Ayudantía 5

Presentación Tarea 3, MiniMax

Por Bernardita Alliende y Anaís Montanares

3 octubre, 2025

Contenidos

1. Búsqueda Adversaria
2. Minimax
3. Tarea 3

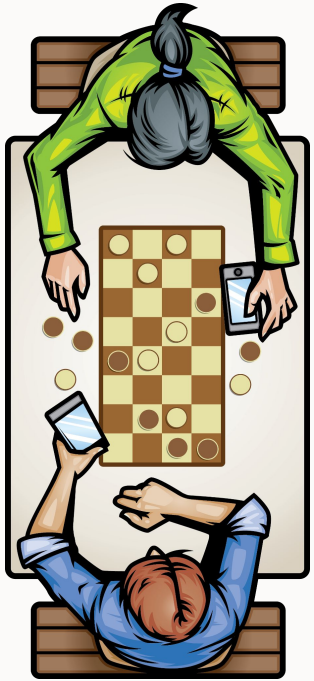
Búsqueda Adversaria

Problemas a desarrollar
Teoría de juegos

Problemas de dos jugadores, de turnos, **información perfecta** y **suma cero**.

- **Información perfecta:** Conocemos toda la información del tablero de juego (estados).

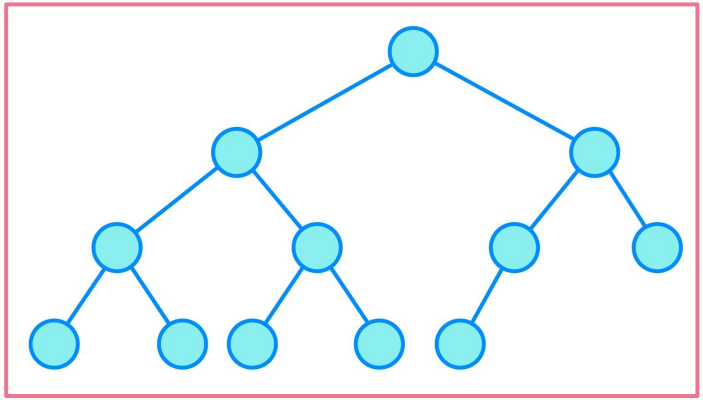
- **Suma cero:** Todo lo que me beneficia a mí, perjudica a mi contrincante en igual medida y viceversa.



Búsqueda

Podemos considerar un juego como un **espacio de búsqueda**:

- **Estado del juego** = nodo
- **Jugadas/acciones** = aristas



El **objetivo** será un **estado donde ganamos** (estado objetivo):

- Gato: Tres fichas (nuestras) en línea
- Conecta-4: Cuatro fichas (nuestras) en línea
- Ajedrez: Jaque-mate al rey del oponente

Adversario

Yo **no sé** cómo va a jugar mi adversario, pero **asumo que siempre juega lo que es mejor para él.** En consecuencia, considero que **juega lo peor para mí**.

Luego, si mi adversario toma decisiones subóptimas durante el juego, solo es mejor para mí.

Algoritmo Minimax

La búsqueda puede presentarse como una simulación del juego entre dos jugadores:

Max y **Min**.

Si es mi turno: **Max**

- Elijo la mejor jugada que tengo disponible, para **max**imizar el puntaje.

Si es el turno de mi oponente: **Min**

- Asumo que mi oponente siempre toma las decisiones óptimas (para él) entonces elige la opción que **min**imiza mi puntaje (la mejor jugada para mi oponente es la que me “hace peor”).

Algoritmo Minimax

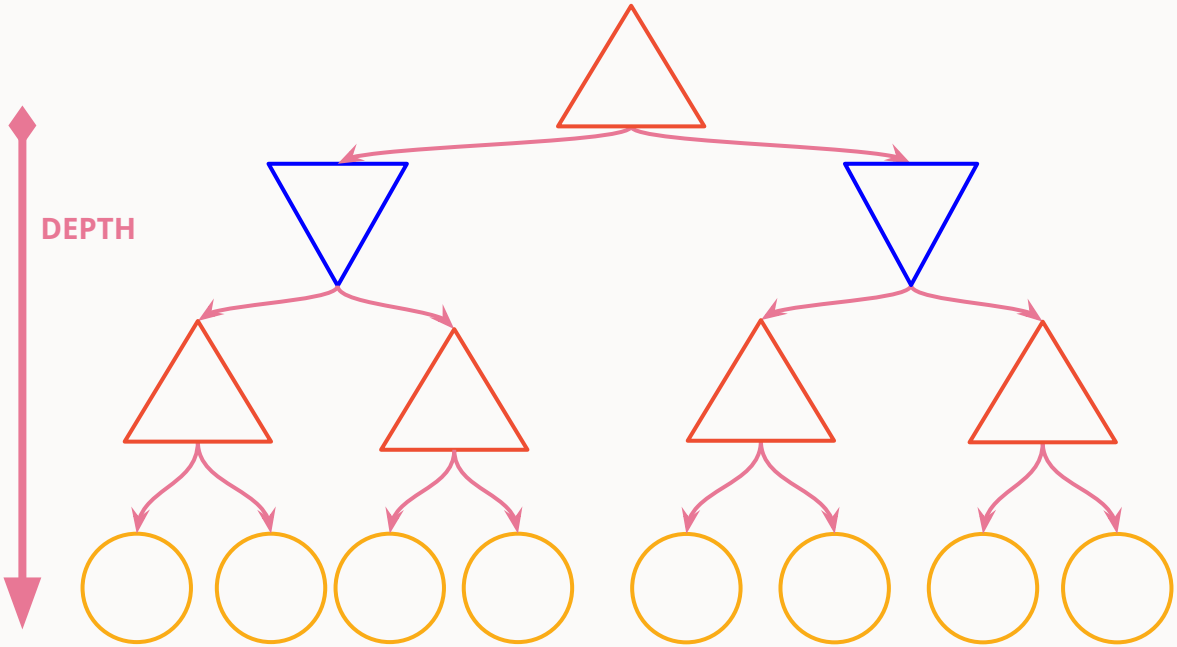
```
function MAX-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v, move ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
    if v2 > v then
      v, move ← v2, a
  return v, move

function MIN-VALUE(game, state) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v, move ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
    if v2 < v then
      v, move ← v2, a
  return v, move
```

Función de evaluación

- Es como una **heurística** pero en el contexto de búsqueda adversaria
- **Asigna puntajes a estados no terminales**, lo que permite limitar a un máximo la altura de un árbol

Básicamente, hace una estimación sobre “quién va ganando” llegada una altura máxima definida.

Cálculo del valor Minimax



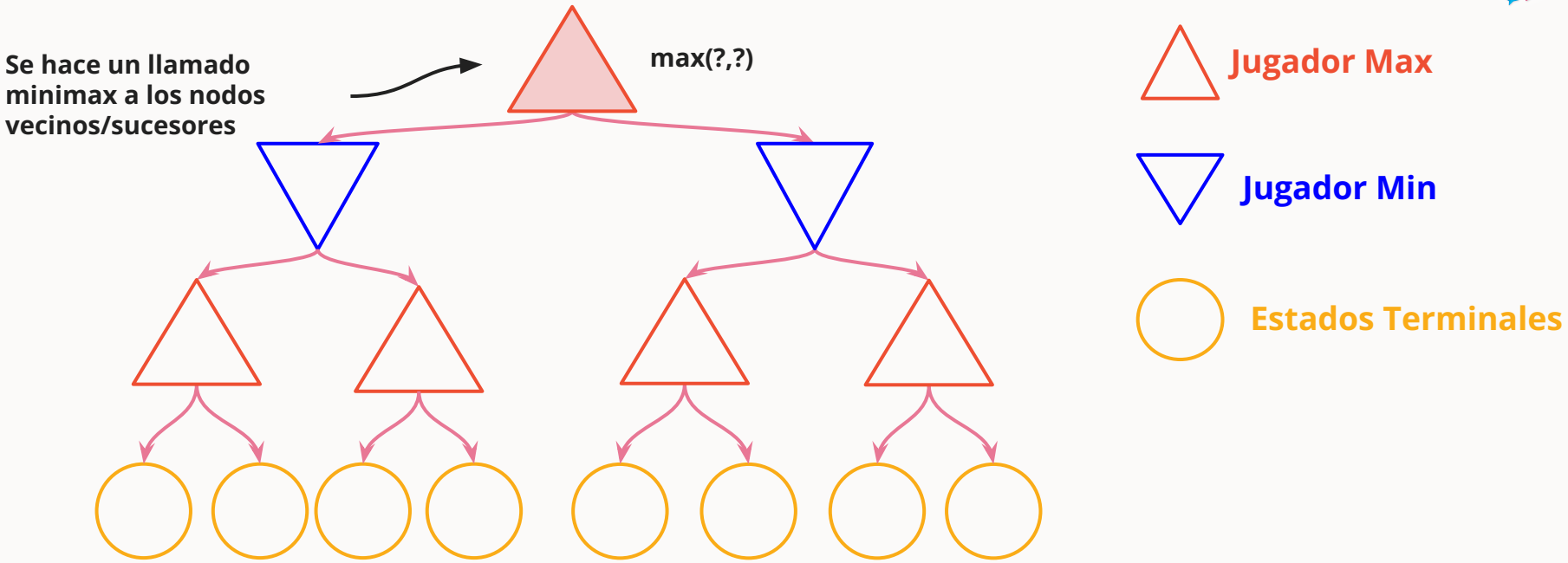
Jugador Max

Jugador Min

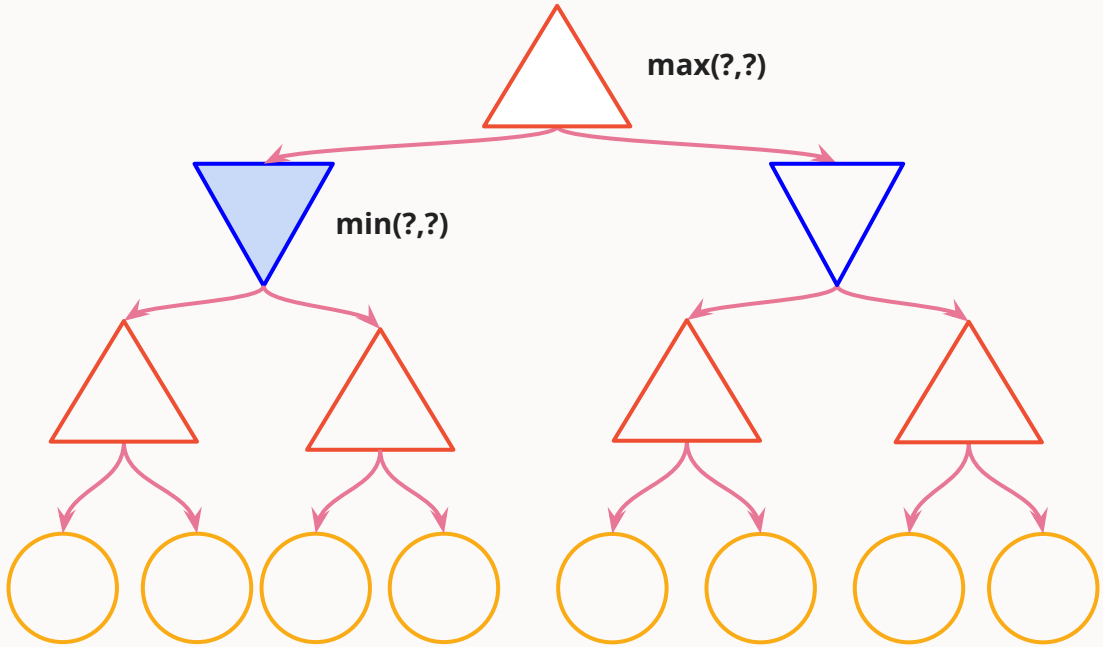
Estados Terminales

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



Cálculo del valor Minimax

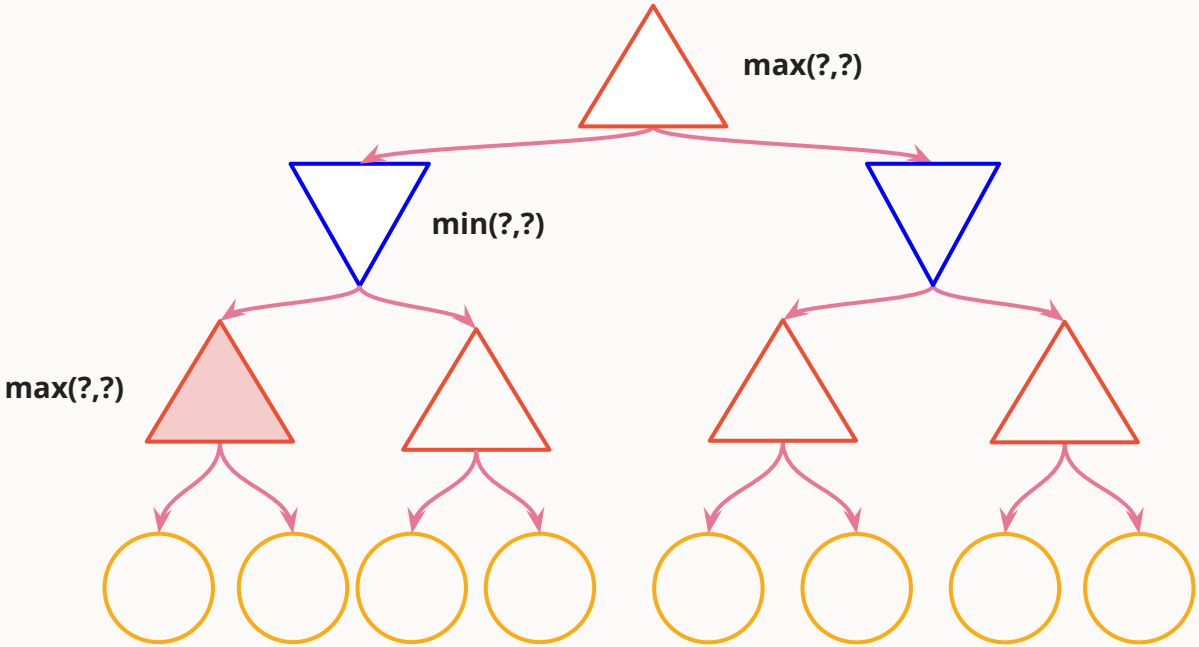


Jugador Max

Jugador Min

Estados Terminales

Cálculo del valor Minimax

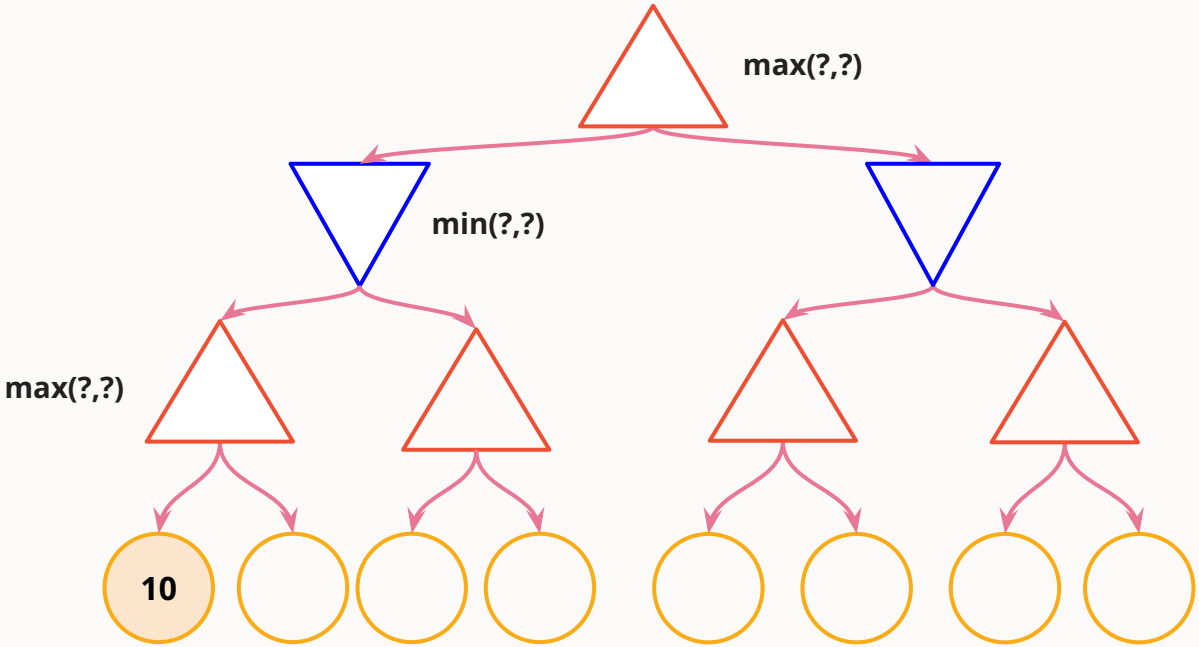


Jugador Max

Jugador Min

Estados Terminales

Cálculo del valor Minimax



Jugador Max

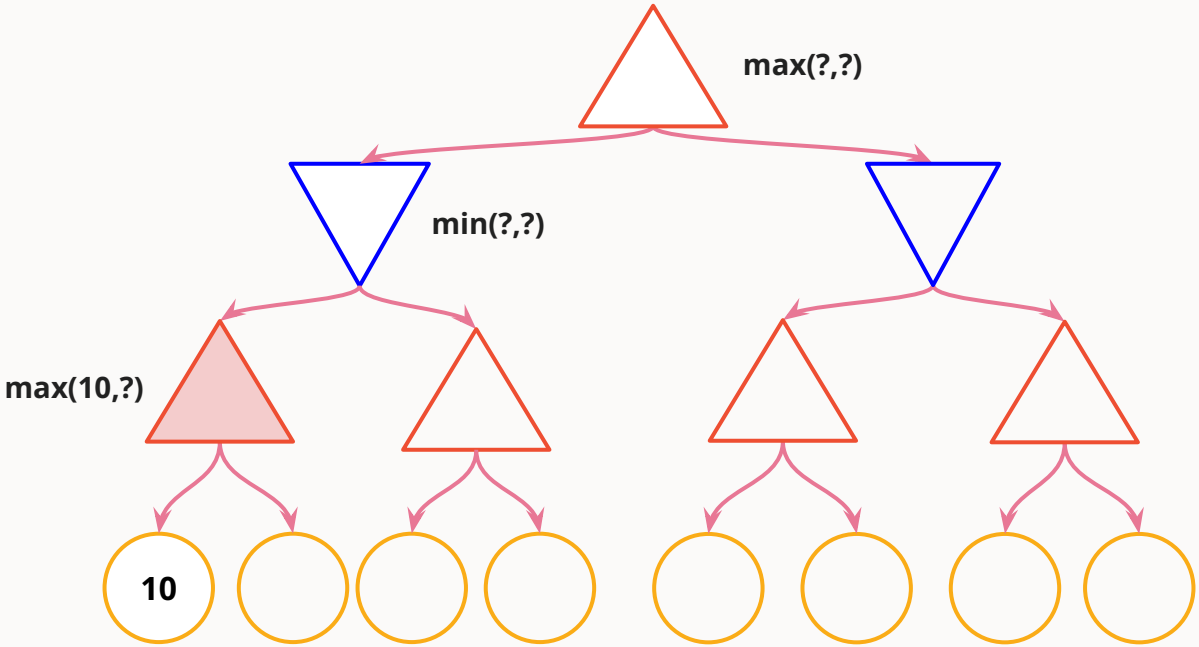
Jugador Min

Estados Terminales

Se calcula el score con la función de evaluación del estado terminal

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax

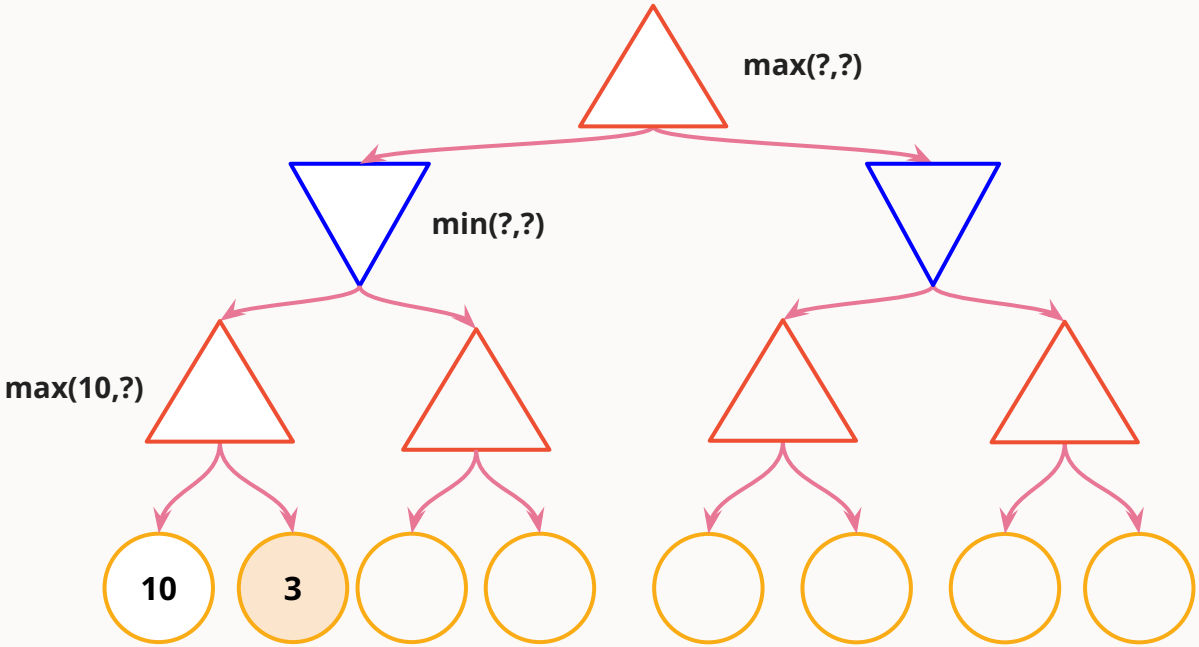


Jugador Max

Jugador Min

Estados Terminales

Cálculo del valor Minimax



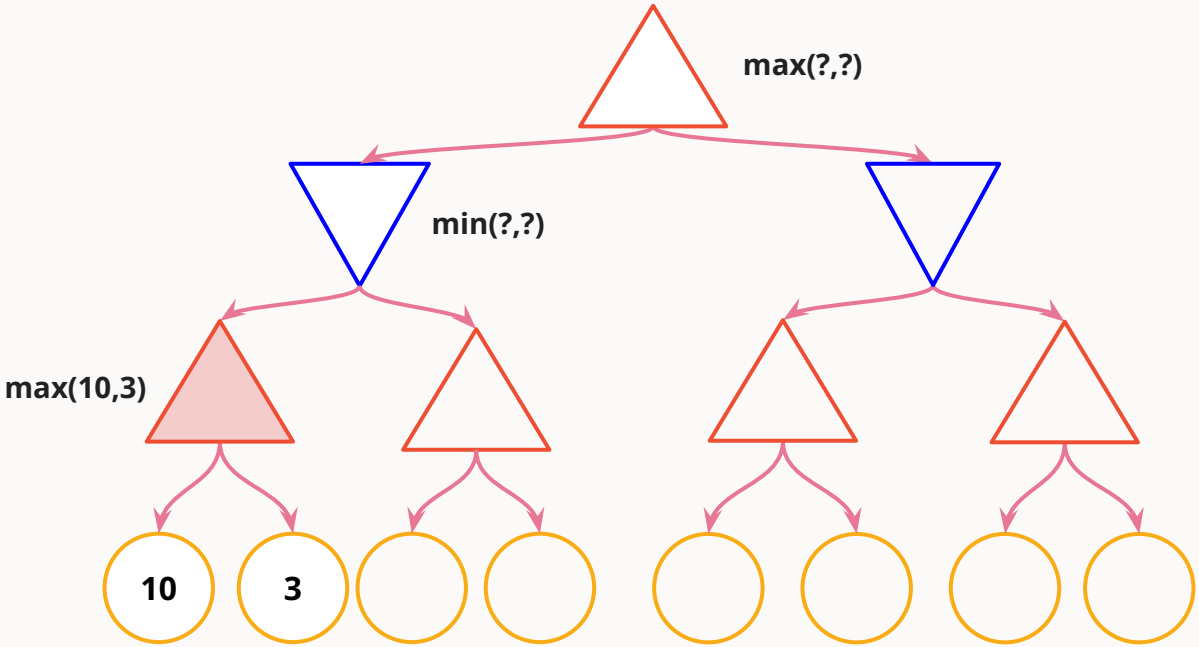
Jugador Max

Jugador Min

Estados Terminales

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



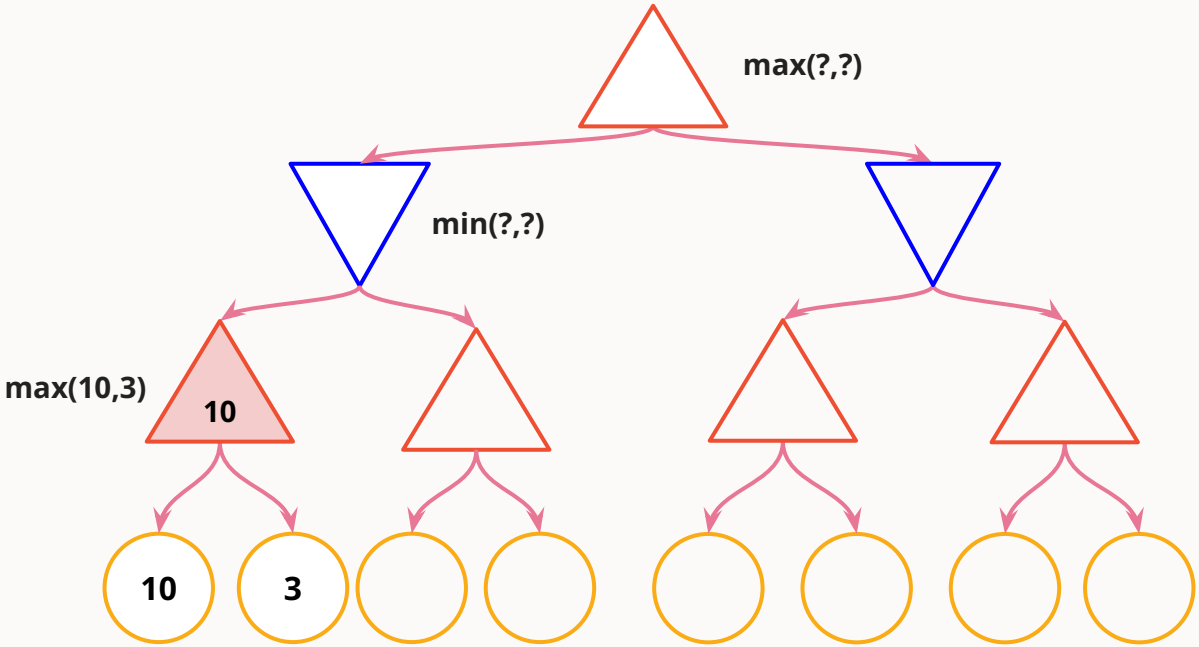
Jugador Max

Jugador Min

Estados Terminales

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



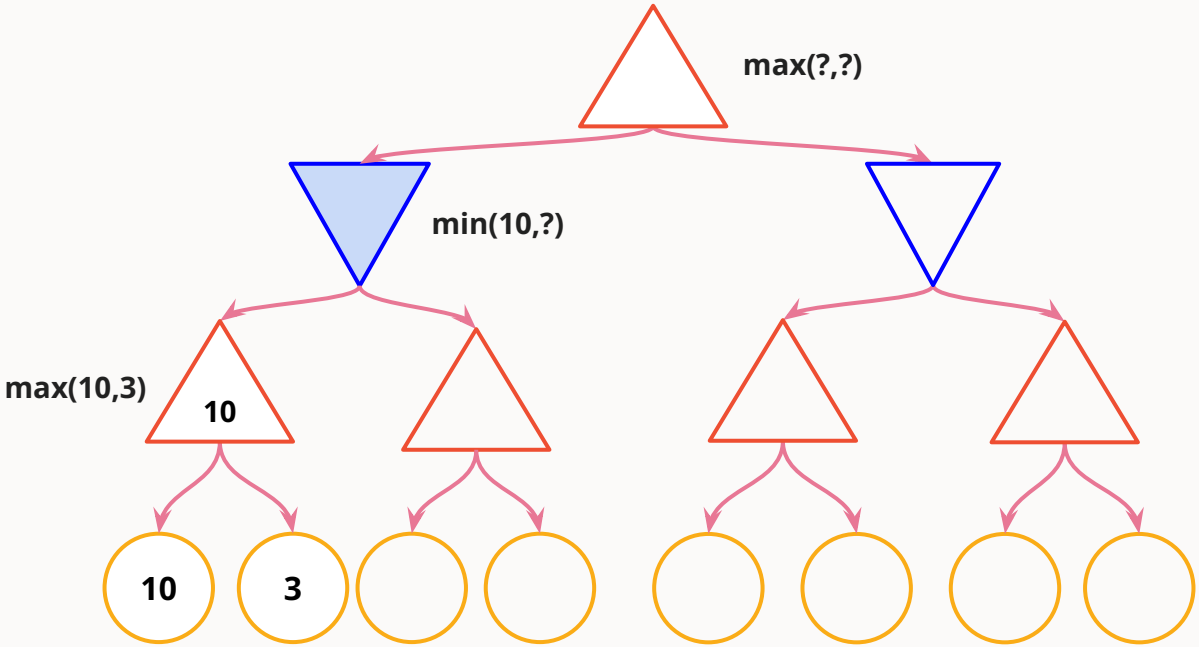
Jugador Max

Jugador Min

Estados Terminales

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax

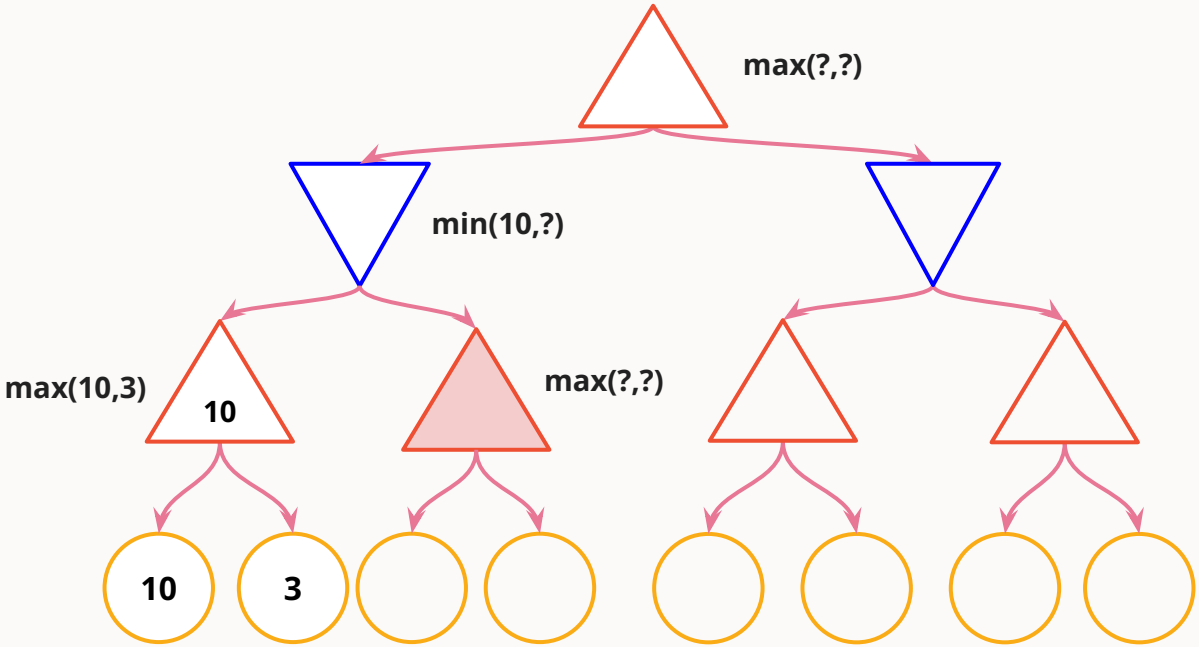


Jugador Max

Jugador Min

Estados Terminales

Cálculo del valor Minimax

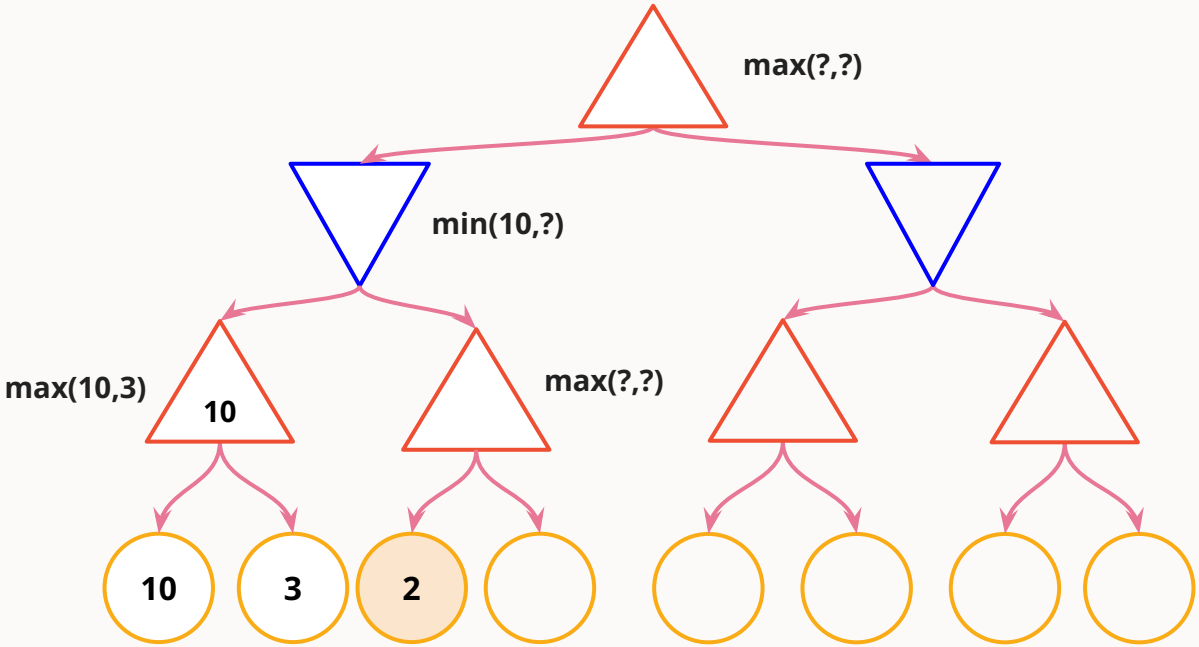


Jugador Max

Jugador Min

Estados Terminales

Cálculo del valor Minimax



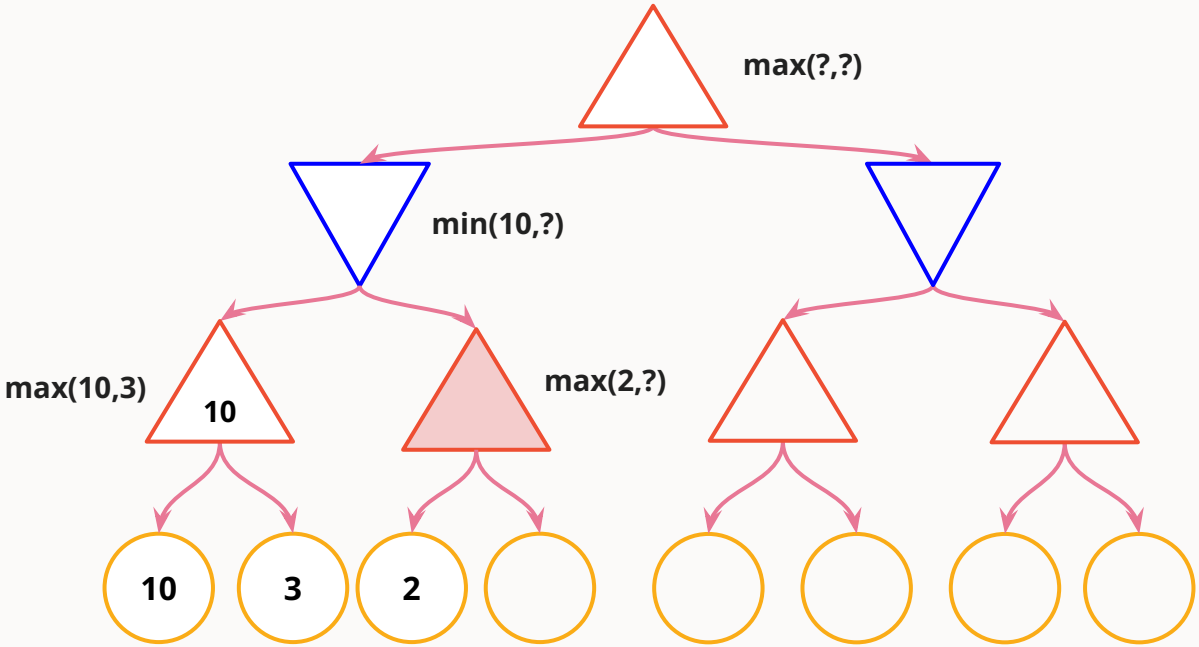
Jugador Max

Jugador Min

Estados Terminales

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax

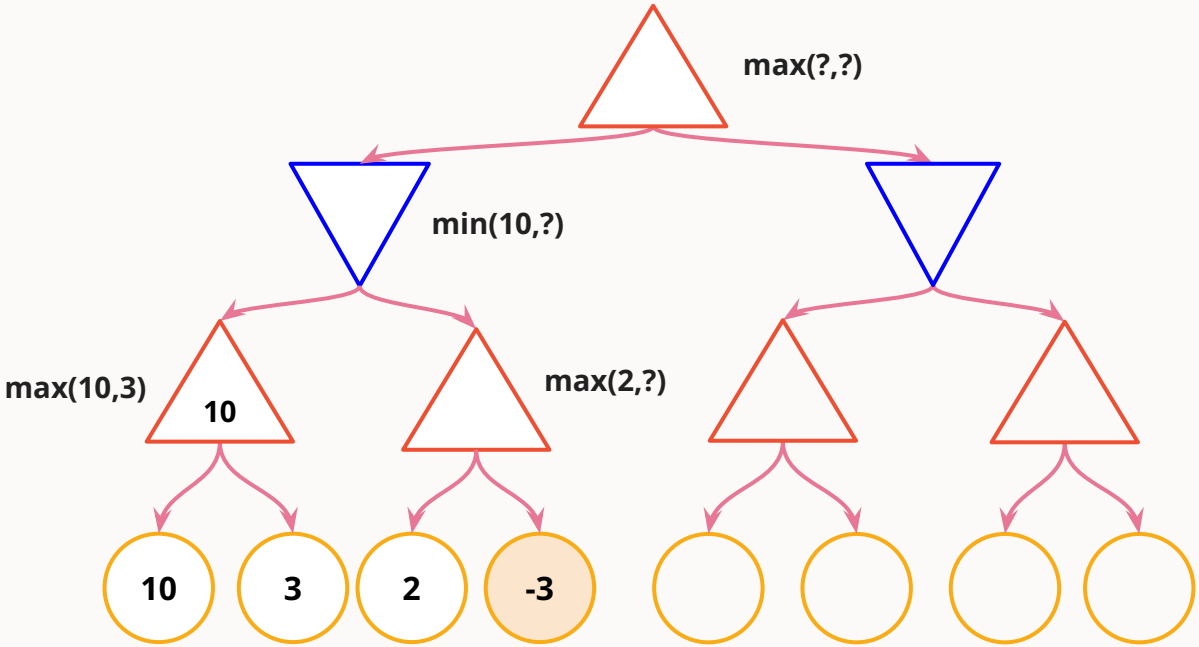


Jugador Max

Jugador Min

Estados Terminales

Cálculo del valor Minimax



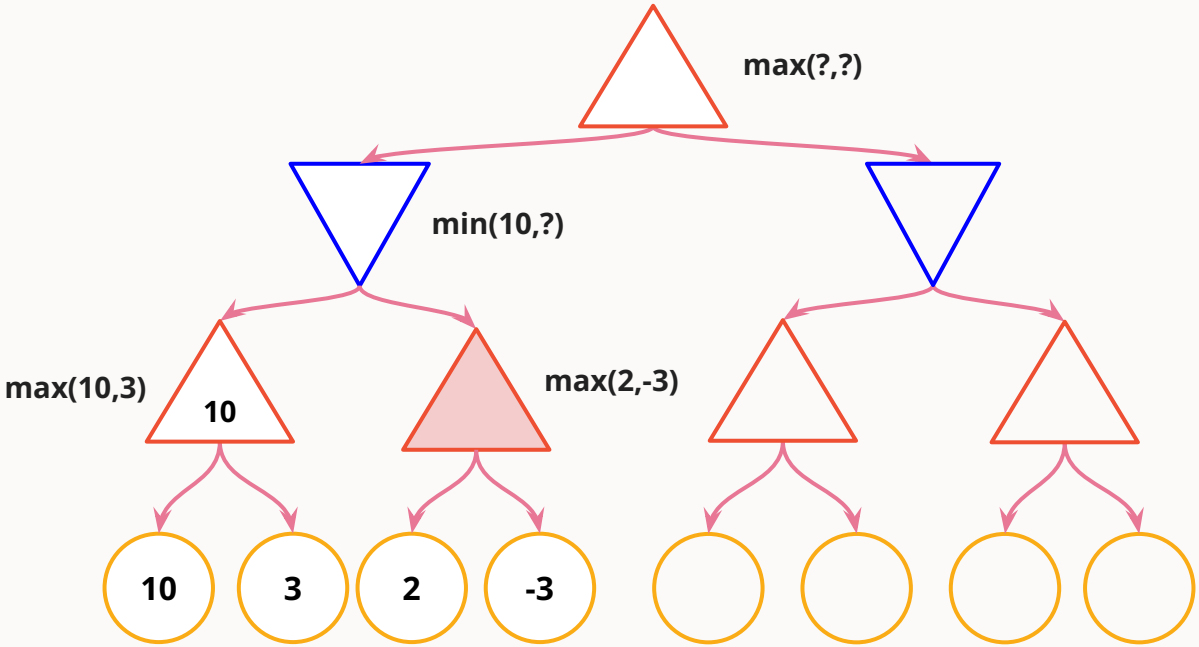
Jugador Max

Jugador Min

Estados Terminales

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



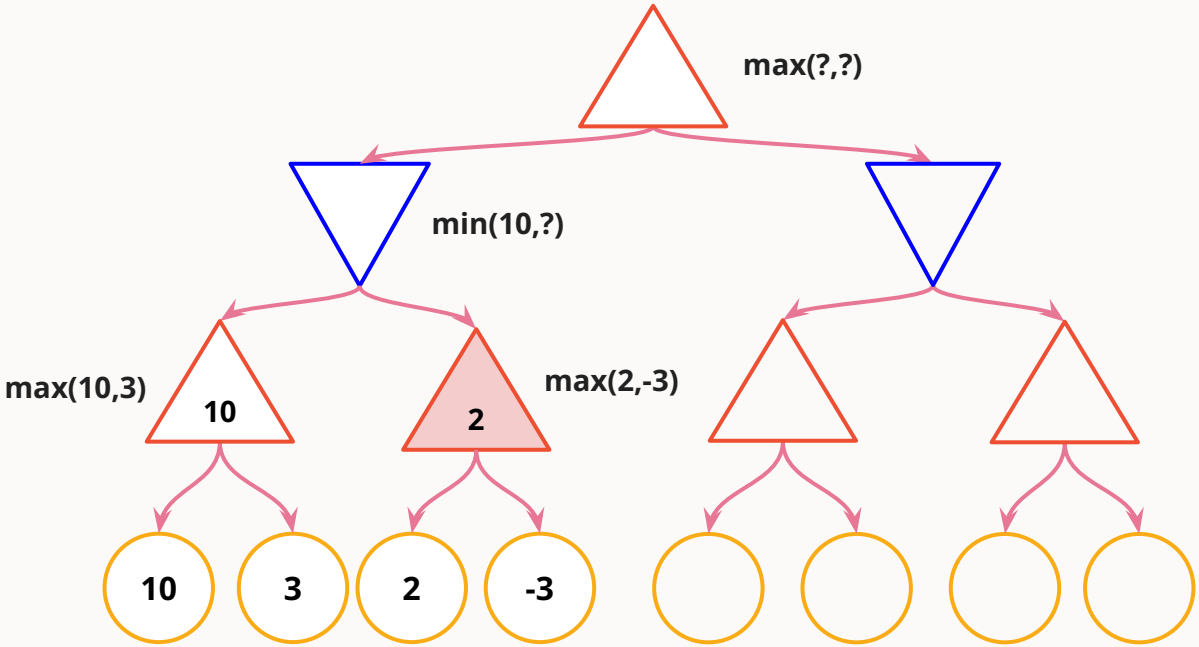
Jugador Max

Jugador Min

Estados Terminales

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



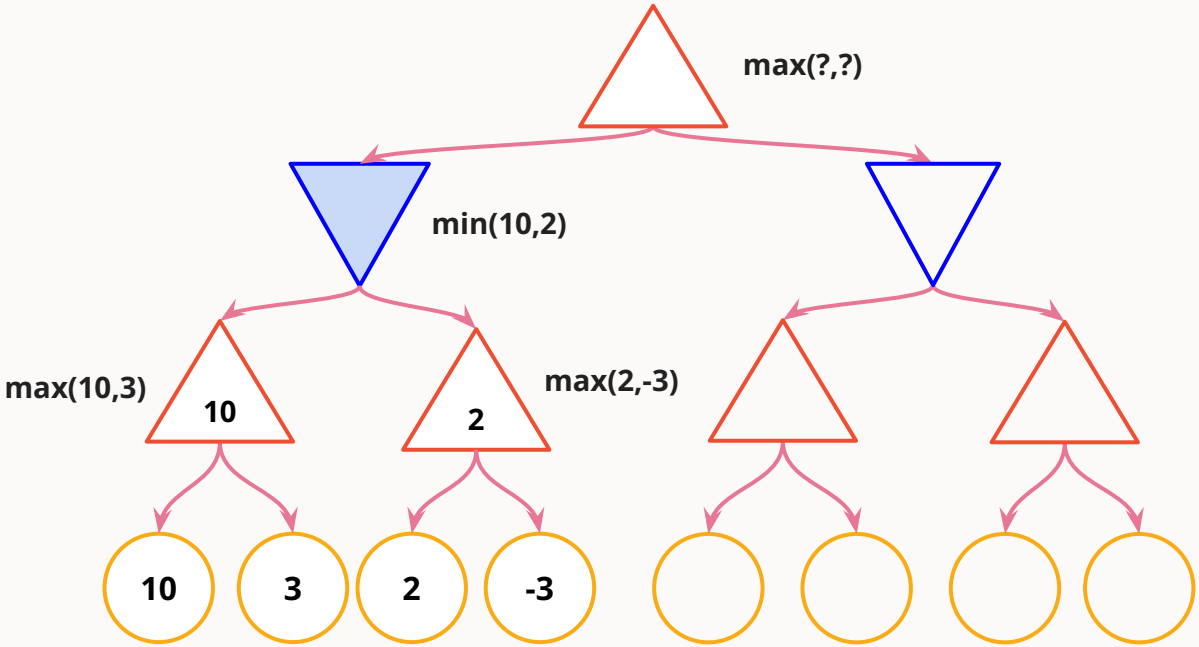
Jugador Max

Jugador Min

Estados Terminales

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



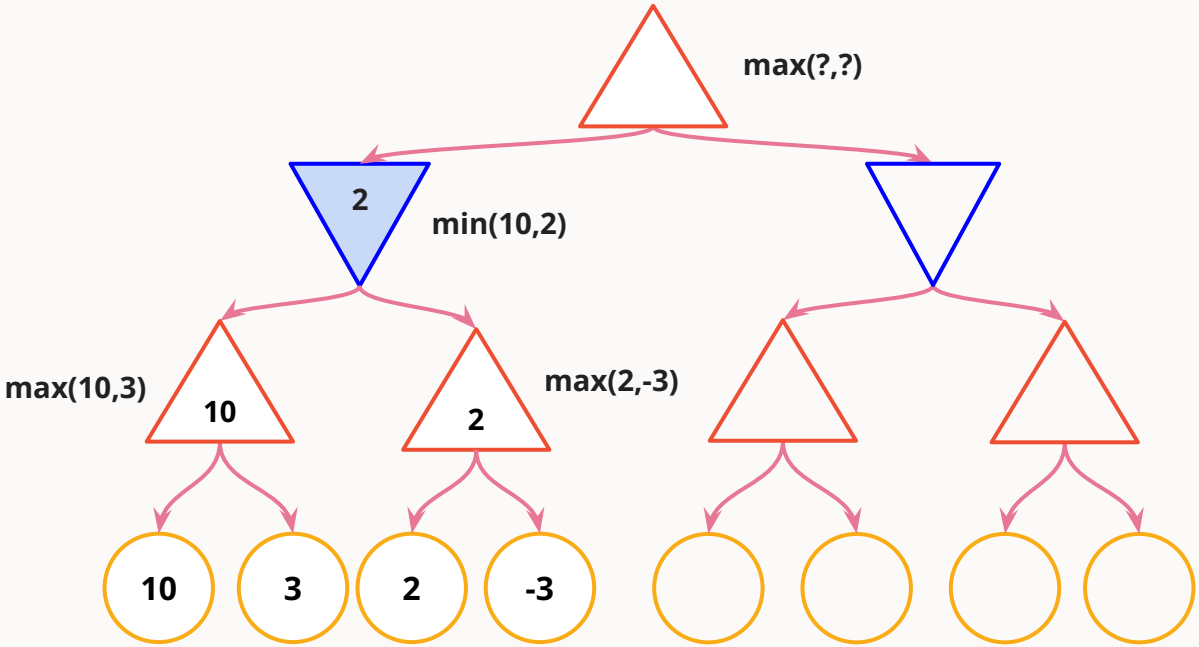
Jugador Max

Jugador Min

Estados Terminales

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



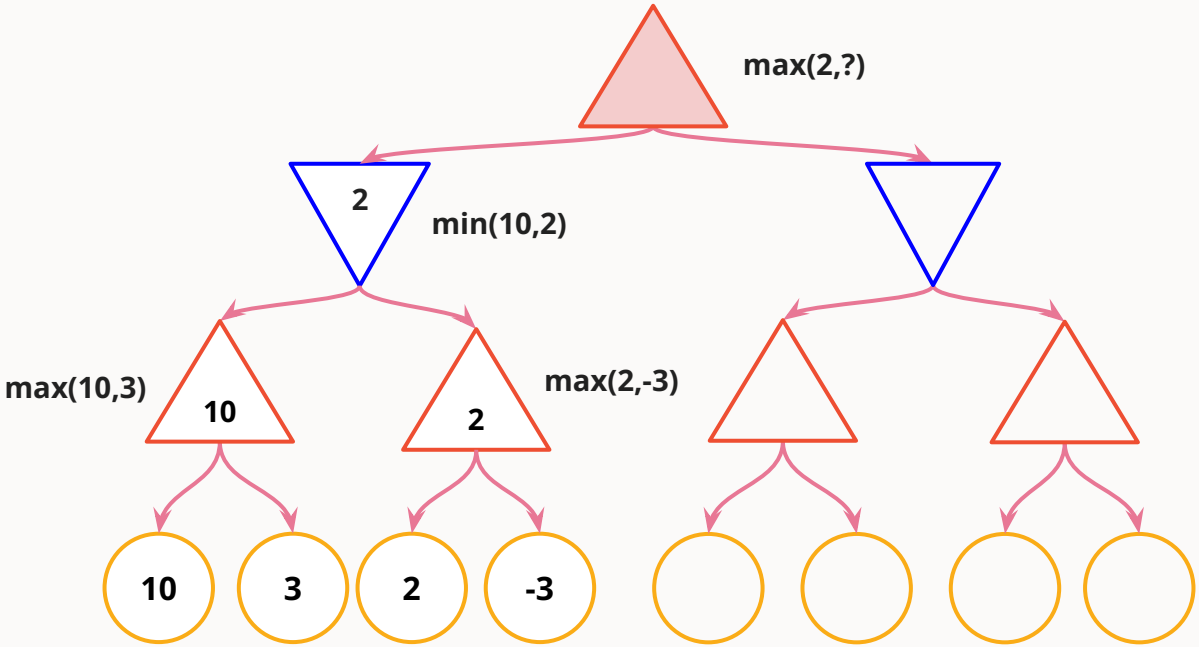
Jugador Max

Jugador Min

Estados Terminales

$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax

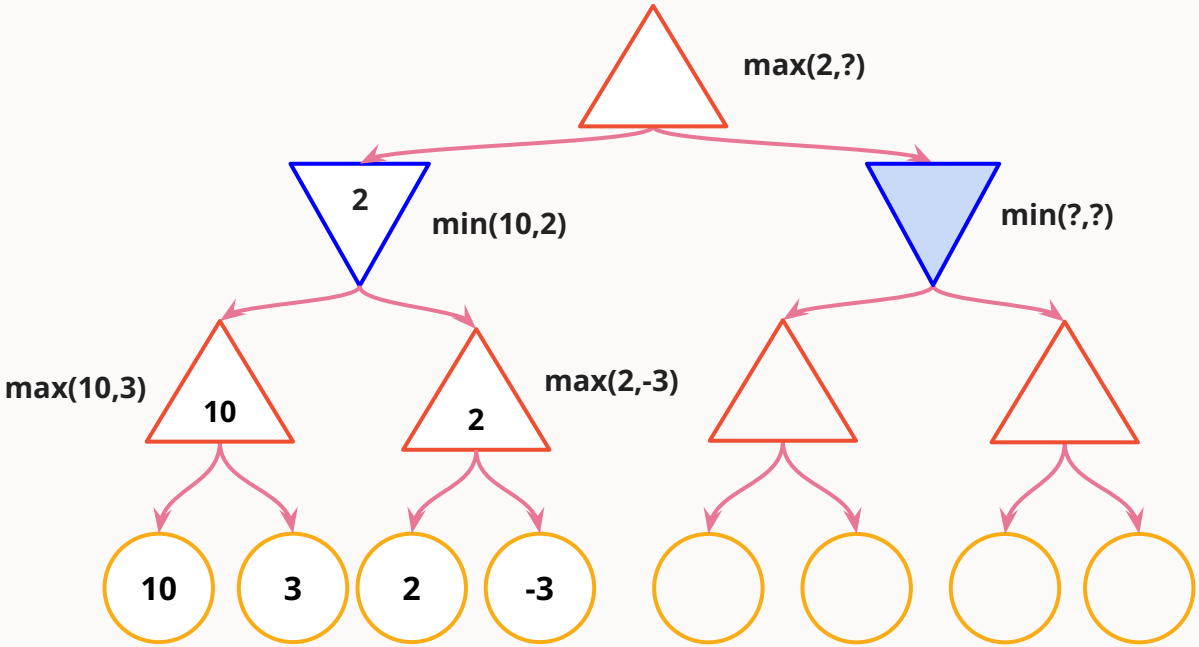


Jugador Max

Jugador Min

Estados Terminales

Cálculo del valor Minimax

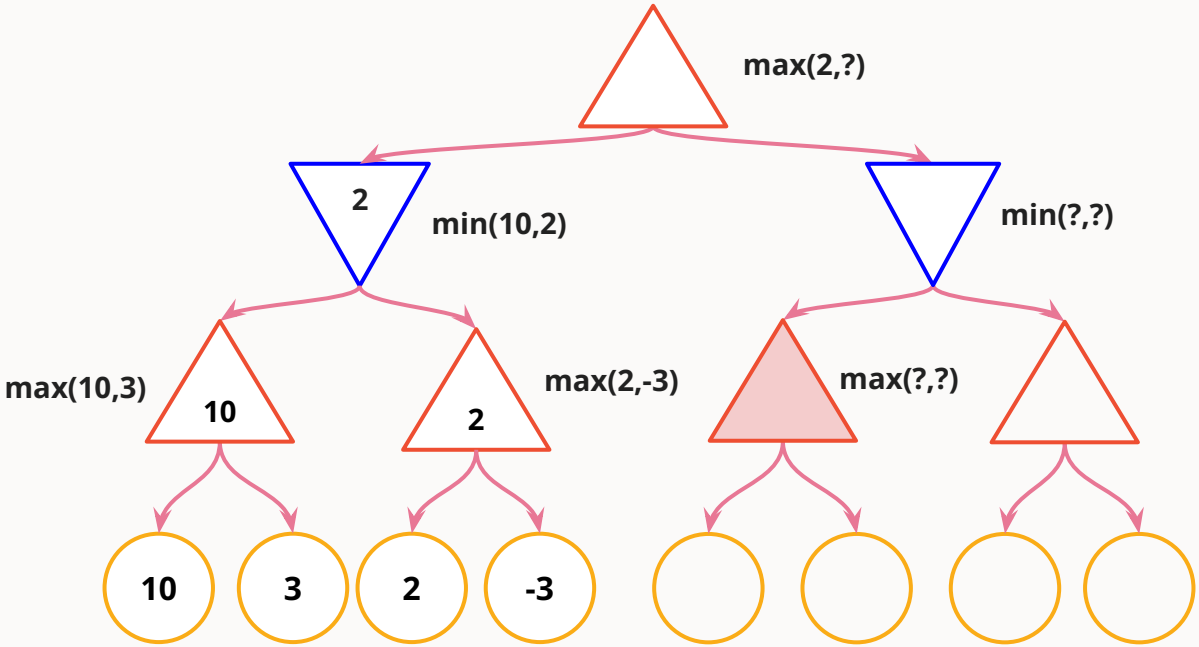


Jugador Max

Jugador Min

Estados Terminales

Cálculo del valor Minimax

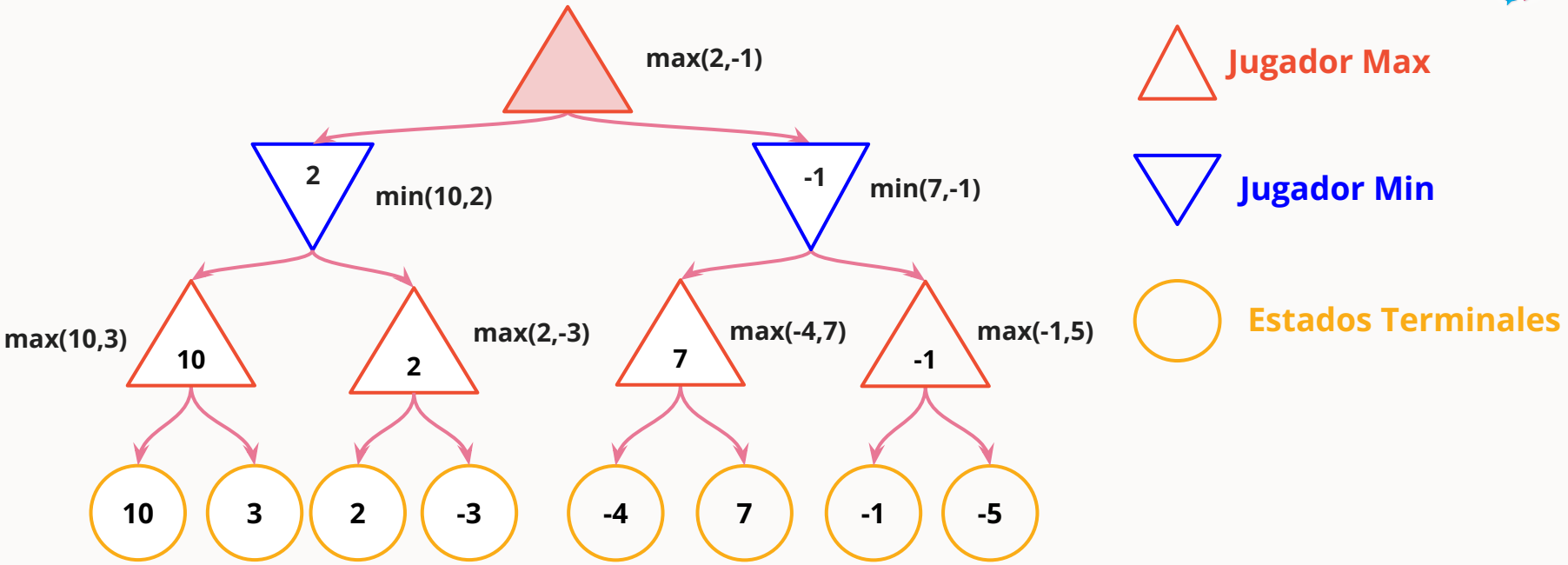


Jugador Max

Jugador Min

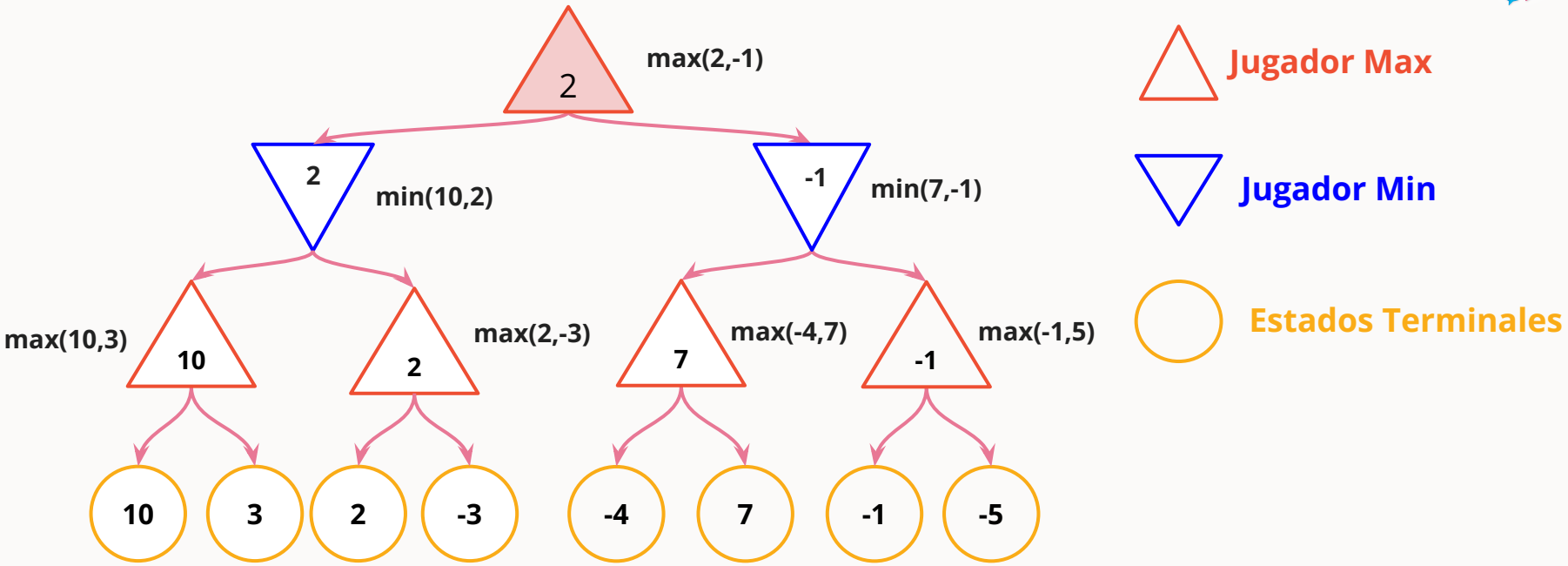
Estados Terminales

Cálculo del valor Minimax



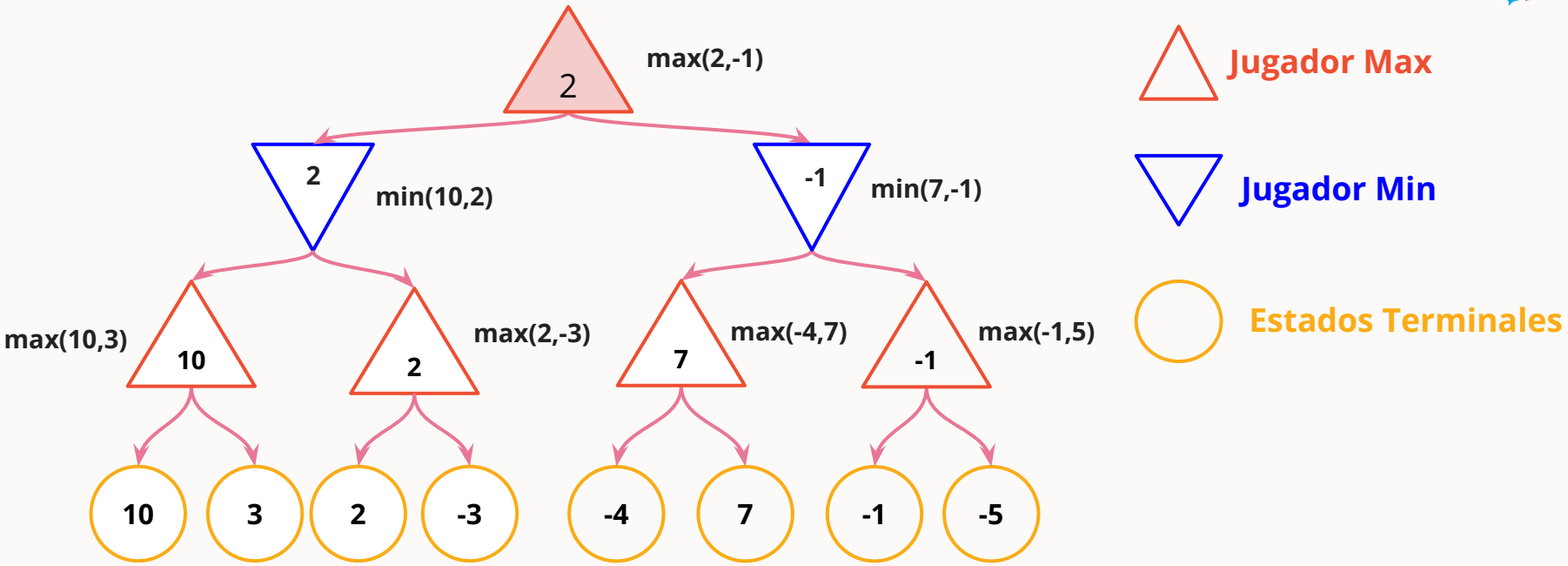
$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



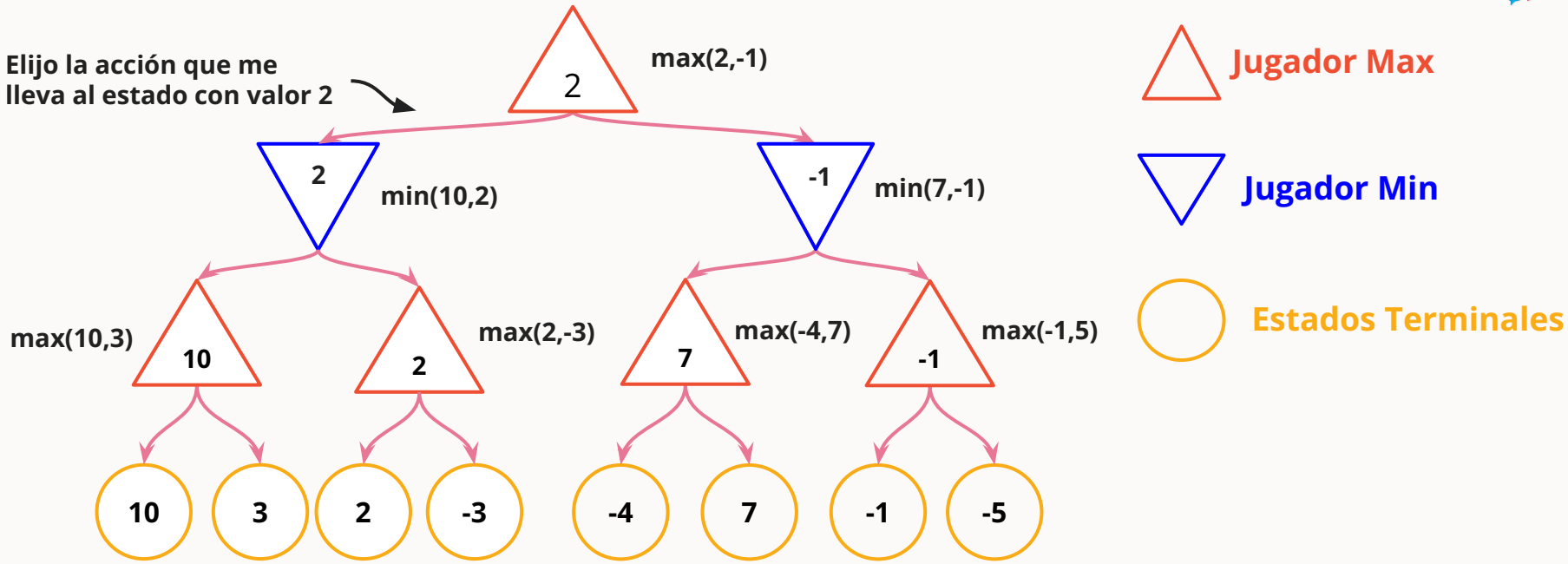
$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



$$\text{MINIMAX(s)} = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \text{máx}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \text{mín}_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

Cálculo del valor Minimax



Rendimiento Minimax

La búsqueda Minimax corre DFS **sobre el árbol de estados completo.** Esto genera muchísimos estados para juegos complejos (ej. Ajedrez)

Para mejorar el rendimiento podemos:

- **Acotar** la profundidad del árbol de búsqueda
- **Podar** ramas del árbol de búsqueda

Poda Alpha-Beta

Podemos podar buena parte del árbol si guardamos dos parámetros adicionales en cada llamada a la búsqueda Minimax:

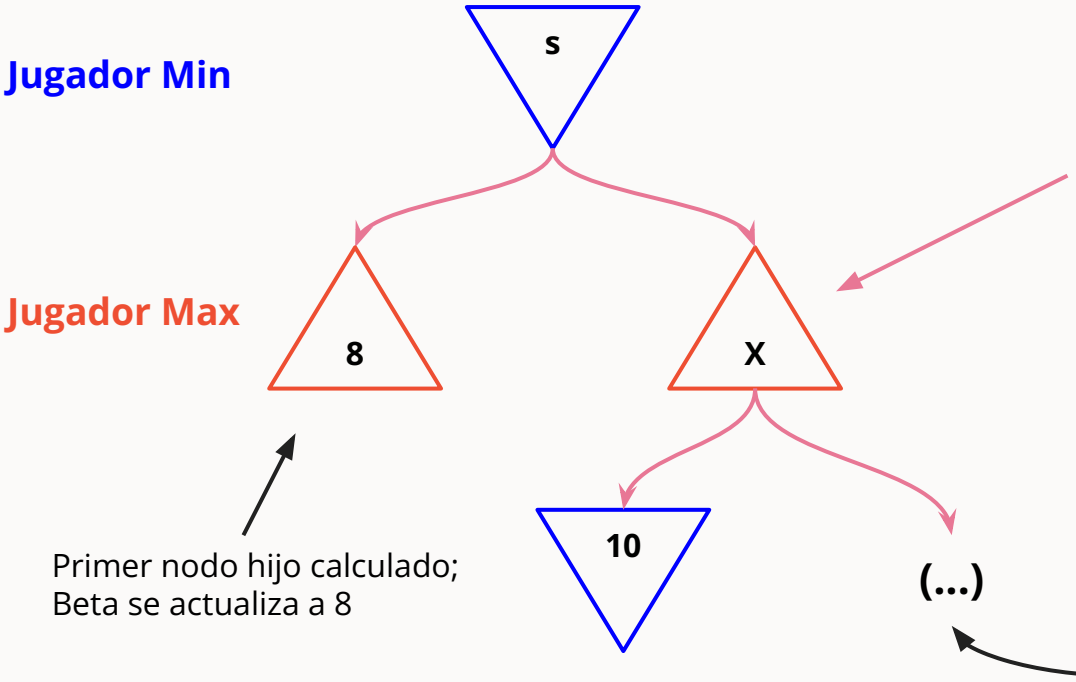
- Alpha: cota **inferior** del valor de un nodo
- Beta: cota **superior** del valor de un nodo

Tener estas cotas nos indica si es necesario calcular un nodo.
Esta decisión **no afecta la optimalidad de la solución.**

Poda Alpha-Beta

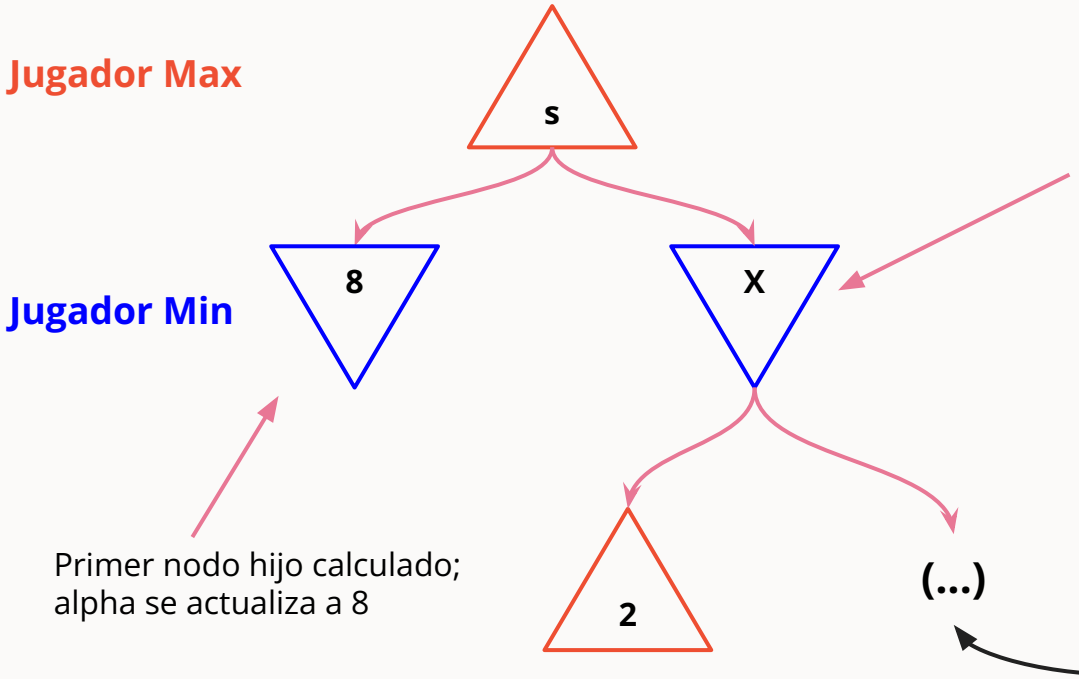
```
function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Poda Alpha-Beta - Cota superior Beta



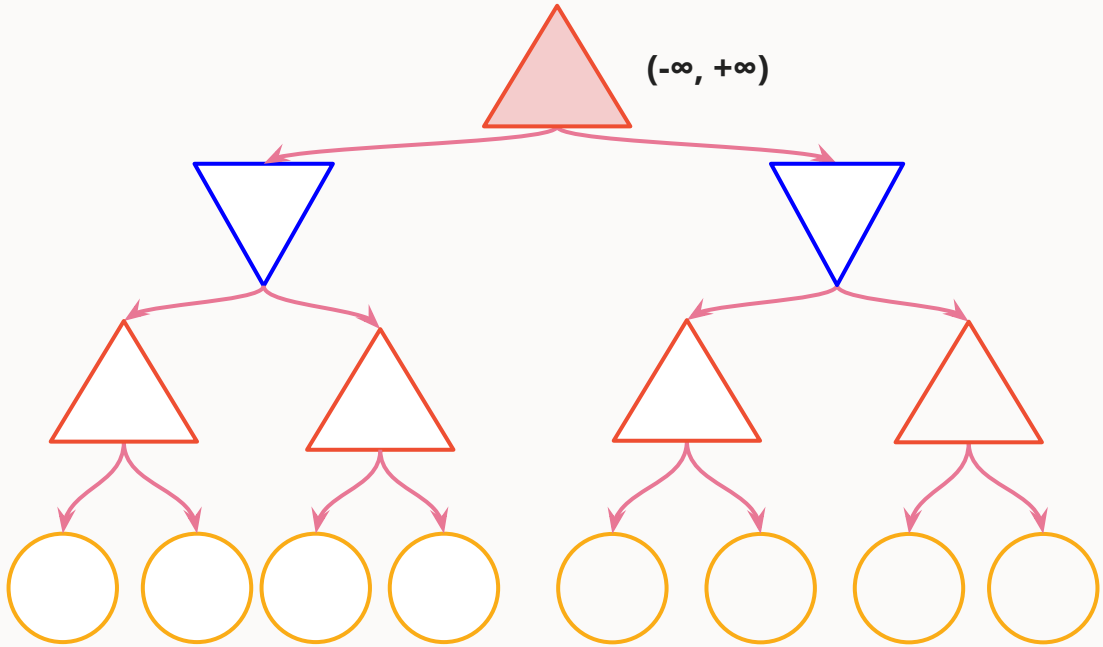
Sabemos que x ≥ 10 apenas verificamos el valor del primer nodo hijo. Como nuestro beta es igual a 8, sabemos que el nodo *s* **no** optará por el valor *x*. Luego podemos omitir el cálculo de los nodos hijos aún no calculados y llegar a la misma elección (podamos el sub-árbol)

Poda Alpha-Beta - Cota inferior Alpha



Sabemos que x ≤ 2 apenas verificamos el valor del primer nodo hijo. Como nuestro alpha es igual a 8, sabemos que el nodo *s* **no** optará por el valor *x*. Luego podemos omitir el cálculo de los nodos hijos aún no calculados y llegar a la misma elección (podamos el sub-árbol)

Poda Alpha-Beta



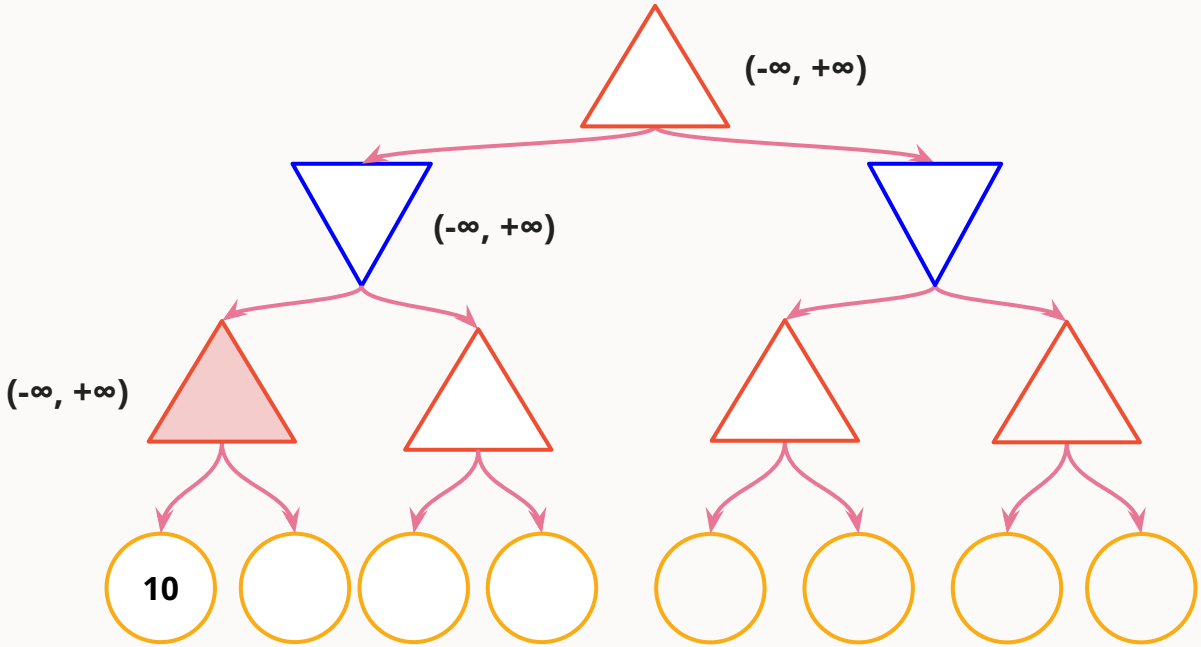
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



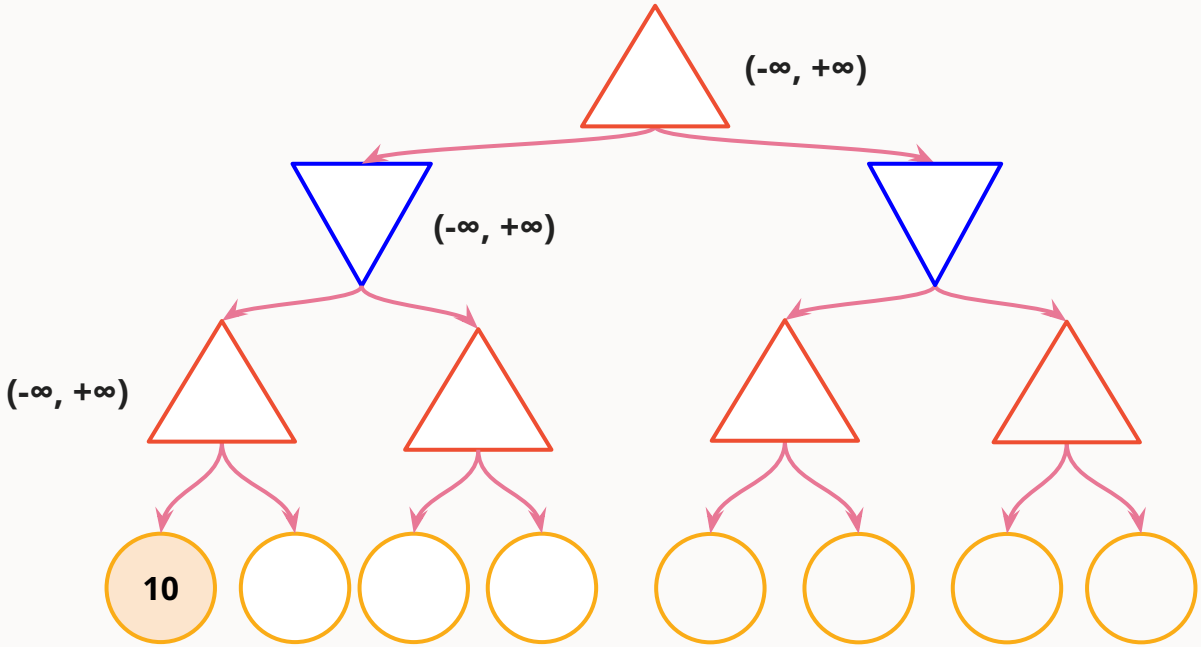
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



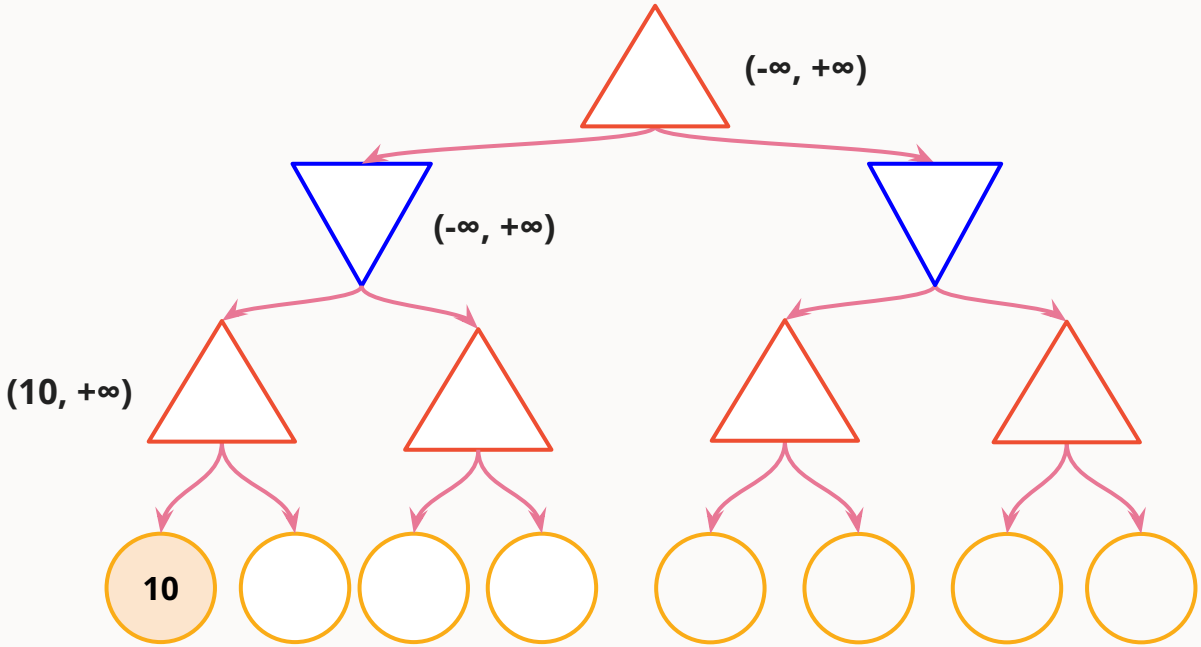
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



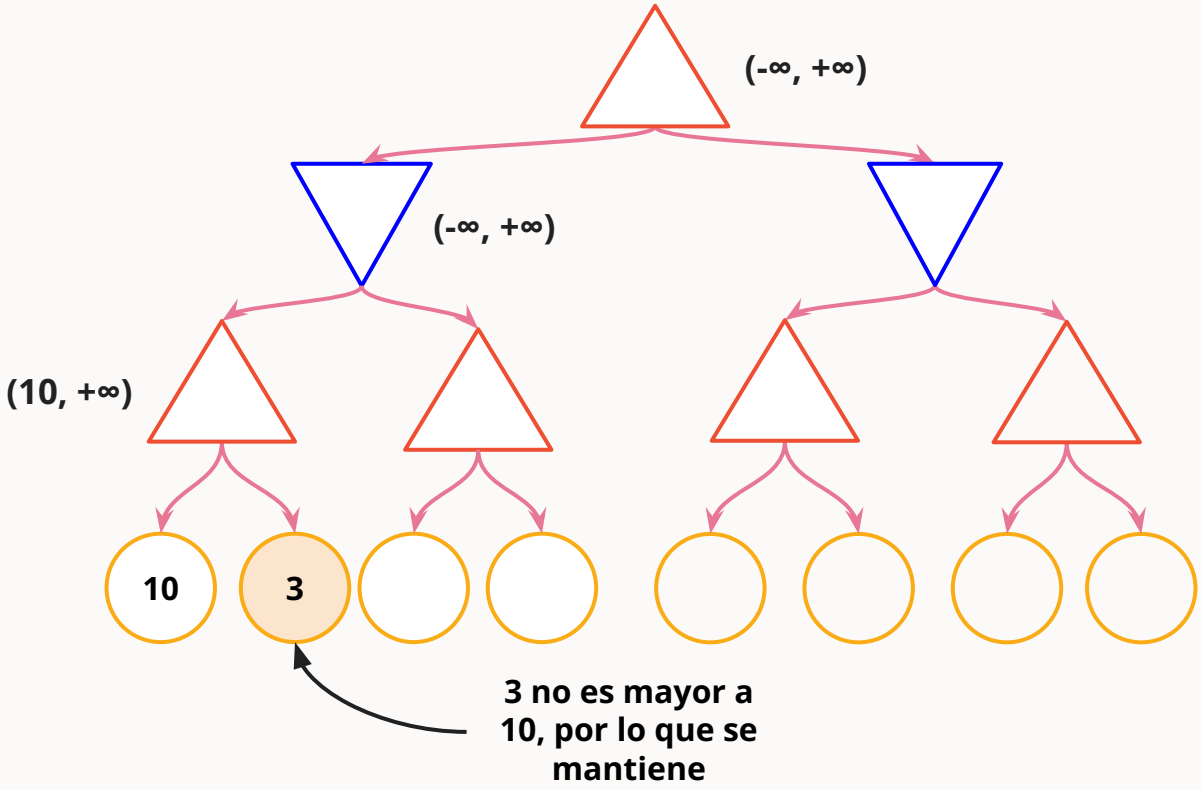
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



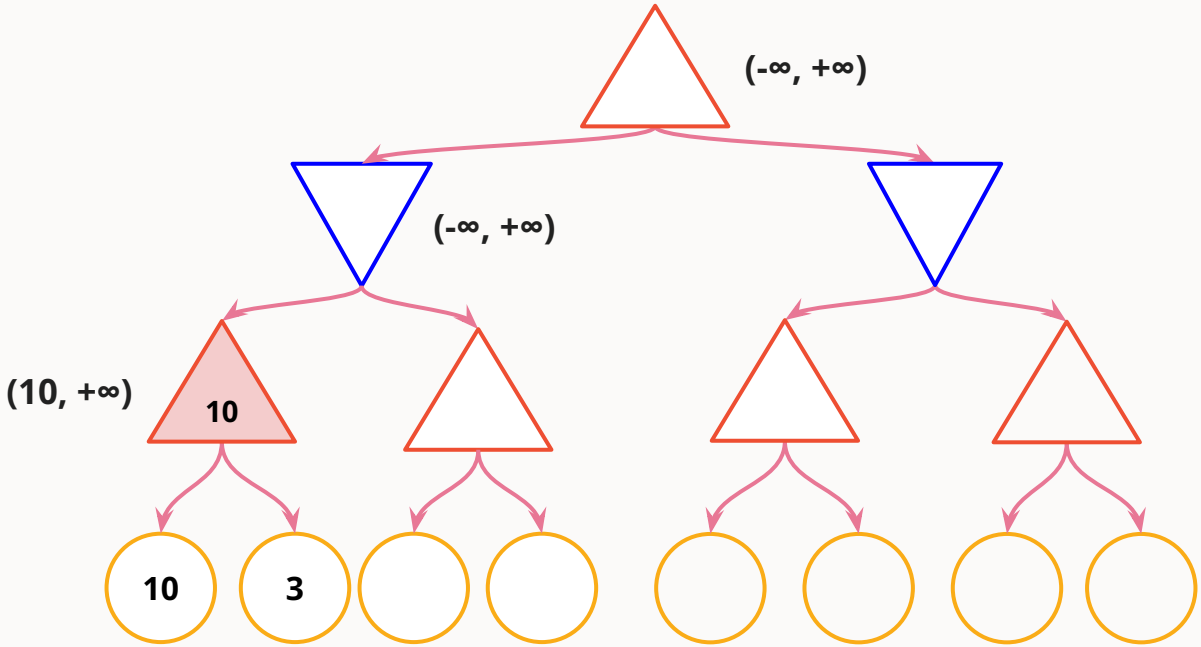
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



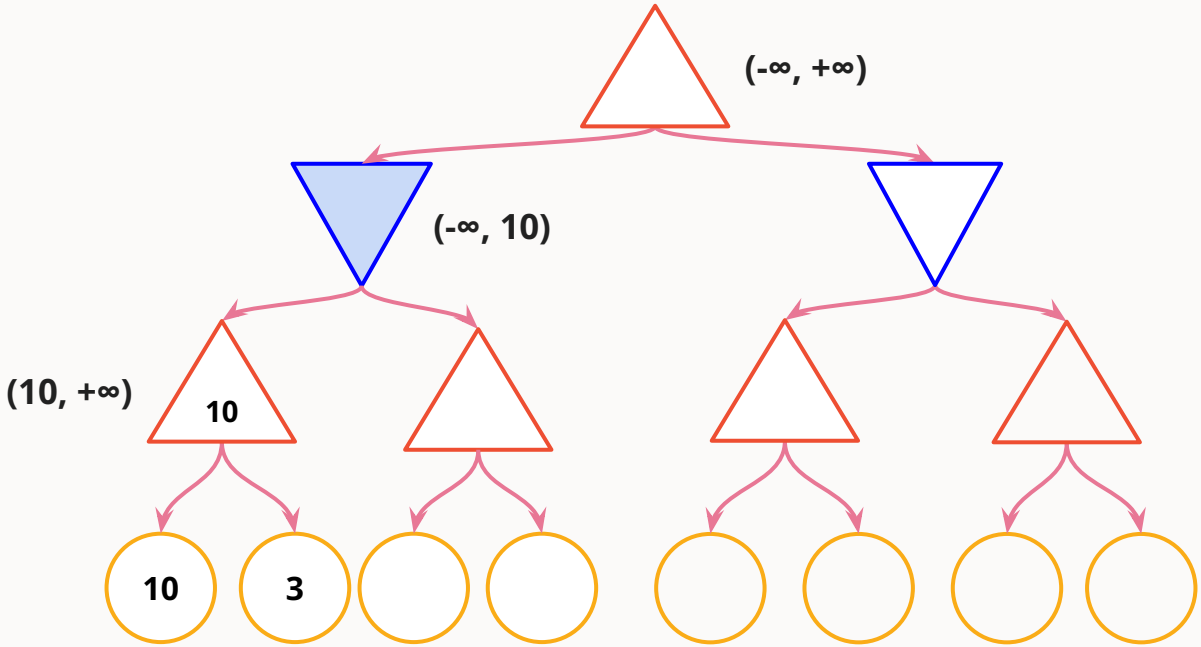
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



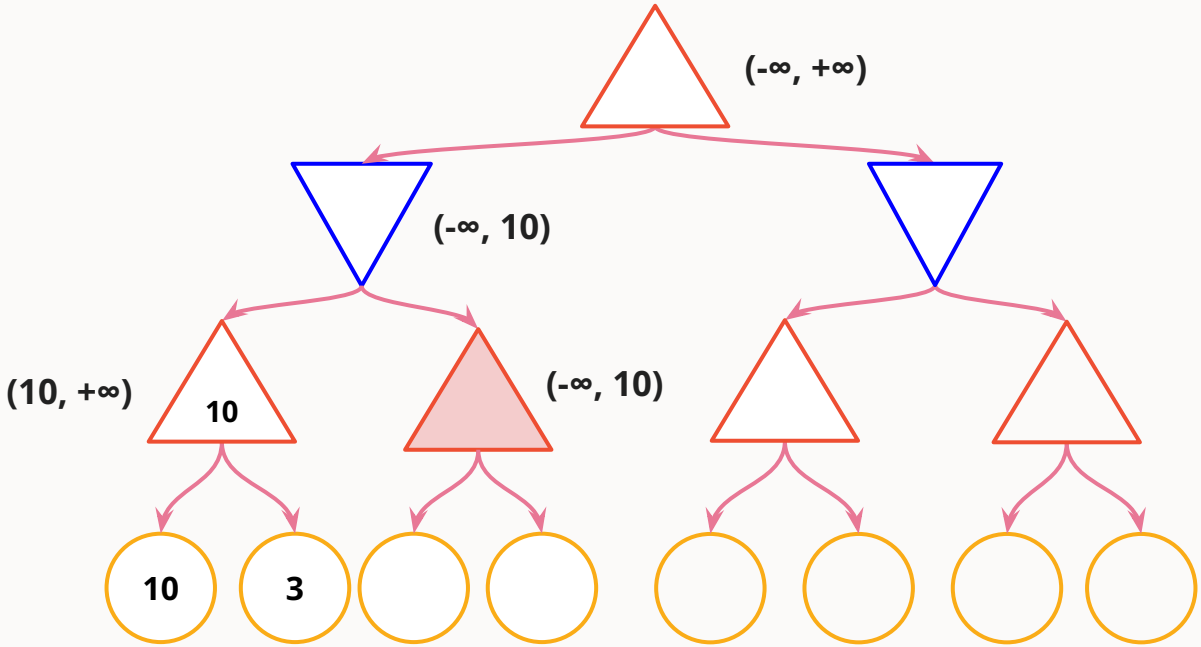
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



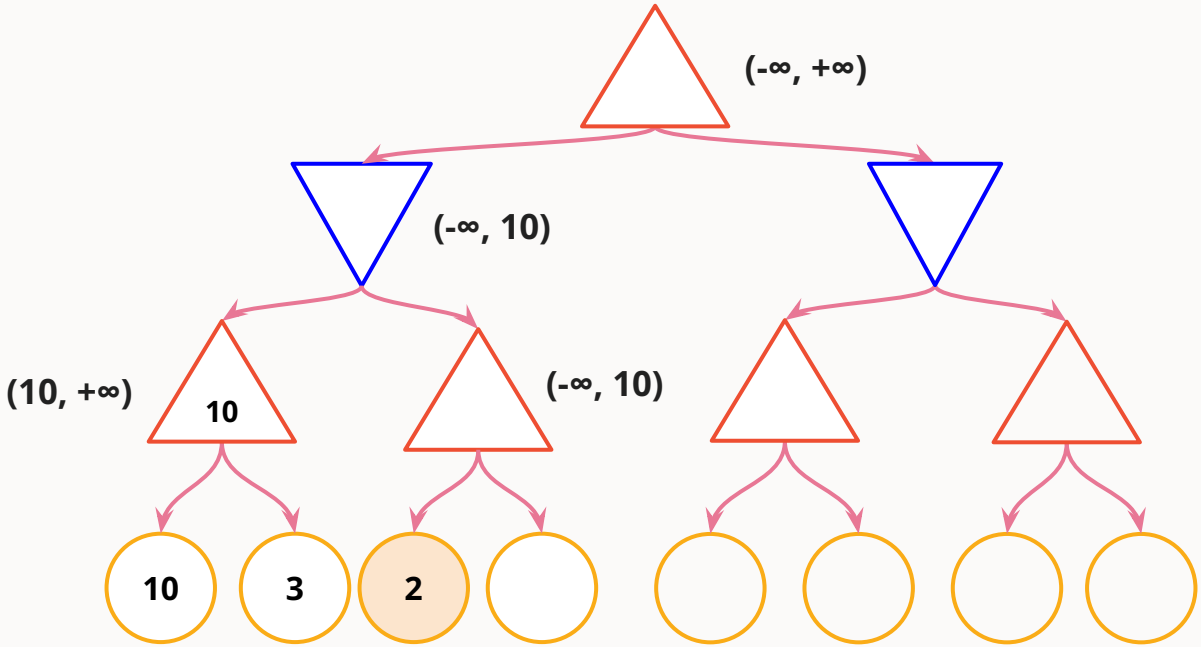
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



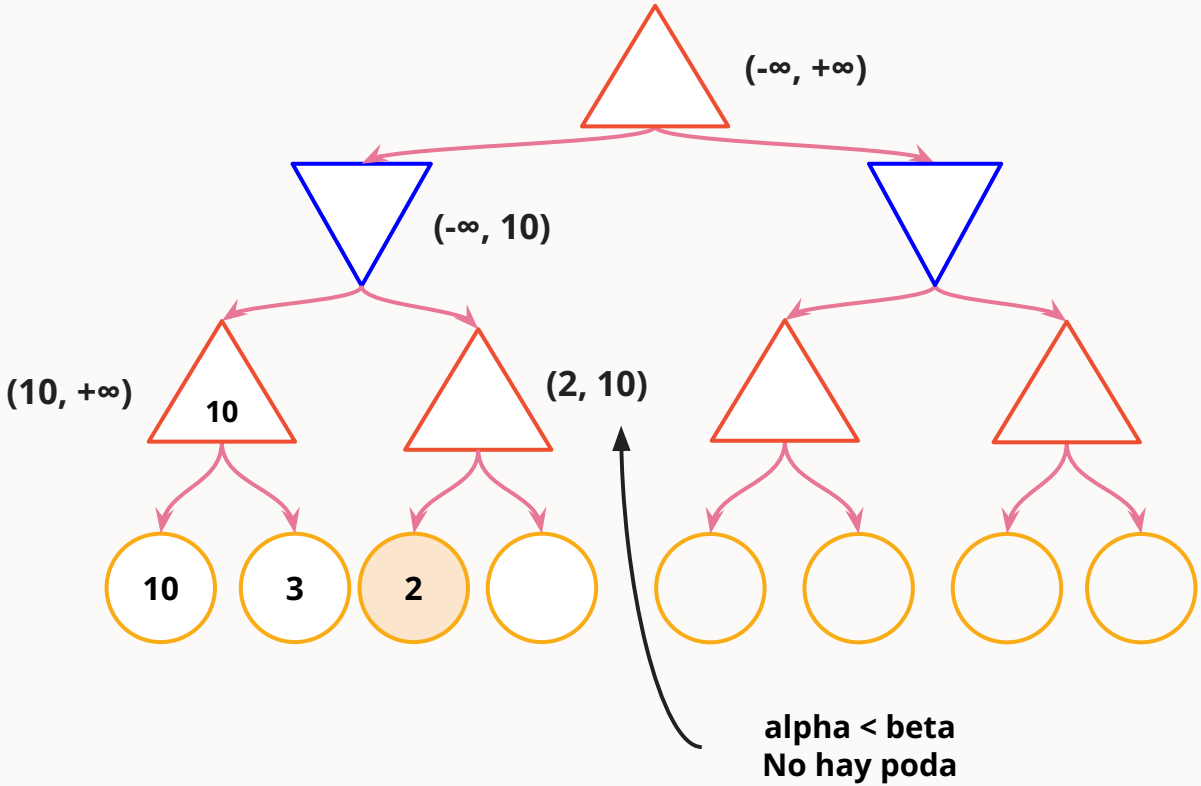
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



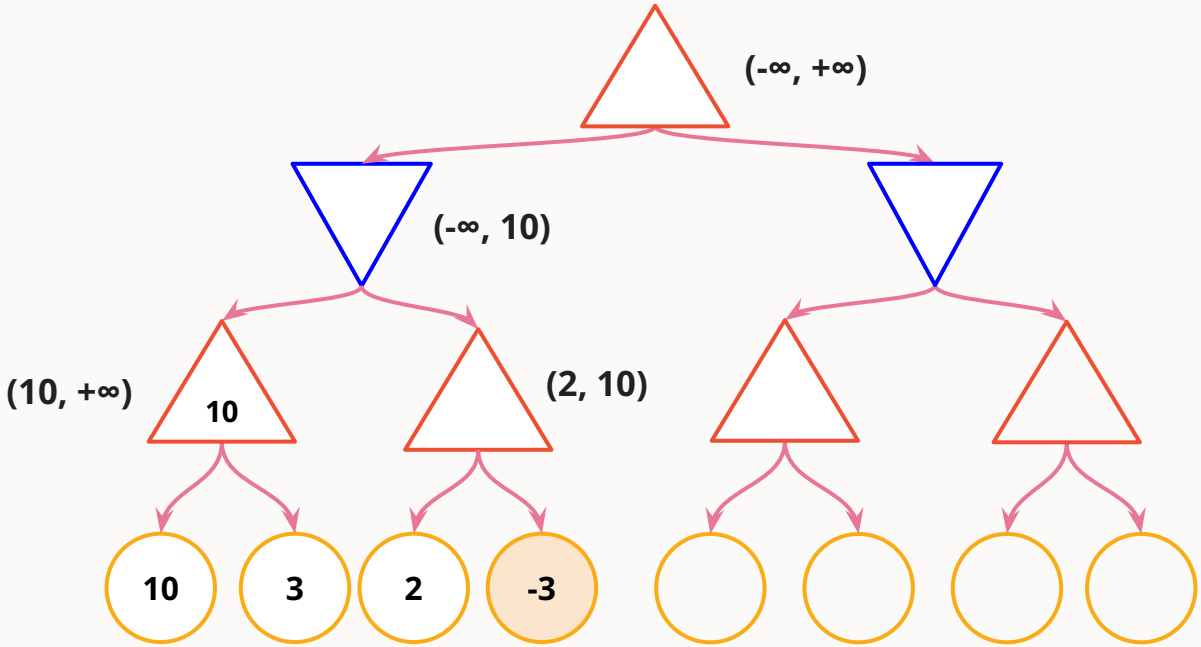
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



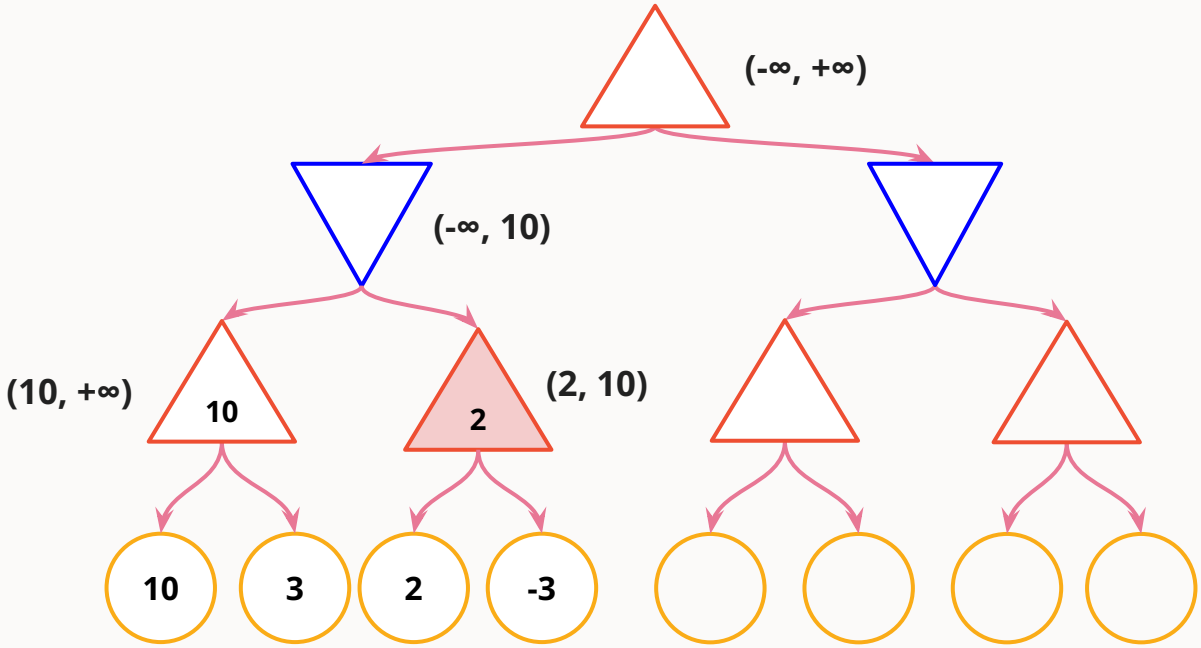
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



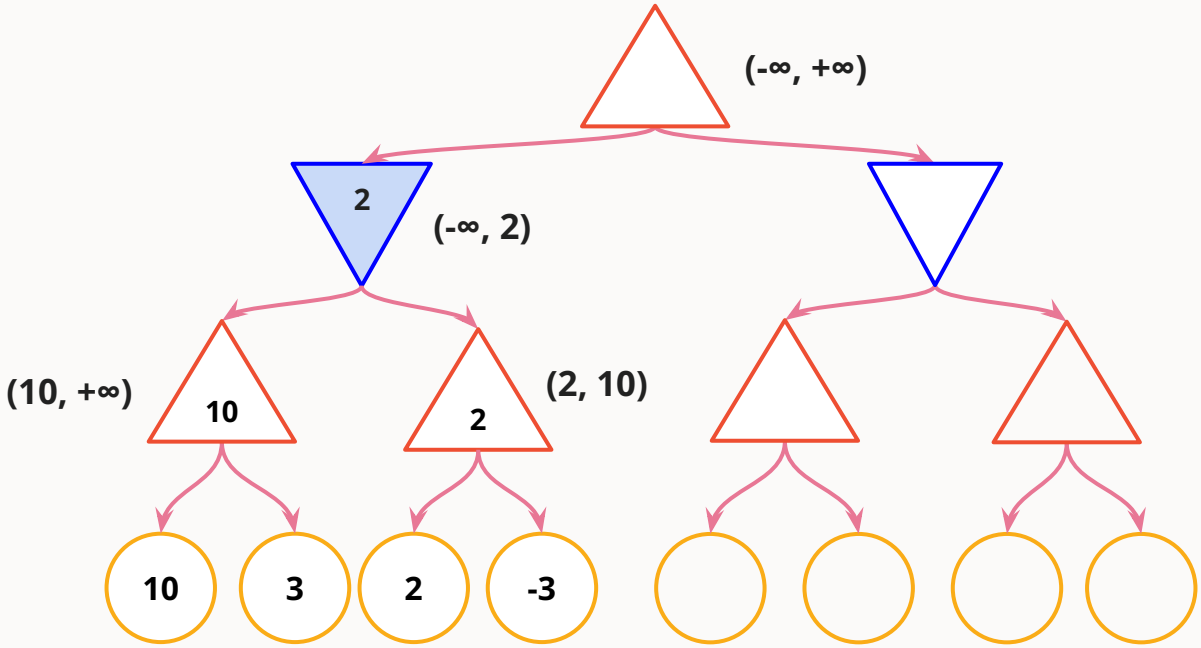
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



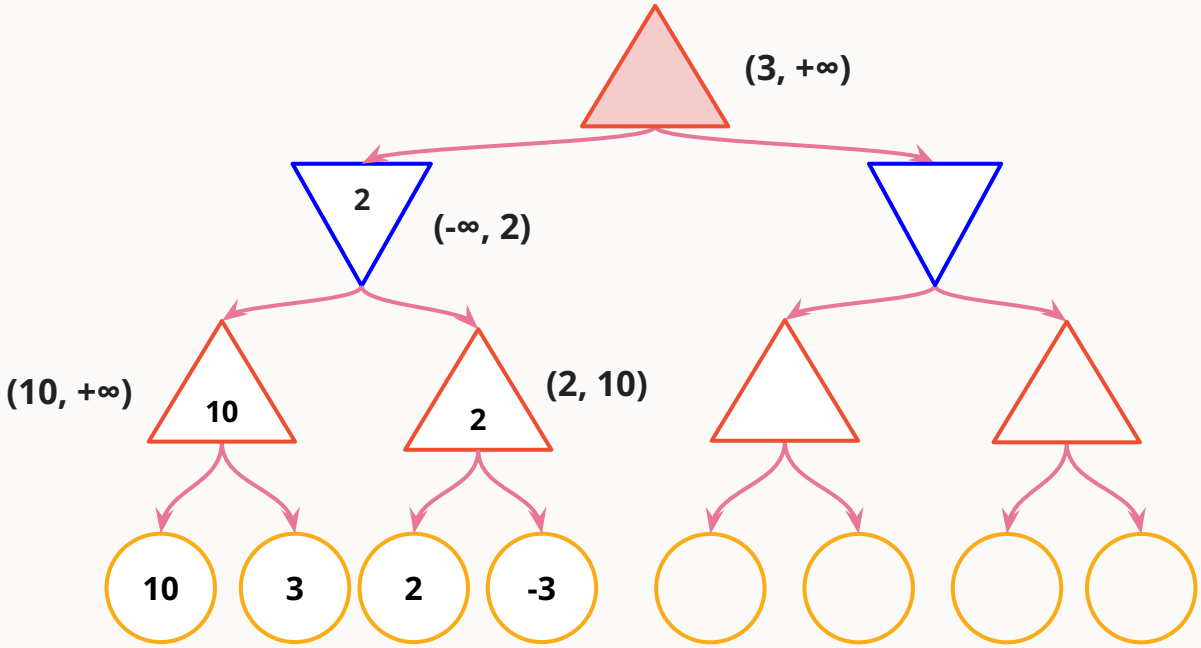
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



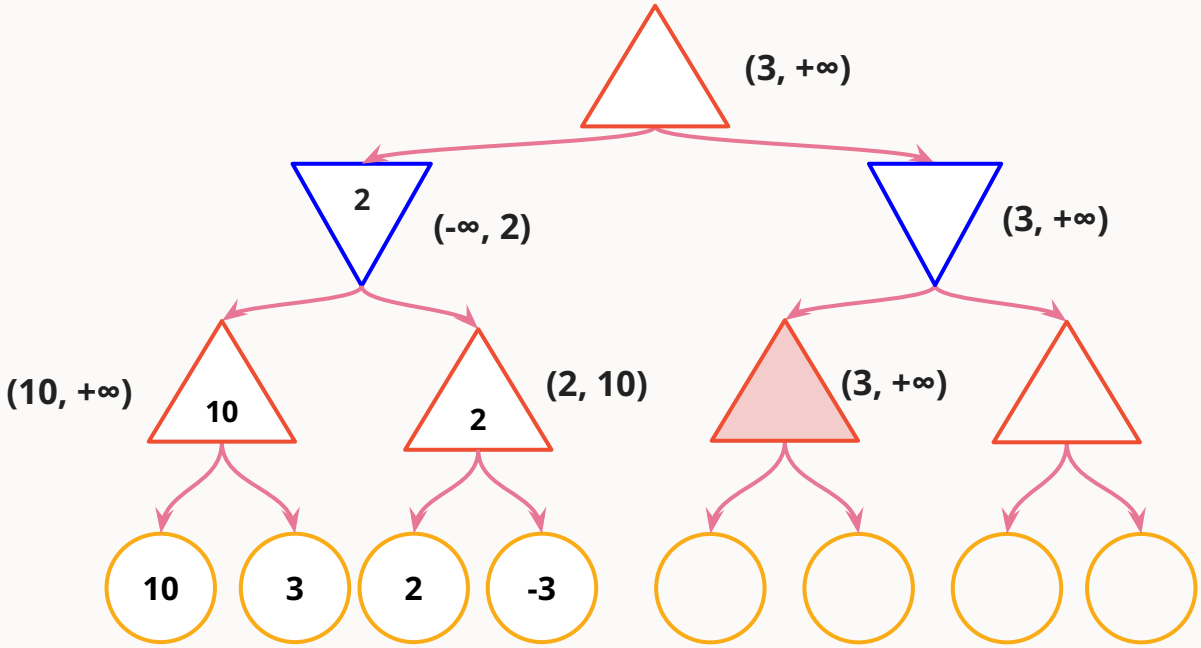
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



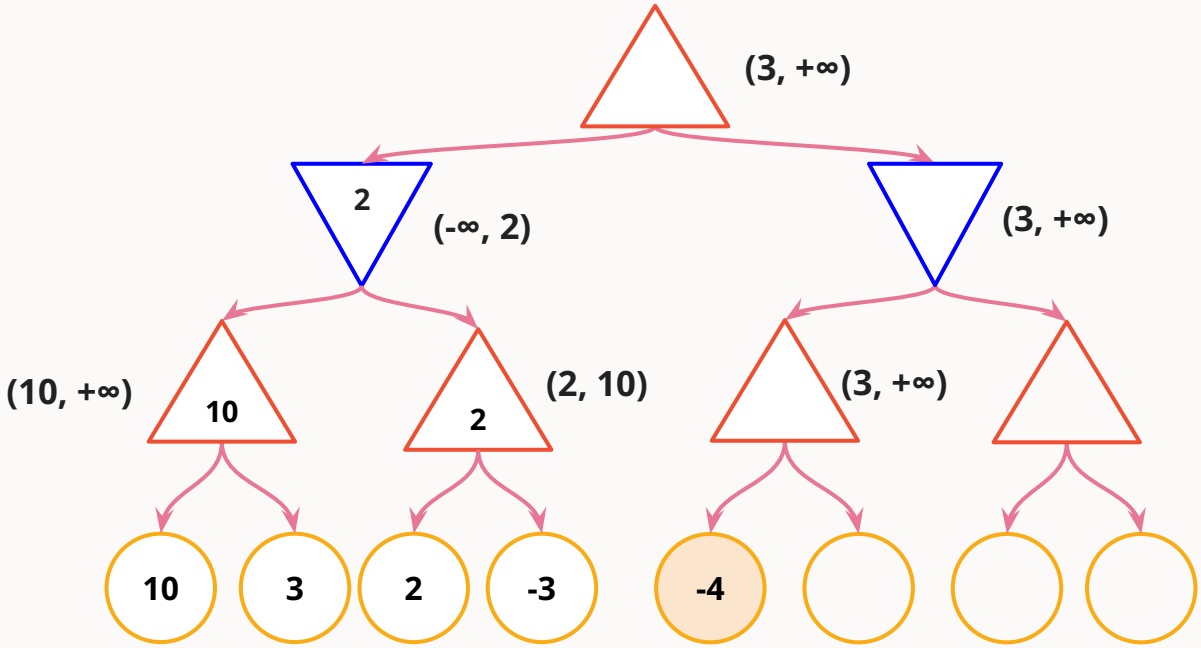
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



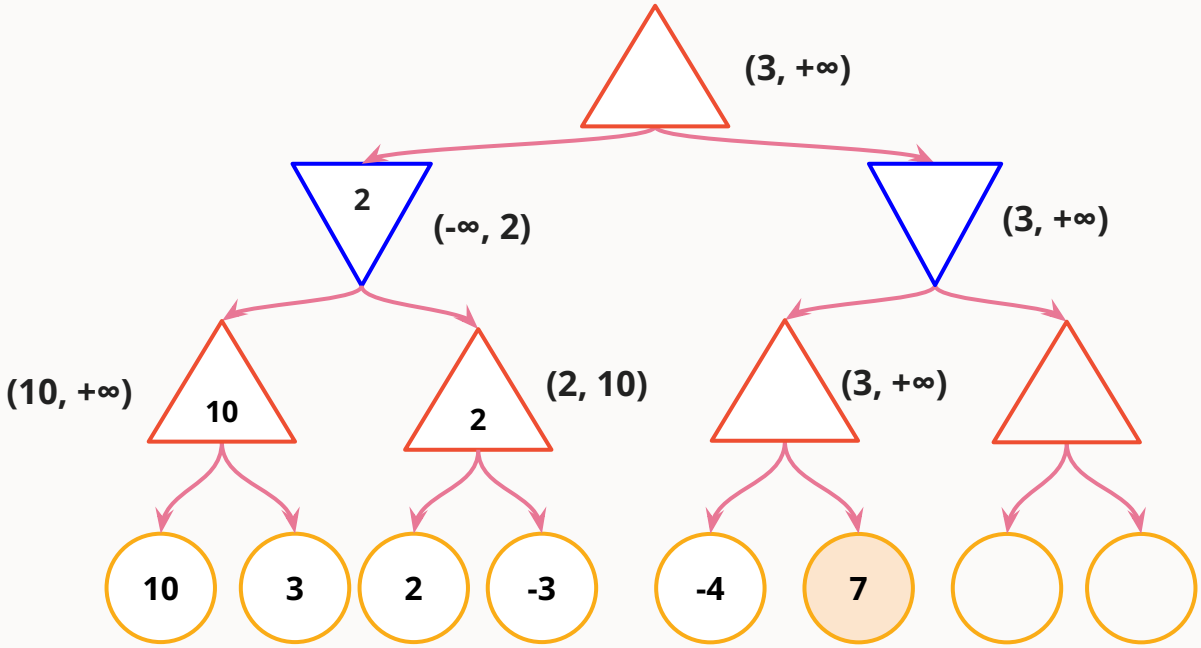
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



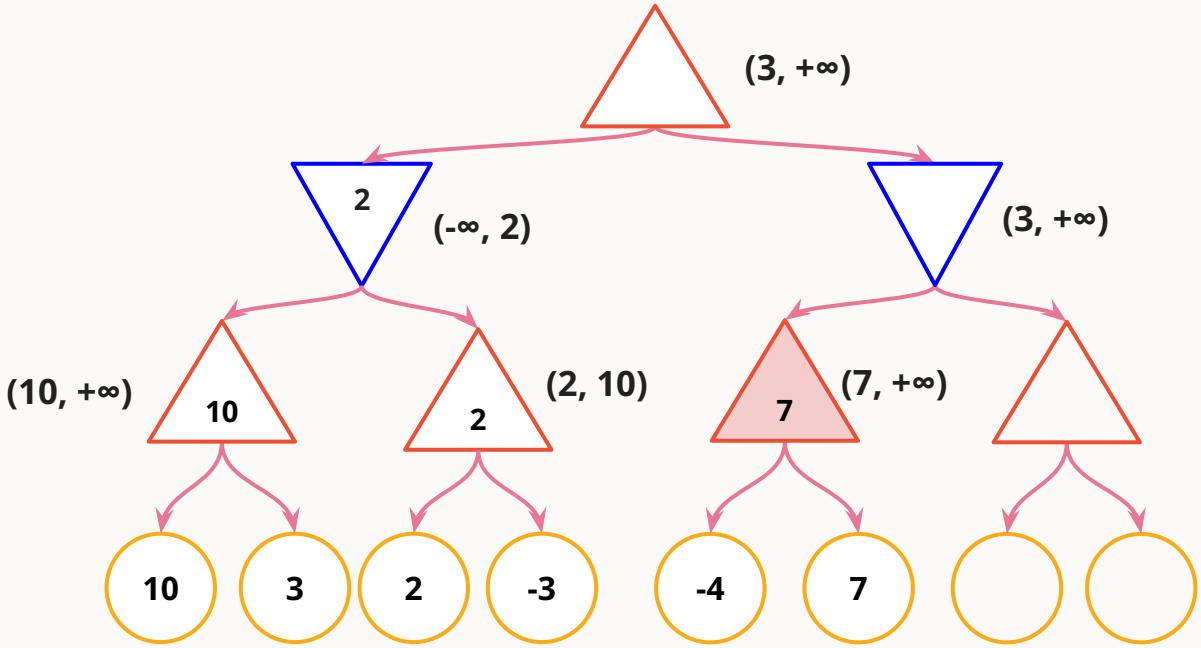
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



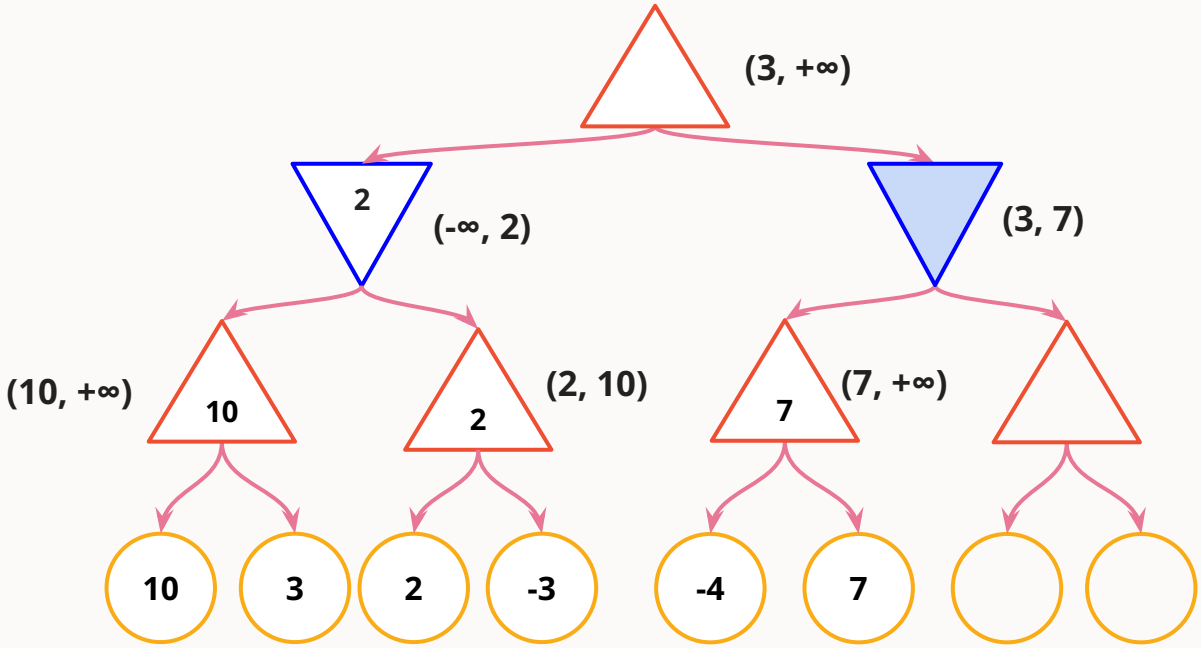
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



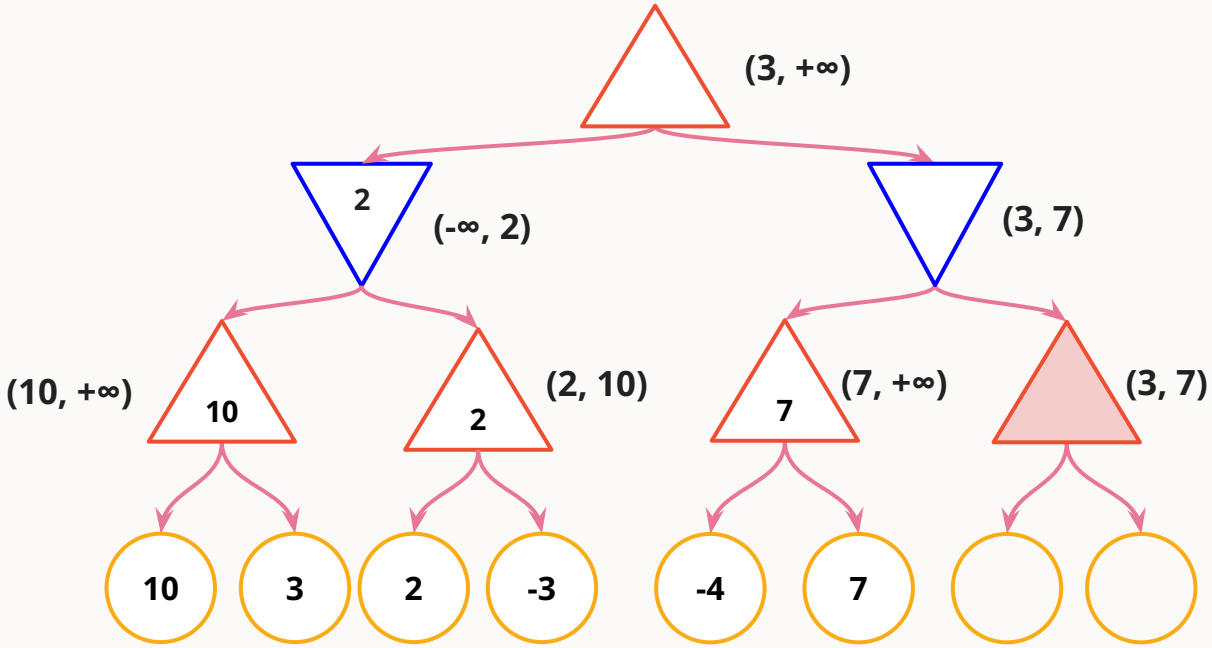
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



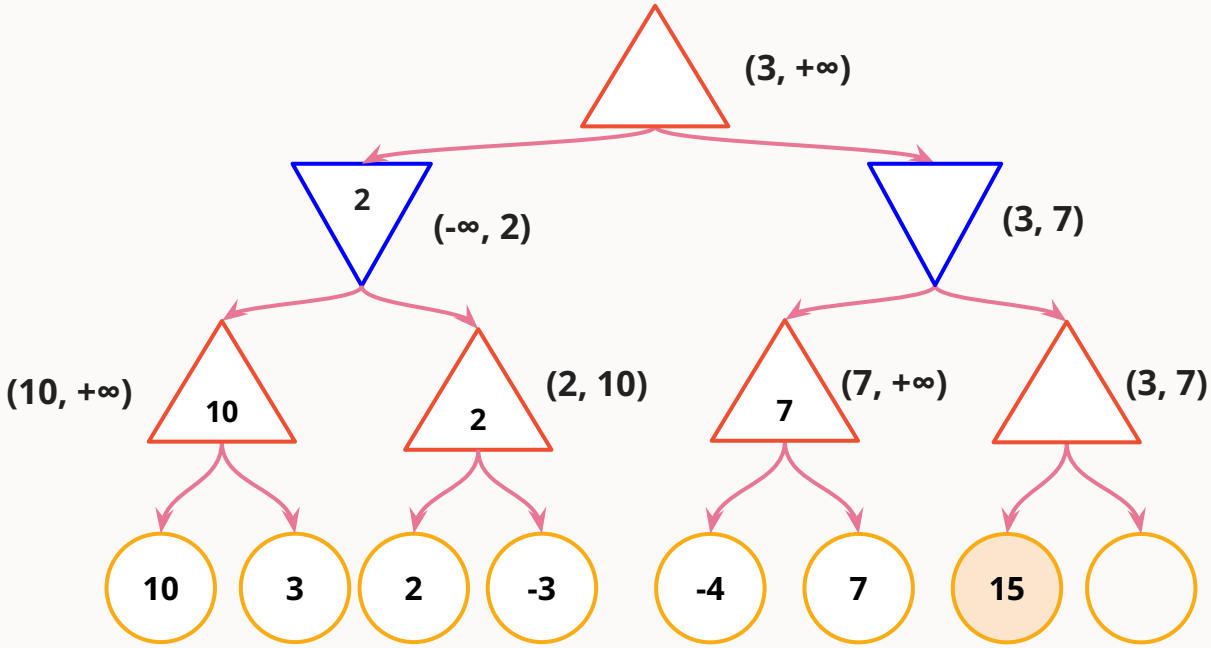
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



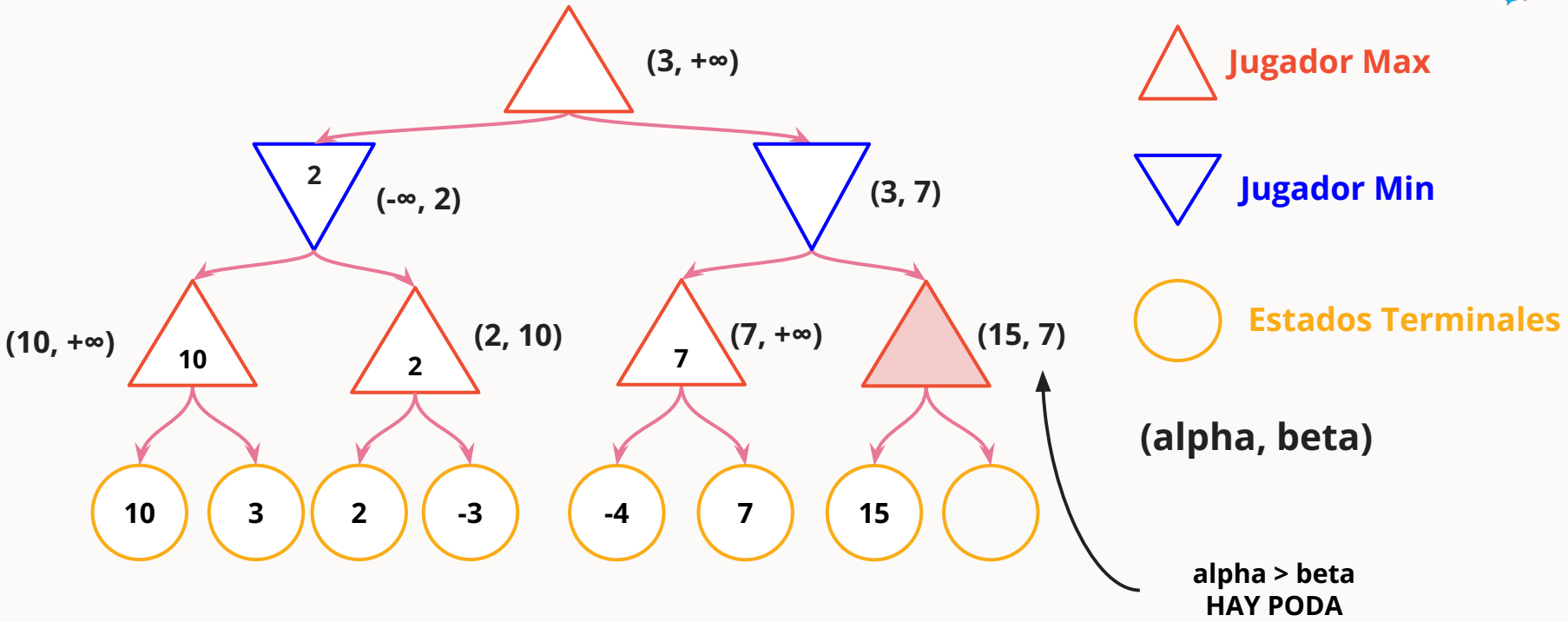
Jugador Max

Jugador Min

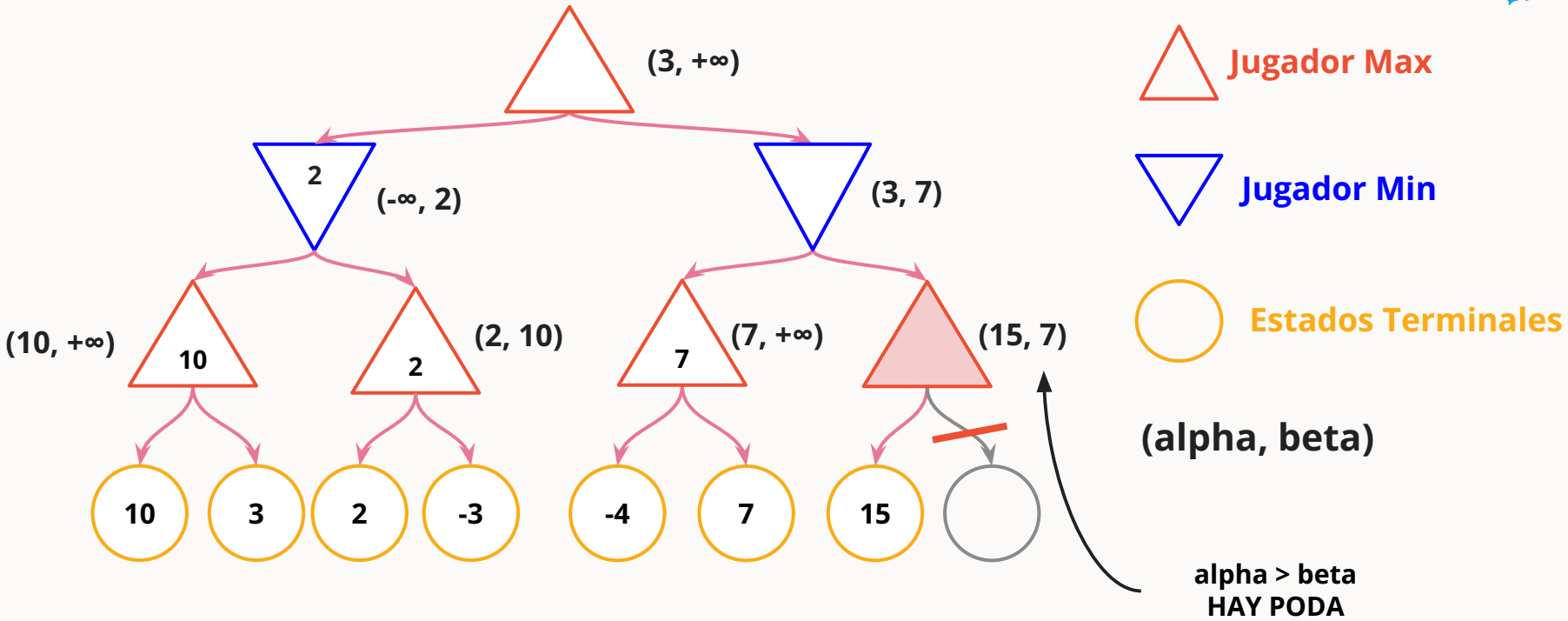
Estados Terminales

(alpha, beta)

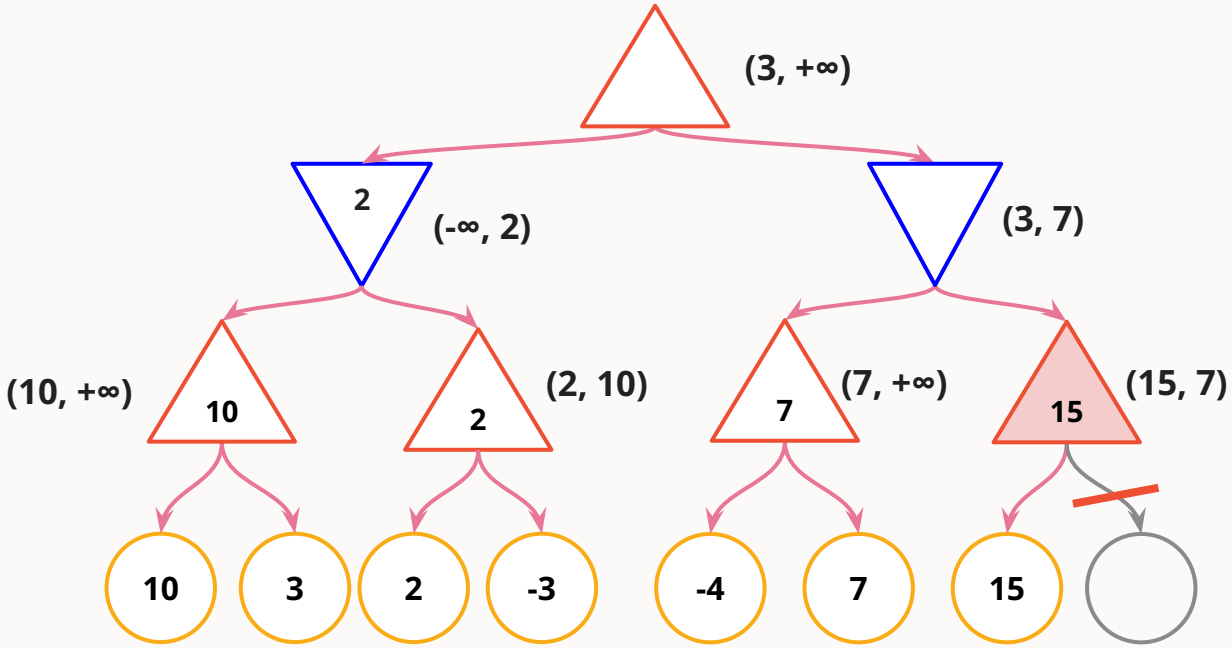
Poda Alpha-Beta



Poda Alpha-Beta



Poda Alpha-Beta



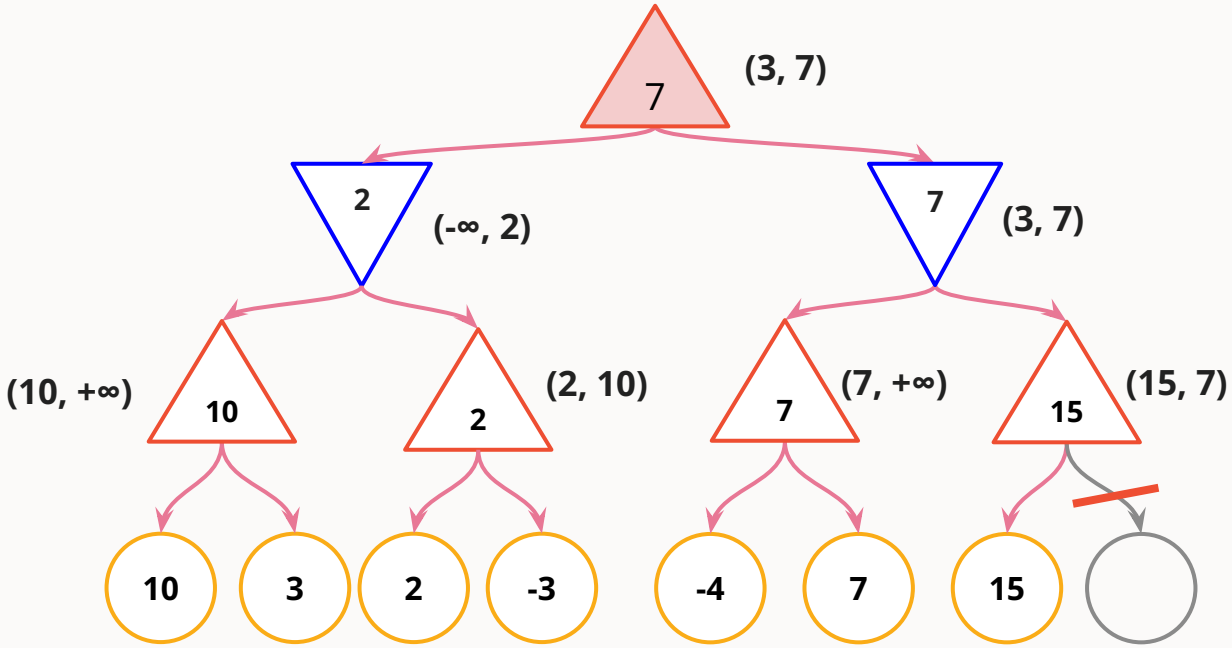
Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta



Jugador Max

Jugador Min

Estados Terminales

(alpha, beta)

Poda Alpha-Beta

Aunque en este ejemplo solo se eliminó un nodo, hay casos en que puede eliminar ramas enteras de un árbol. Influye el orden de los nodos, por lo que funciona mejor cuando las jugadas están ordenadas por prioridad.

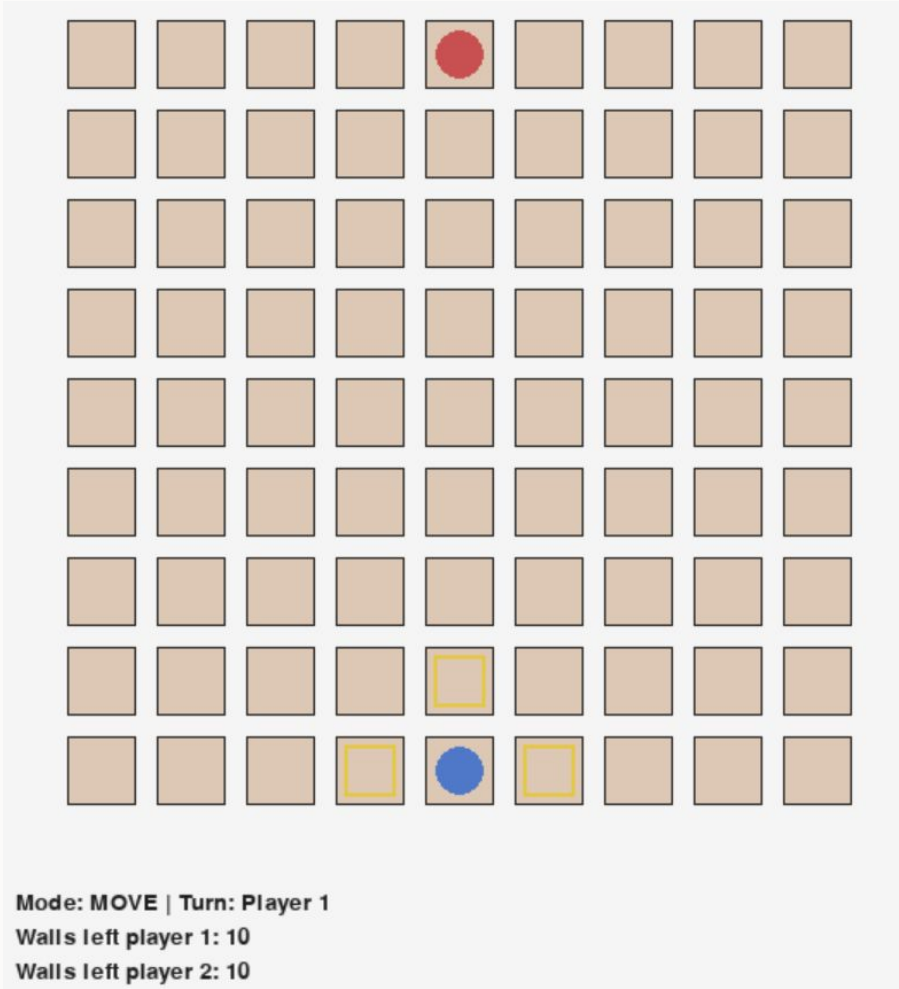
Lo importante es que NO afecta la optimalidad, ya que solamente elimina nodos que no pueden mejorar la solución.

Tarea 3

DCC Quoridor

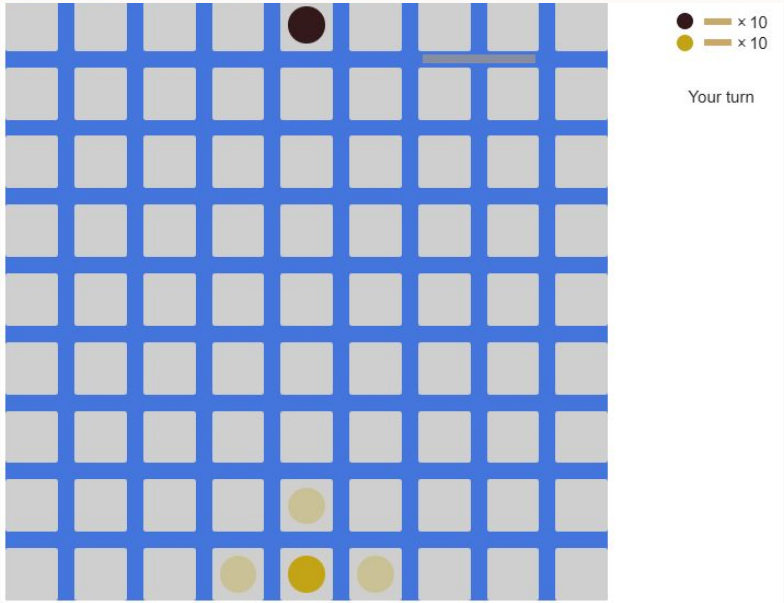
Juego

- 2 jugadores adversarios
- Cada jugador posee 10 paredes
- Cada turno se debe decidir si poner una pared o mover el peón
- **SIEMPRE** debe existir un camino para ganar
- **OBJETIVO**: llegar al extremo opuesto antes que el oponente



Juego

Link: https://es.igre.games/quoridor-online/play/



Actividades

- ***Act. 1: Poda Alpha-Beta***
- *Act. 2: Comparación profundidades*
- *Act. 3: Comparación de funciones de evaluación*
- *Act. 4: Nueva función de evaluación*

Poda Alpha-Beta

```
function MAX-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← −∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
    if v2 > v then
      v, move ← v2, a
      α ← MAX(α, v)
    if v ≥ β then return v, move
  return v, move

function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  v ← +∞
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
    if v2 < v then
      v, move ← v2, a
      β ← MIN(β, v)
    if v ≤ α then return v, move
  return v, move
```

Poda Alpha-Beta

```
player1 = MinimaxPlayer("AI 1", depth=3, eval_fun=evaluate, use_alphabeta=True)
player2 = MinimaxPlayer("AI 2", depth=2, eval_fun=chat_gpt_eval, use_alphabeta=True)
```

```
def minimax(game, player_id,

        # Poda alfa-beta
        if use_alphabeta:
            # COMPLETAR
            pass
```

Actividades

- *Act. 1: Poda Alpha-Beta*
- ***Act. 2: Comparación profundidades***
- *Act. 3: Comparación de funciones de evaluación*
- *Act. 4: Nueva función de evaluación*

Comparación profundidades

Depth = 3

Depth = 2

```
player1 = MinimaxPlayer("AI 1", depth=3, eval_fun=evaluate, use_alphabeta=True)
player2 = MinimaxPlayer("AI 2", depth=2, eval_fun=chat_gpt_eval, use_alphabeta=True)
```

Actividades

- *Act. 1: Poda Alpha-Beta*
- *Act. 2: Comparación profundidades*
- ***Act. 3: Comparación de funciones de evaluación***
- *Act. 4: Nueva función de evaluación*

Comparación de funciones de evaluación

Depth = evaluate

Depth = chat_gpt_eval

```
player1 = MinimaxPlayer("AI 1", depth=3, eval_fun=evaluate, use_alphabeta=True)
player2 = MinimaxPlayer("AI 2", depth=2, eval_fun=chat_gpt_eval, use_alphabeta=True)
```

Actividades

- *Act. 1: Poda Alpha-Beta*
- *Act. 2: Comparación profundidades*
- *Act. 3: Comparación de funciones de evaluación*
- ***Act. 4: Nueva función de evaluación***

Nueva función de evaluación

score.py

quoridor_alumnos > score.py > evaluate

```
21    def chat_gpt_eval(game, player_idx):
52        tempo = 0.15 if (game.turn % 2) == player_idx else -0.15
53
54        return dist_score + wall_score + center_score + progress_score + tempo
55
56    # Acá defines tus funciones de evaluación
57
```

Importante

- Reportar TODOS los experimentos pedidos
- Las profundidades más altas pueden demorarse, así que partan con anticipación
- Para el desarrollo, se recomienda apagar la visualización, así se demora menos y hay menos riesgo de quedarse pegado

DCCBrazo


igus

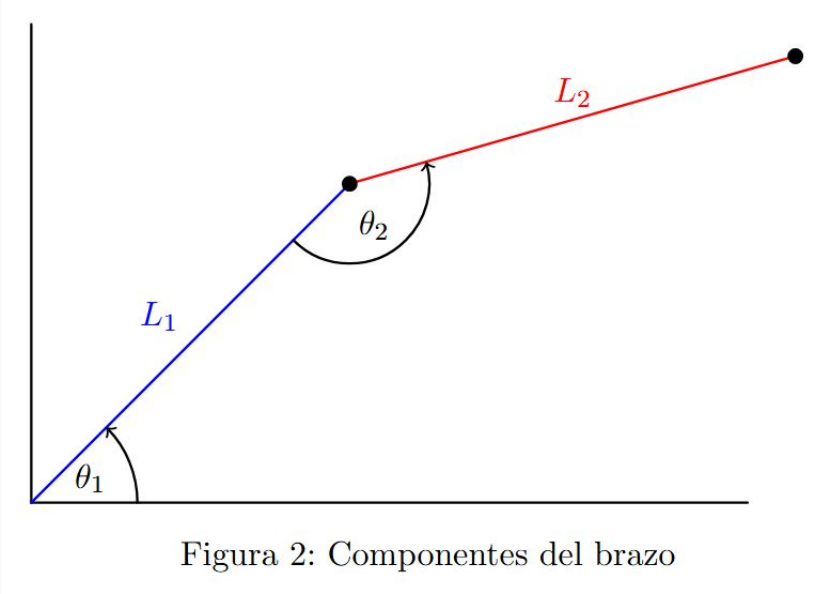
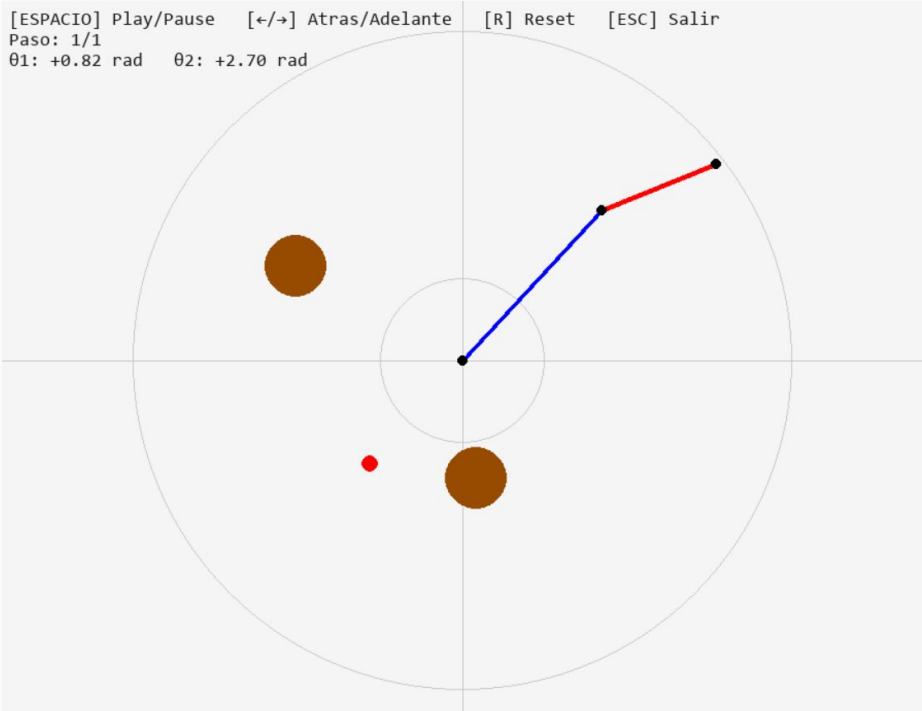


Figura 2: Componentes del brazo



Actividades

- *Act. 1: Ecuaciones y heurística*
- *Act. 2: Problemas sin obstáculos*
- *Act. 3: Problemas con obstáculos*
- *Act. 4: Weighted A**
- *Act. 5: Búsqueda Anytime*

Actividades

- ***Act. 1: Ecuaciones y heurística***
- *Act. 2: Problemas sin obstáculos*
- *Act. 3: Problemas con obstáculos*
- *Act. 4: Weighted A**
- *Act. 5: Búsqueda Anytime*

Act. 1: Ecuaciones y heurística

Deberás implementar dos funciones:

- **`ang_to_cart(theta1, theta2, L1, L2):`** Función que dado los ángulos theta1, theta2 y los largos L1, L2 de los segmentos del brazo, retorna la posición del extremo de cada segmento.
- **`heuristic_trig(state, goal, L1, L2):`** Función heurística que recibe el estado (los ángulos del brazo), el objetivo (las coordenadas a las que queremos que llegue el final del brazo) y los largos de los segmentos del brazo

ang_to_cart()

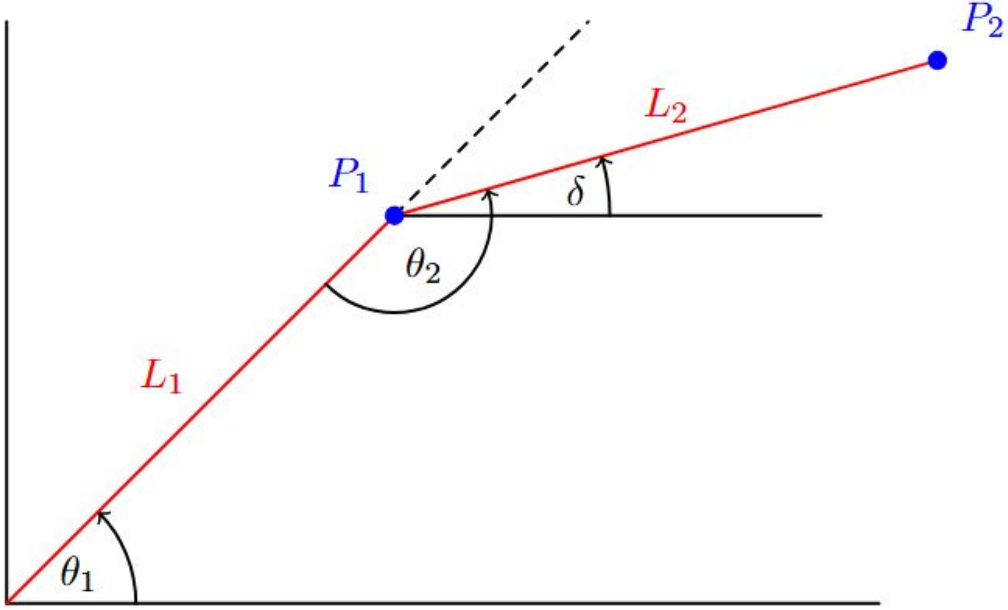
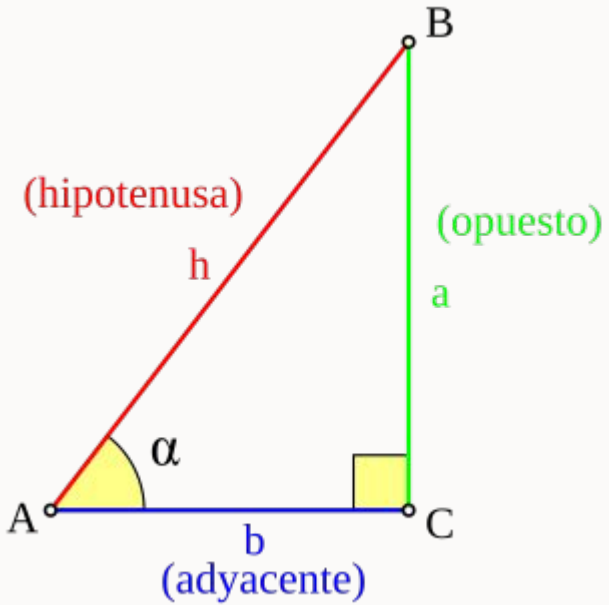


Figura 4: Brazo en el plano cartesiano

$$\text{sen}\,\alpha = \frac{\text{opuesto}}{\text{hipotenusa}} = \frac{a}{h}$$

$$\cos\alpha = \frac{\text{adyacente}}{\text{hipotenusa}} = \frac{b}{h}$$

$$\text{tg}\,\alpha = \frac{\text{opuesto}}{\text{adyacente}} = \frac{a}{b}$$



heuristic_trig()

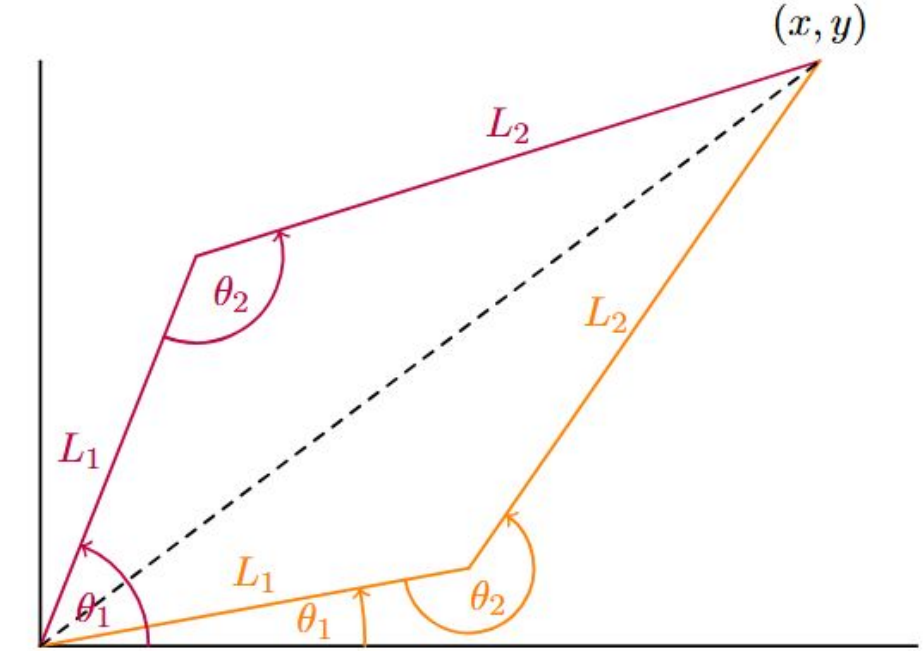
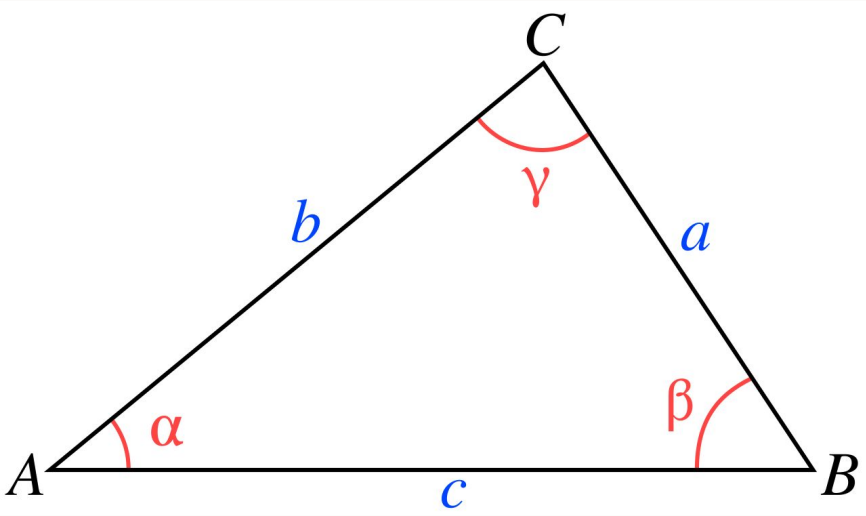


Figura 5: Brazo en el plano cartesiano



$$c^2 = a^2 + b^2 - 2ab\cos\gamma,$$
$$a^2 = b^2 + c^2 - 2bc\cos\alpha,$$
$$b^2 = a^2 + c^2 - 2ac\cos\beta.$$

Actividades

- *Act. 1: Ecuaciones y heurística*
- ***Act. 2: Problemas sin obstáculos***
- *Act. 3: Problemas con obstáculos*
- *Act. 4: Weighted A**
- *Act. 5: Búsqueda Anytime*

Act. 2: Problemas sin obstáculos

- **BFS**
- **A*** con heurística `zero`
- **A*** con heurística `heuristic_trig`

Actividades

- *Act. 1: Ecuaciones y heurística*
- *Act. 2: Problemas sin obstáculos*
- ***Act. 3: Problemas con obstáculos***
- *Act. 4: Weighted A**
- *Act. 5: Búsqueda Anytime*

Act. 3: Problemas con obstáculos

- **BFS**
- **A*** con heurística `zero`
- **A*** con heurística `heuristic_trig`

Actividades

- *Act. 1: Ecuaciones y heurística*
- *Act. 2: Problemas sin obstáculos*
- *Act. 3: Problemas con obstáculos*
- ***Act. 4: Weighted A****
- *Act. 5: Búsqueda Anytime*

Act. 4: Weighted A*

- **A*** con heurística `heuristic_trig` y w = 2
- **A*** con heurística `heuristic_trig` y w = 3
- **A*** con heurística `heuristic_trig` y un valor de w que te resulte interesante

Actividades

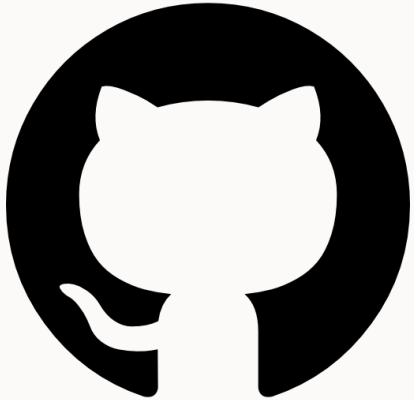
- *Act. 1: Ecuaciones y heurística*
- *Act. 2: Problemas sin obstáculos*
- *Act. 3: Problemas con obstáculos*
- *Act. 4: Weighted A**
- ***Act. 5: Búsqueda Anytime***

Act. 5: Búsqueda Anytime

- A grandes rasgos, les pedimos implementar una búsqueda donde el peso (w) es **variable**

$$w = \frac{c_{\text{ult}}}{\min_{s \in Open}(g(s) + h(s))}$$

Pueden utilizar las issues para responder sus dudas :)



Dudas