GAME TREE

¿Agentes en nuestra contra? ¡Juegos!

Juego: entorno con más de un agente
Problema: 1 agente

VARIABLES

- Determinista o estocástico ?
- Completamente observable ?
- 2, 3, o más jugadores ?
- Equipo o individual ?
- Por turnos o simultáneas ?
- Adversarial o cooperativo ?
- Suma zero ?
  (uno gana y otro pierde)

☆ ganancia = 0 → perdí o empate
En suma zero es darle la menor ganancia al adversario

Nos interesan los de suma zero y los de suma zero determinista con 2 agentes.

¿Qué tengo?

- S := Conjunto de estados
  $S_0 \in S$ estado inicial
- A := Acciones
- P := Jugadores $P = \{1, \dots, N\}$
- $S \times P \to \mathcal{P}(A)$:= acciones_legales
- $S \times A \times P \to S$:= transición
- $S \to \{T, F\}$:= es_terminal
- $S \times P \to \mathbb{R}$:= ganancia

POLÍTICA (Policy): Solución para un jugador $\pi_p = S \to A$

```
class ModeloJuegoDetTZ2
    def inicializa(self):
        return (s0, j)        j ∈ {-1, 1}
    def acciones_legales(self, s, j):
        return {a1, a2, ...}
    def transición(self, f, a, j):
        return s`
    def terminal(self, s):
        return bool
    def ganancia(self, s):
        return ganancia de 1
```

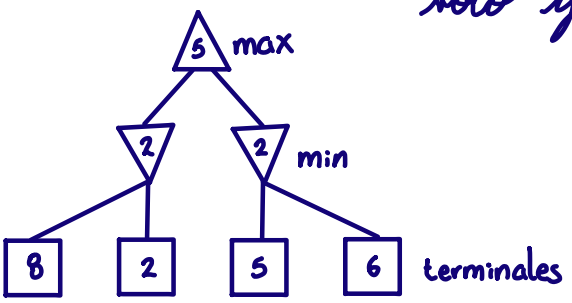
ÁRBOL DE JUEGO:
Todas las posibilidades de lo que yo hago y lo que hace mi oponente.

EJ. TIC-TAC-TOE
$s_0 = \{0,0,0,0,0,0,0,0,0\}$
$s` = \{1,0,-1,0,0,0,0,0,0\}$
donde yo soy 1 y mi oponente -1
La profundidad del árbol es $9 < 9!$
└ Solo hay 9 jugadas

MINIMAX:

- Juegos de suma zero determinísticos: gato, ajedrez, checkers
- Un jugador maximiza el resultado, el otro minimiza el resultado.

Hoja galería

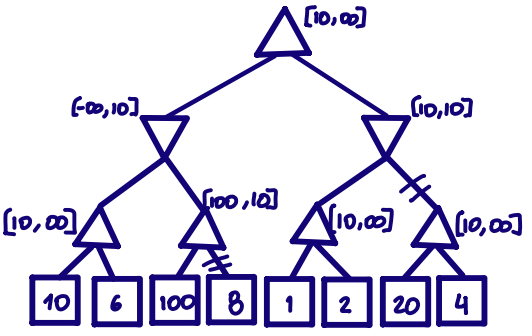



```
def max-value(state):
    return max
def min-value(state):
    return min
def value(state):
    if Δ return max-value
    else return min-value
```

Siempre pregunta los max y min con base a los valores terminales.

* Siempre es bueno contra un jugador perfecto, si no hay jugador perfecto, regresa como el jugador perfecto

EFICIENCIA: Tiempo: $O(b^m)$ y Espacio: $O(bm)$
└ Es demasiado lento, hay que reducirlo
→ Para reducir el tiempo lo ¡PODAMOS!



PODA ALPHA-BETA

Podemos pasar de un tiempo de $O(b^m)$ a $O(b^{m/2})$ en el mejor caso, $O(b^{3/4 \cdot m})$ en el caso promedio.
→ Cortamos camino al no buscar más opciones cuando nuestra cota inferior es mayor a la cota superior, porque entonces ya no hay solución.

α: Mejor opción de MAX
β: Mejor opción de MIN

```
def MAX(...): ...
    if v ≥ α return v ...
```

```
def MIN(...): ...
    if v ≤ β return v ...
```

Muy bonito, pero no es suficiente. ¿Qué más podemos hacer?
→ Limitar la profundidad de la búsqueda.

Usamos la suma linear de pesos de los features para que funcione

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

¿Negmax? solo funciona para dos jugadores, intercala de uno a uno entre min y max