state:  
        v = min(v,  
                 value(successor))  
    return v
```



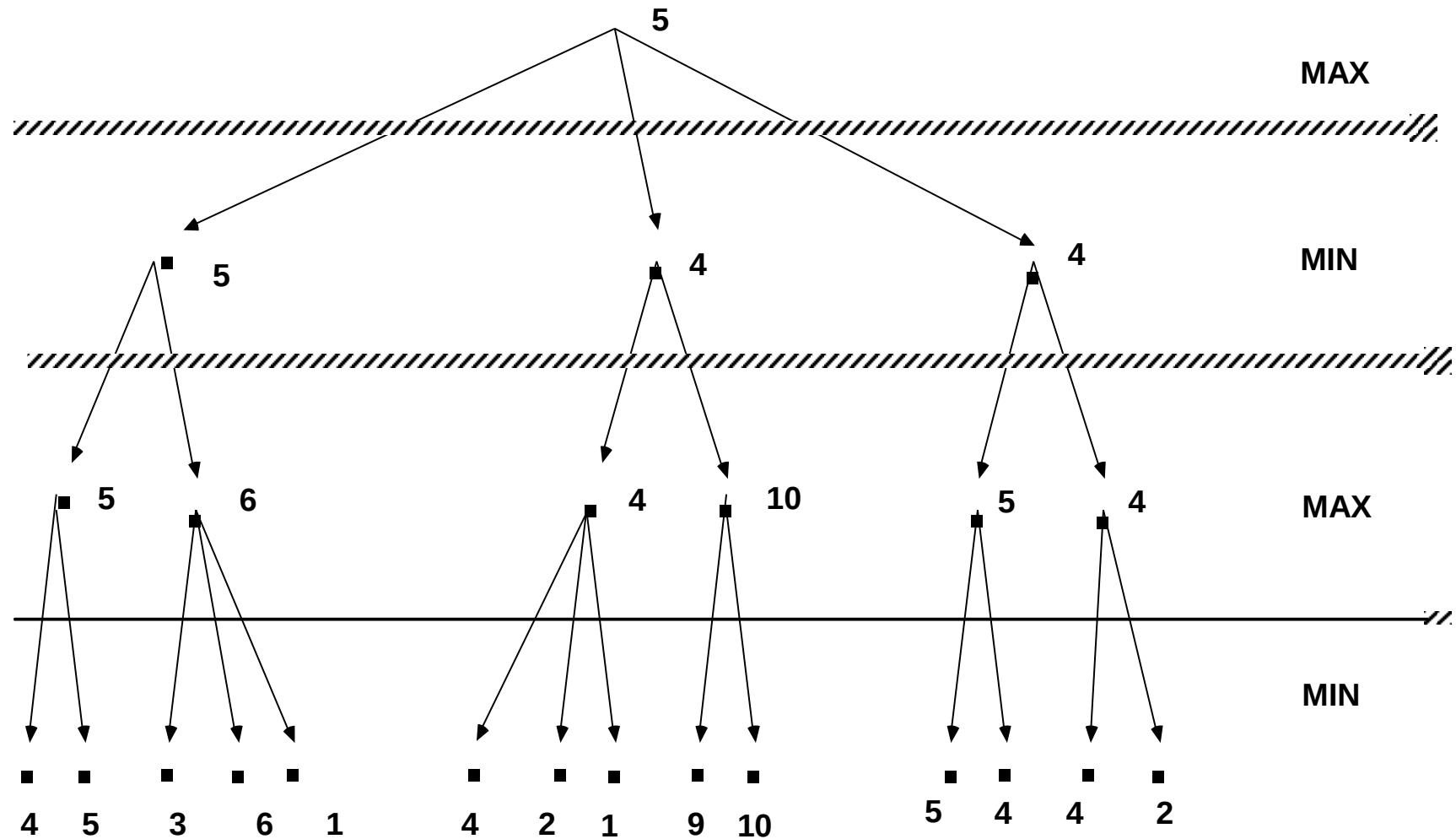
# Ejemplo de Minimax



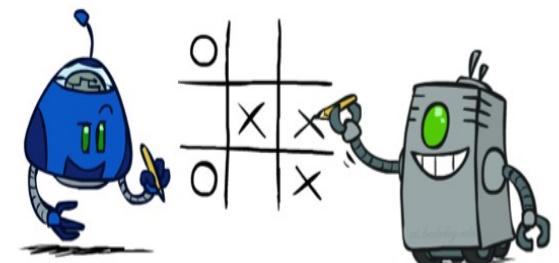
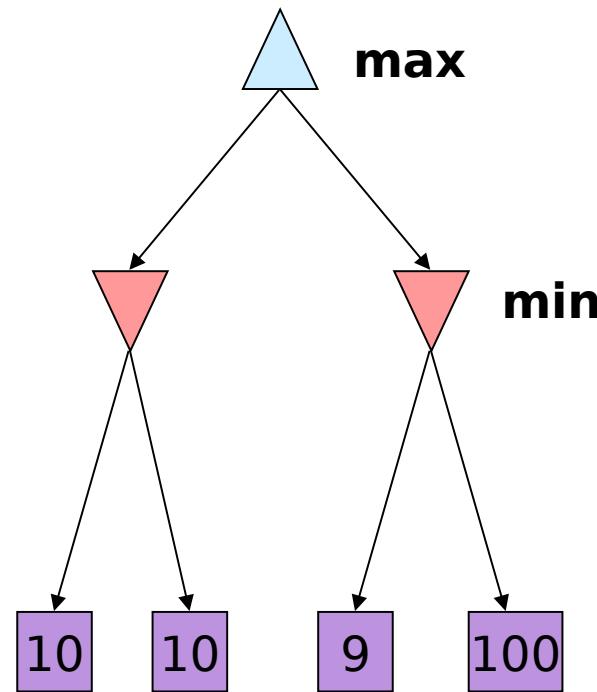
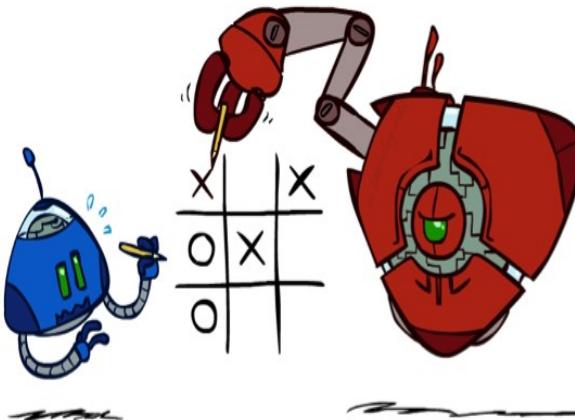
# Árbol de una aplicación del mini-max



# Árbol de una aplicación del mini-max



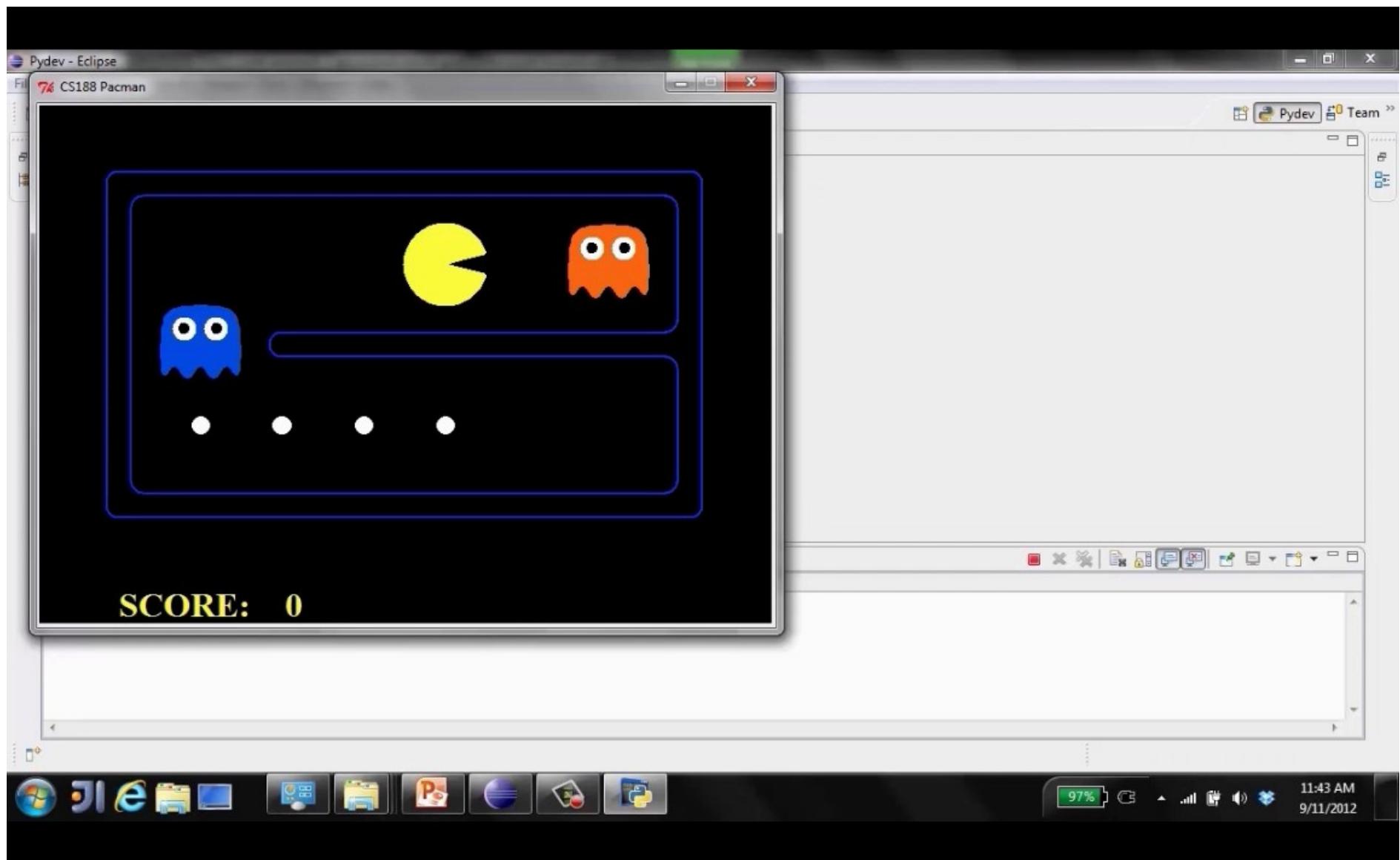
# Propiedades de Minimax



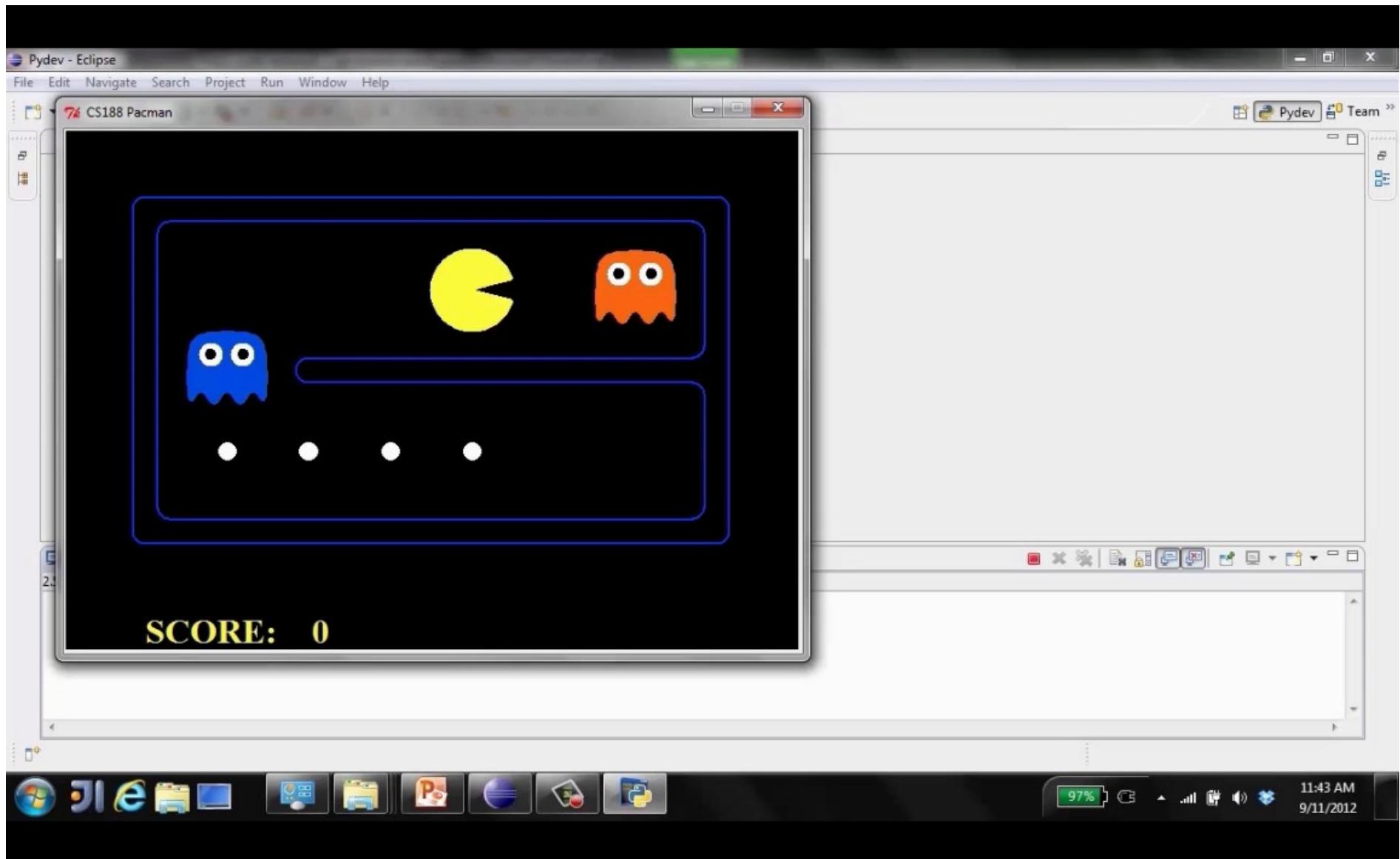
Óptimo contra un jugador perfecto. ¿En otro caso?

[Demo: min vs exp (L6D2, L6D3)]

# Video Demo Minimax



# Video Demo Minimax vs. Exp (Exp)



# Eficiencia de Minimax

- ¿Cómo de eficiente es minimax?
  - Igual que (exhaustivo) DFS
  - Tiempo:  $O(b^m)$
  - Espacio:  $O(b^m)$
- Ejemplo: para ajedrez,  $b \sim 35$ ,  $m \sim 100$ 
  - Una solución exacta es inviable
  - Pero, ¿Tenemos que explorar el árbol entero?

