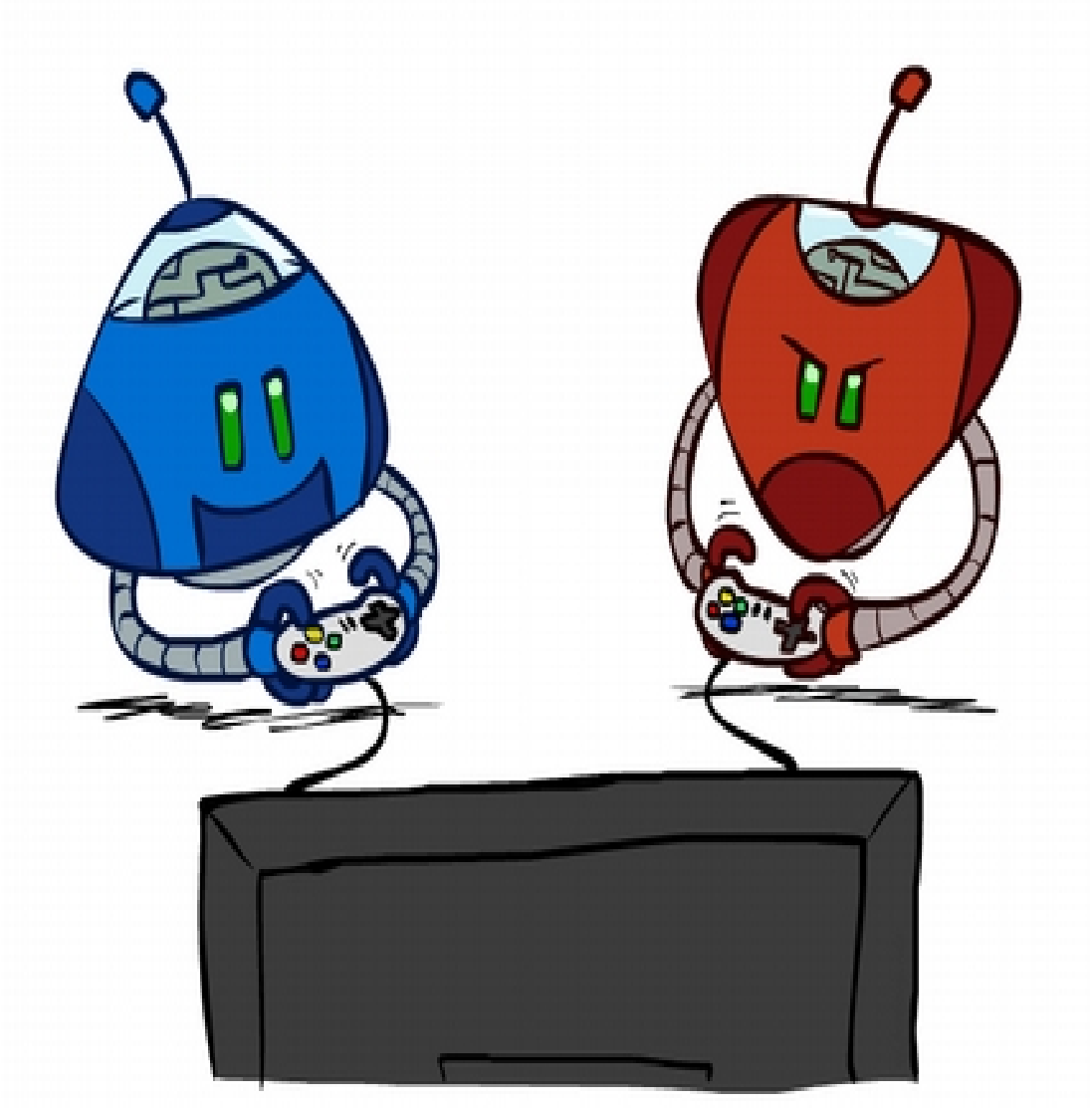
Inteligencia Artificial

Búsqueda adversarial y Árboles de juegos

[Slides adapted from Dan Klein and Pieter Abbeel (ai.berkeley.edu).

Sergey Levine & Stuart Russell University of California, Berkeley]

Búsqueda adversarial y Árboles de juegos

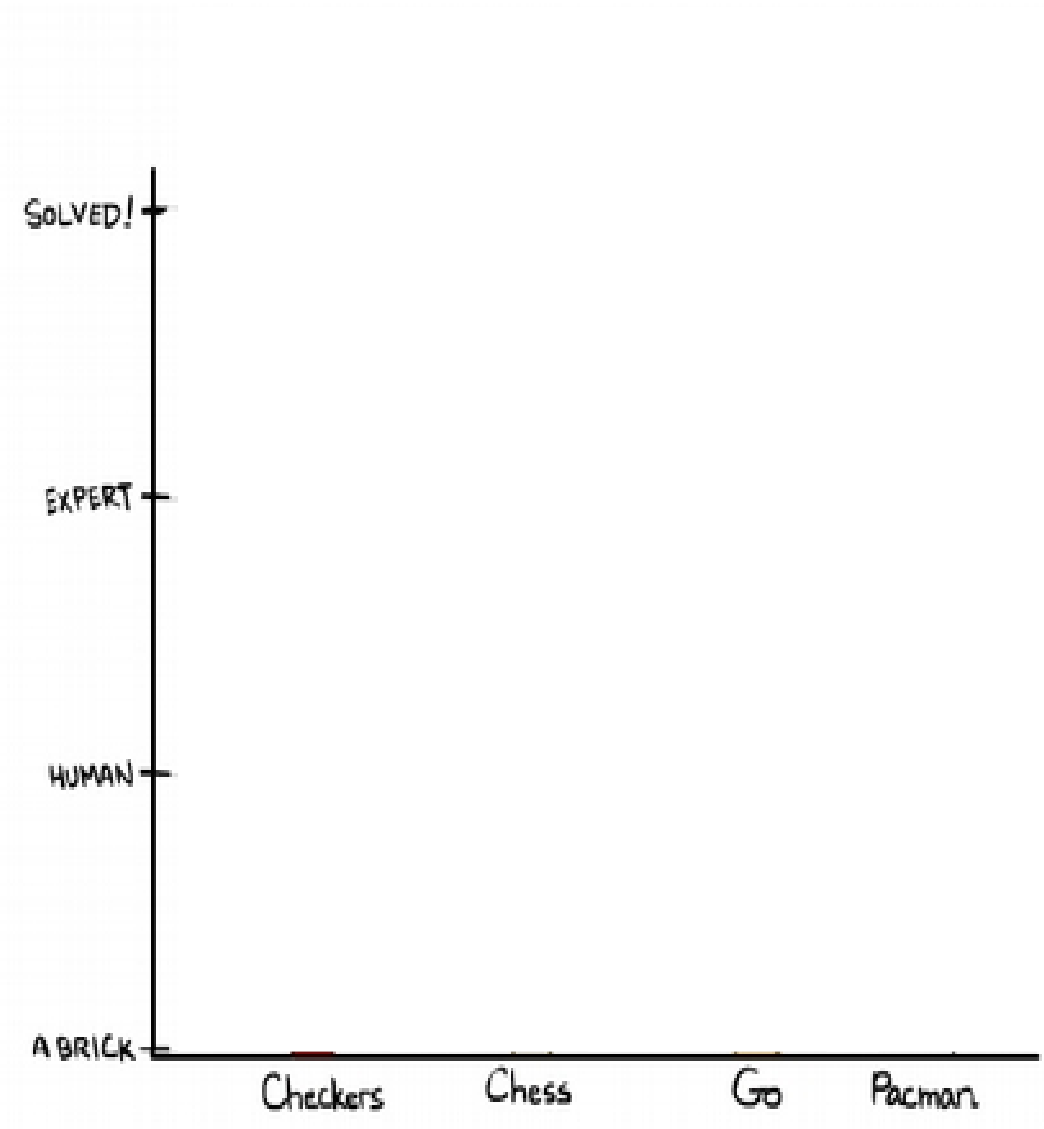


Estado del arte en juegos

- Damas: 1950: Primer sistema automático. 1994: Primer campeón automático: Chinook acabó con el reinado de 40 años de la campeona Marion Tinsley. 2007: ¡Damas resueltas!

- Ajedrez: 1997: Deep Blue derrota al campeón Gary Kasparov en un torneo de 6 partidas. Deep Blue examinó 200M posiciones por segundo, usando una evaluación sofisticada y descubrió métodos para extender varias líneas de investigación hasta 40 capas. Los programas actuales son todavía mejores.

- Go: Los campeones humanos empiezan a ser superados por las máquinas. En go, b > 300! Los programas clásicos usan bases de conocimiento de patrones, pero recientes avances usan métodos de expansión Monte Carlo (randomizados)



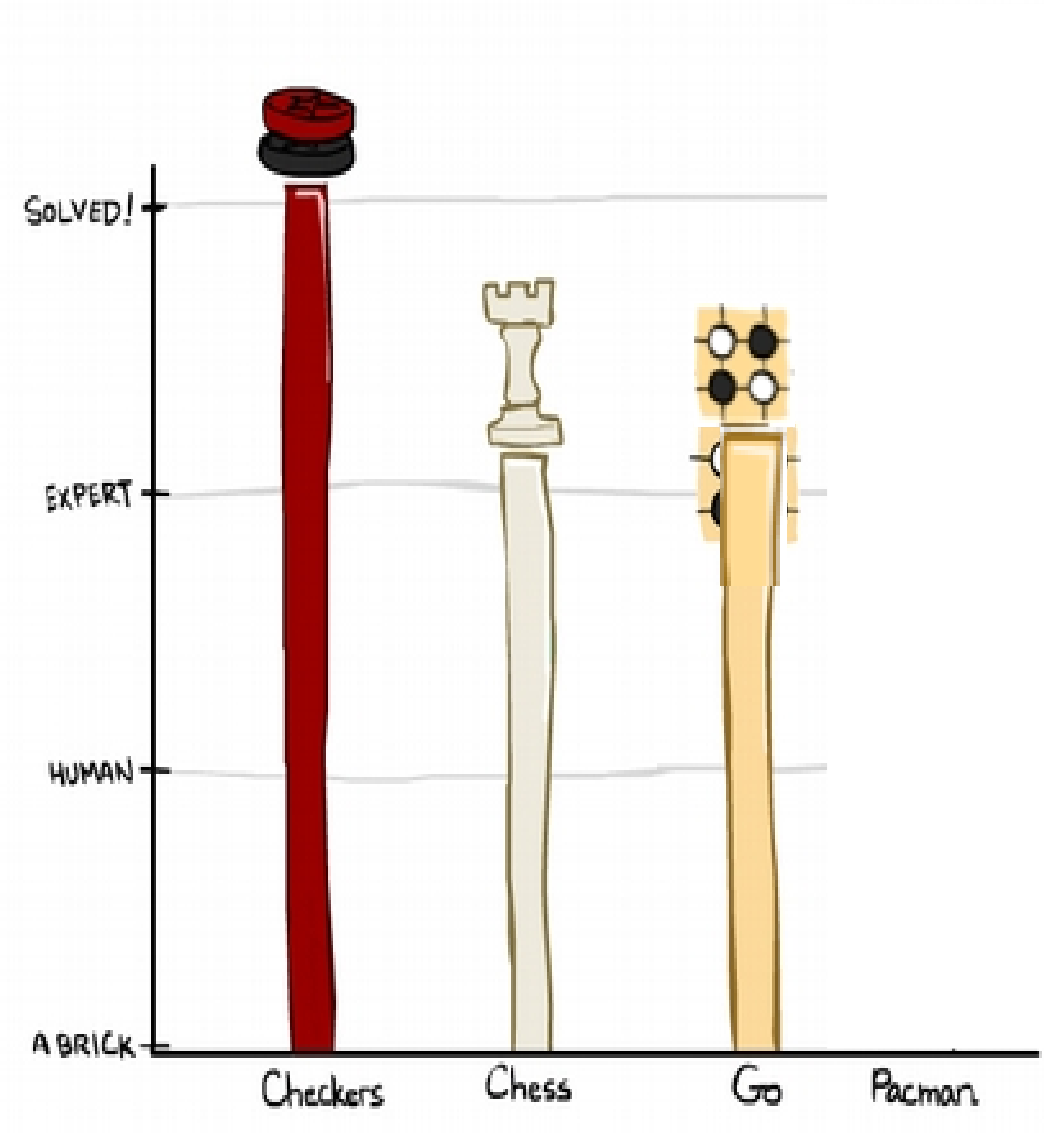
Estado del arte en juegos

- Damas: 1950: Primer sistema automático. 1994: Primer campeón automático: Chinook acabó con el reinado de 40 años de la campeona Marion Tinsley. 2007: ¡Damas resueltas!

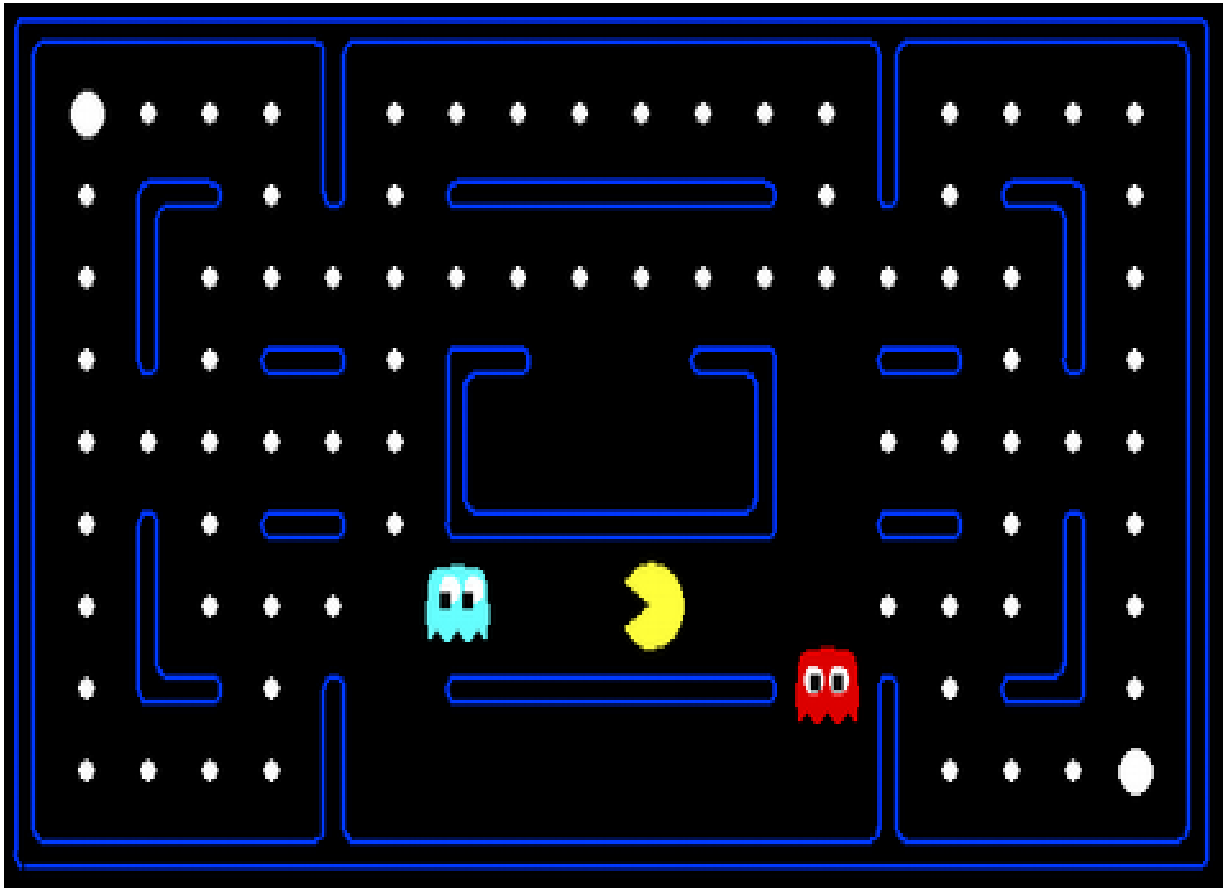
- Ajedrez: 1997: Deep Blue derrota al campeón Gary Kasparov en un torneo de 6 partidas. Deep Blue examinó 200M posiciones por segundo, usando una evaluación sofisticada y descubrió métodos para extender varias líneas de investigación hasta 40 capas. Los programas actuales son todavía mejores.

- Go: 2016: Alpha GO derrota al campeón. Usa Monte Carlo Tree Search, función de evaluación aprendida.

- Pacman



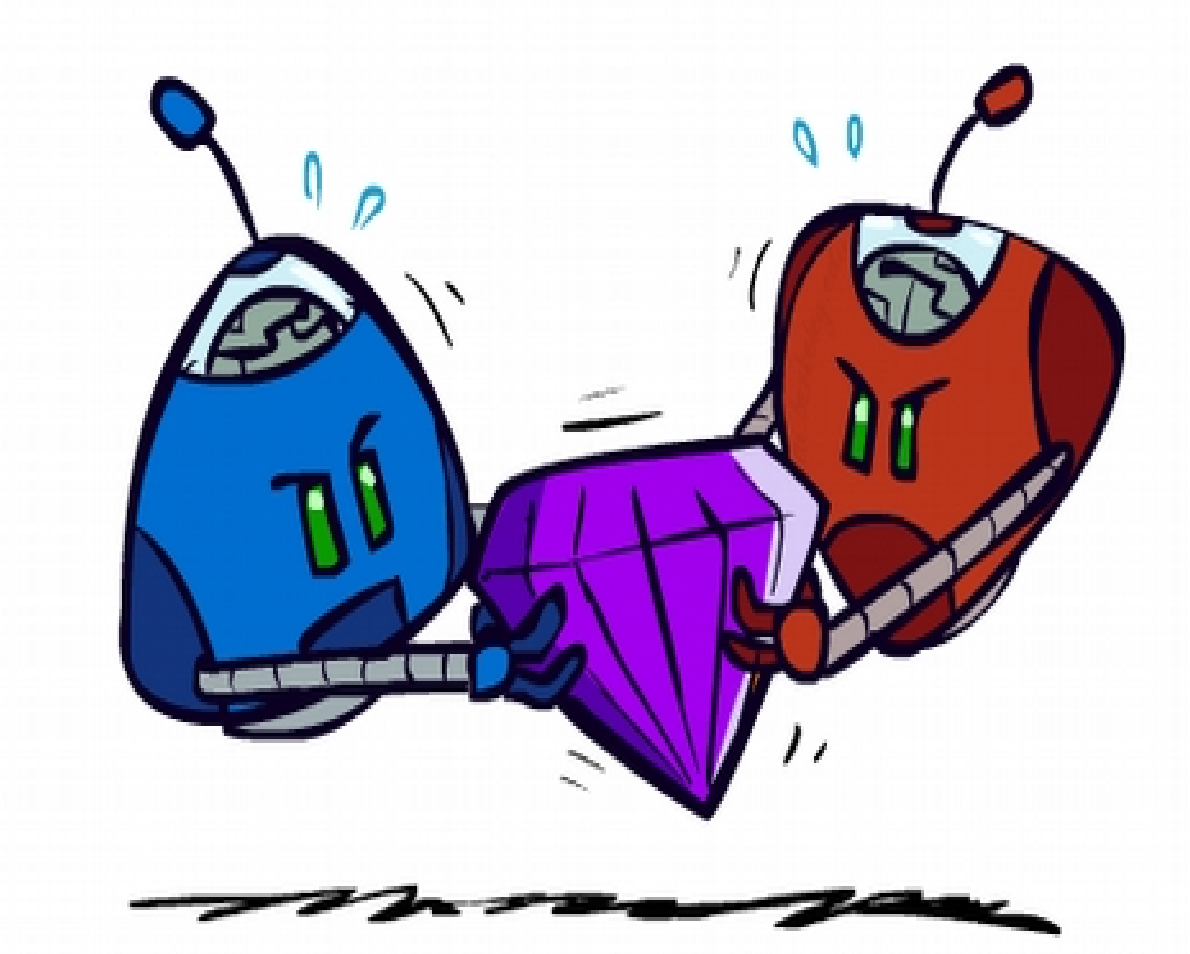
Comportamiento y computación



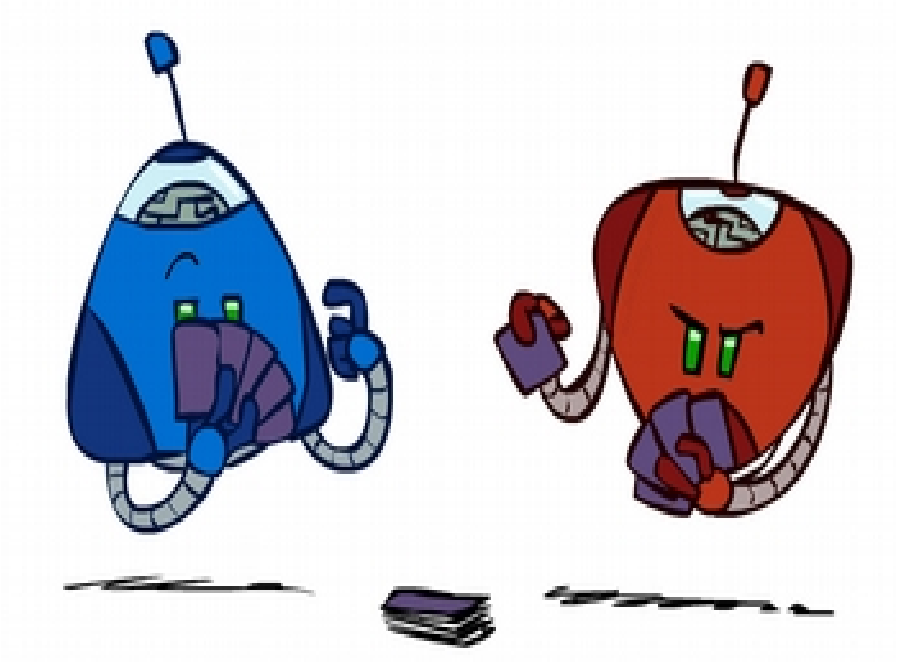
[Demo: mystery pacman (L6D1)]

Video Demo Mystery Pacman

Juegos Adversariales



Tipos de juegos



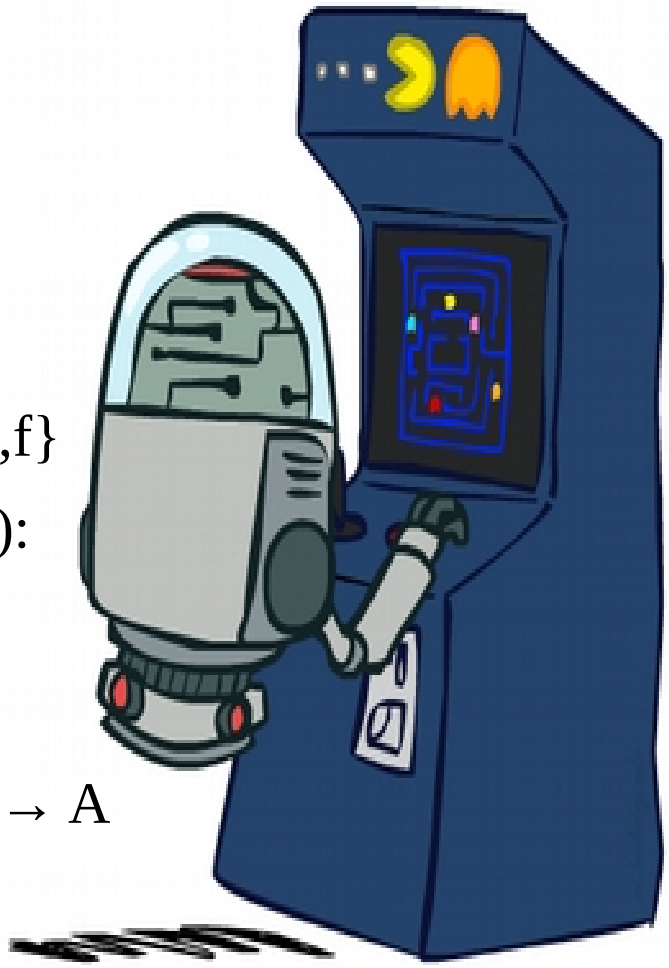
- ¡Hay muchas clases de juegos!

- A tener en cuenta:
  - ¿Determinístico o estocástico?
  - ¿Uno, dos o más jugadores?
  - ¿Suma cero?
  - ¿Información perfecta (podemos ver el estado)?

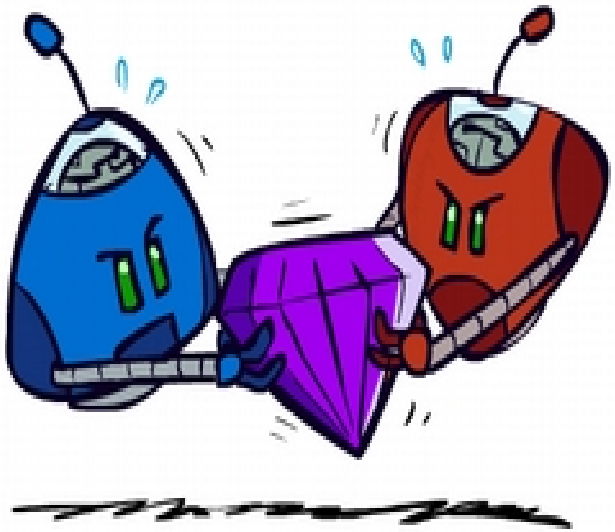
- Queremos algoritmos para calcular una estrategia (política o policy) que recomiende un movimiento desde cada estado

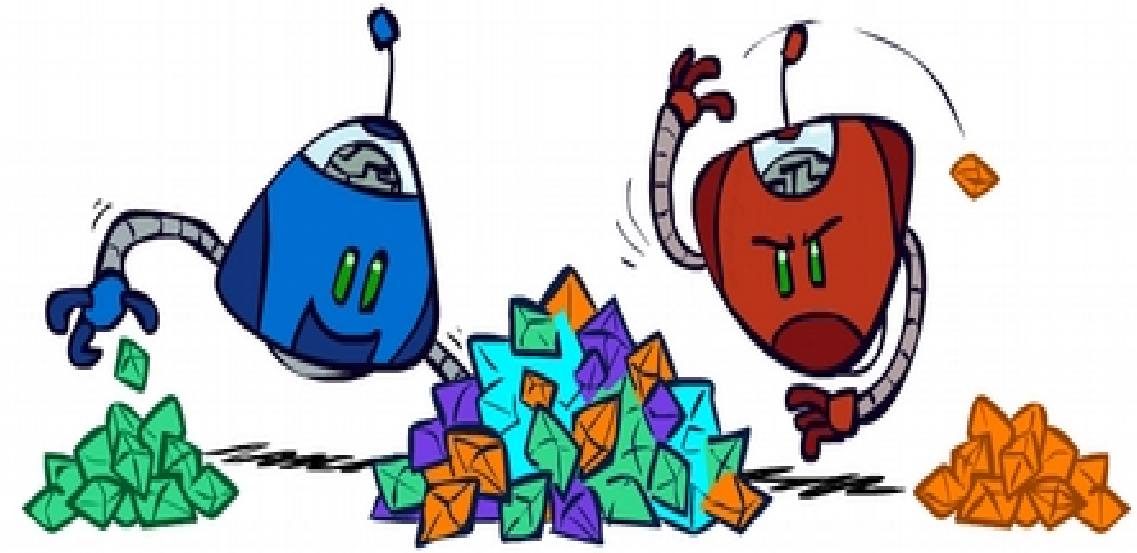
Juegos determinísticos

- Muchas formalizaciones posibles, una es:
  - Estados: S (inicio en s_0)
  - Jugadores: P={1...N} (normalmente a turnos)
  - Acciones: A (puede depender del jugador / estado)
  - Función de transición: S x A → S
  - Test de terminación (estado objetivo o final): S → {t,f}
  - Función de utilidad para terminal (Terminal Utilities):
  - ¿qué valor tiene este estado final para un jugador?
  - S x P → R
- La solución para un jugador es una política (policy): S → A



Juegos de suma cero

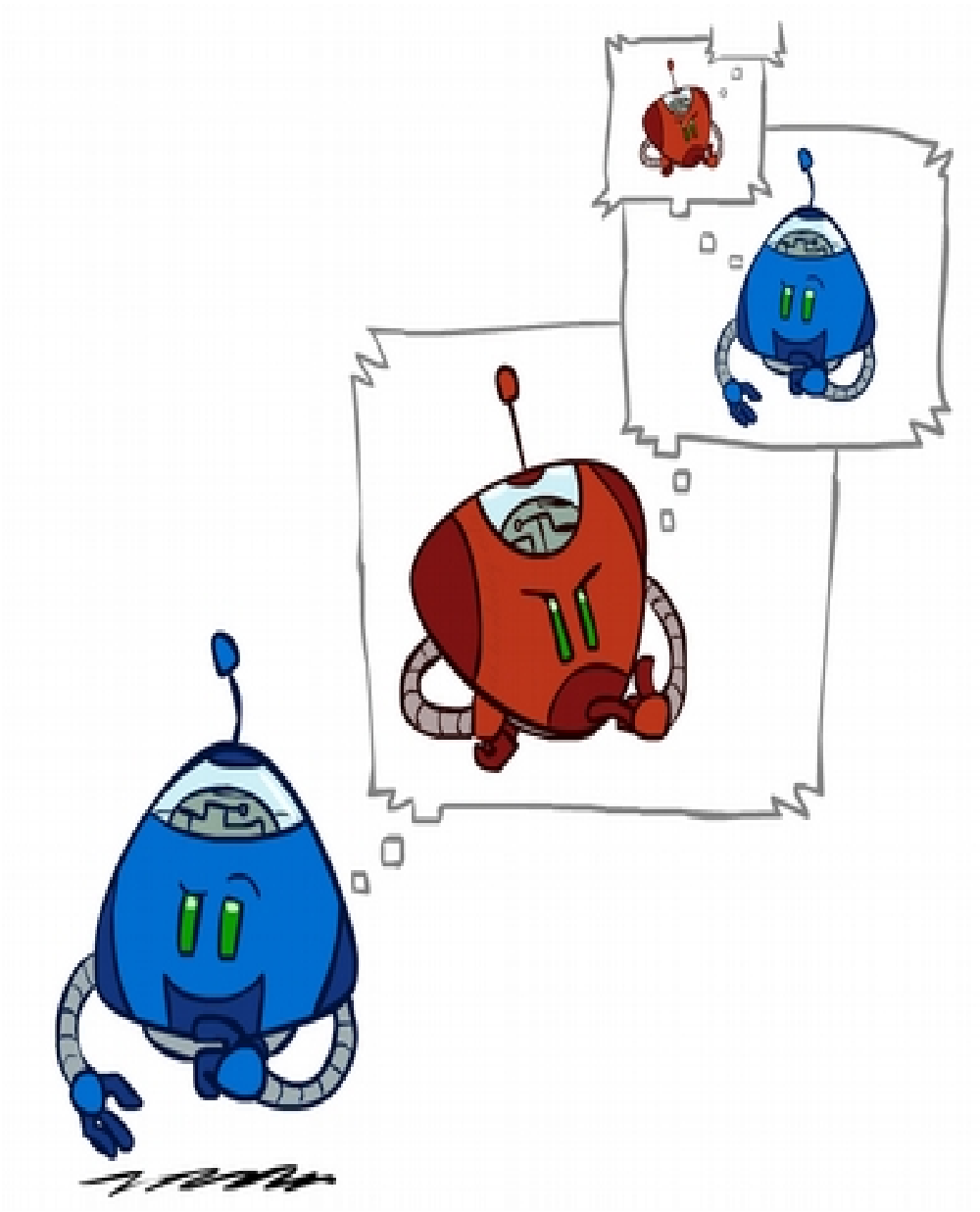




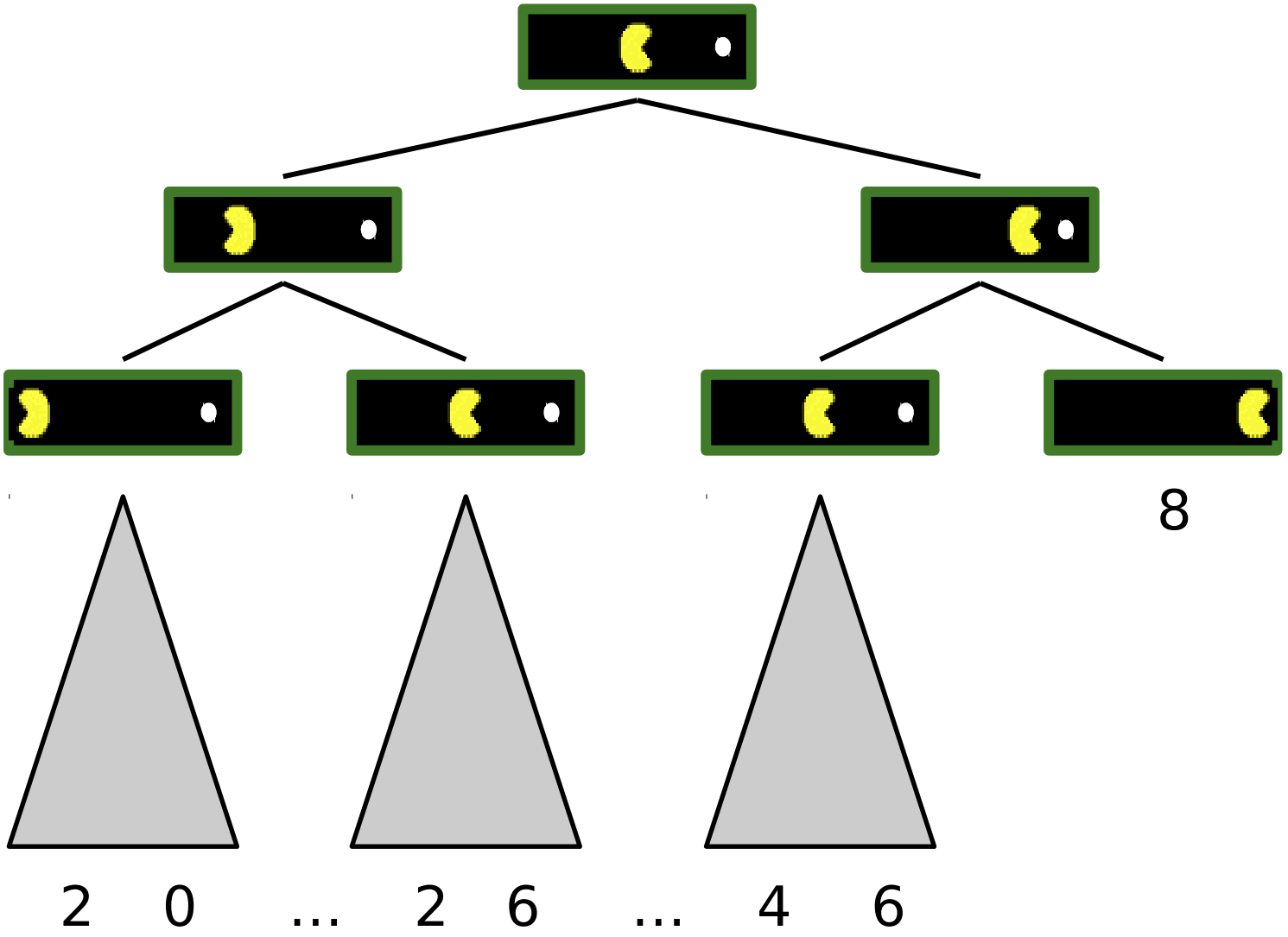
- Juegos de suma cero
  - Los agentes tienen utilidades opuestas (valores)
  - Podemos pensar en un único valor que uno maximiza y el otro minimiza
  - Adversarial, competición pura

- 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

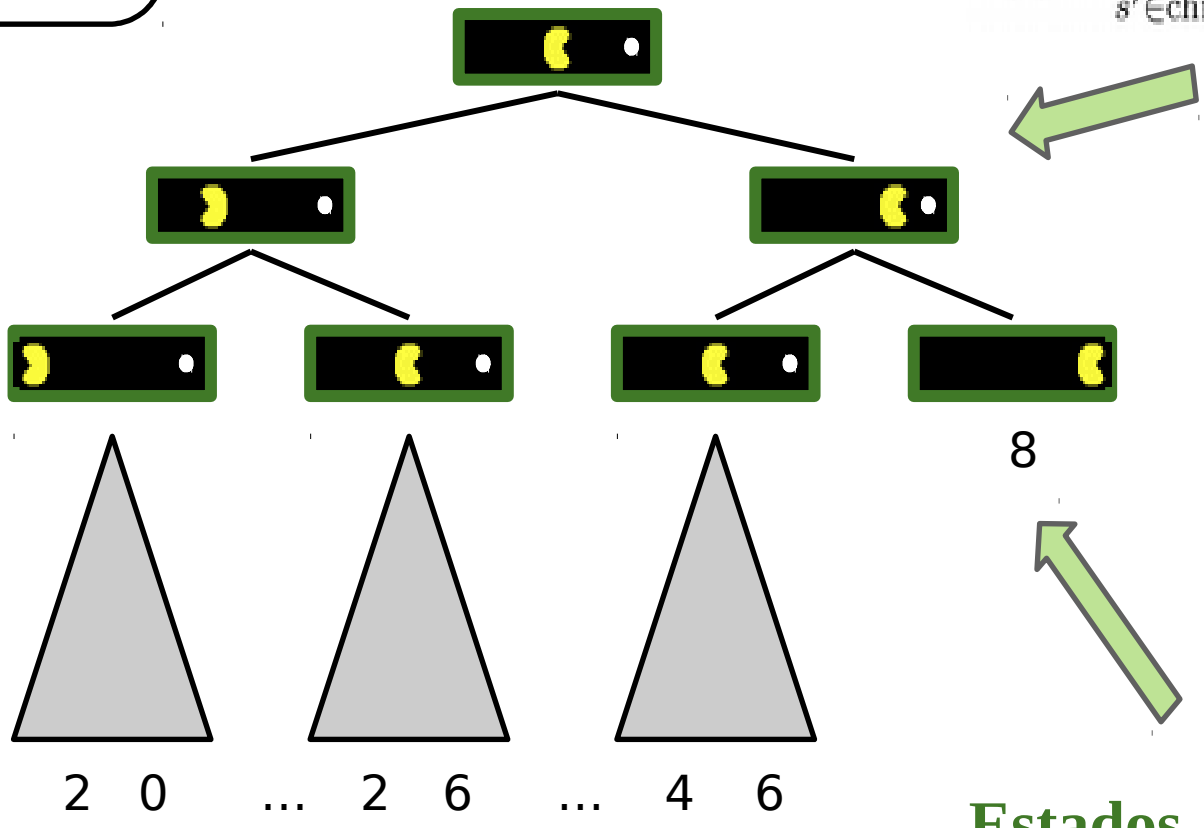


Valor de un estado

Valor de un estado: El mejor resultado (utility) desde ese estado

Estados no terminales:

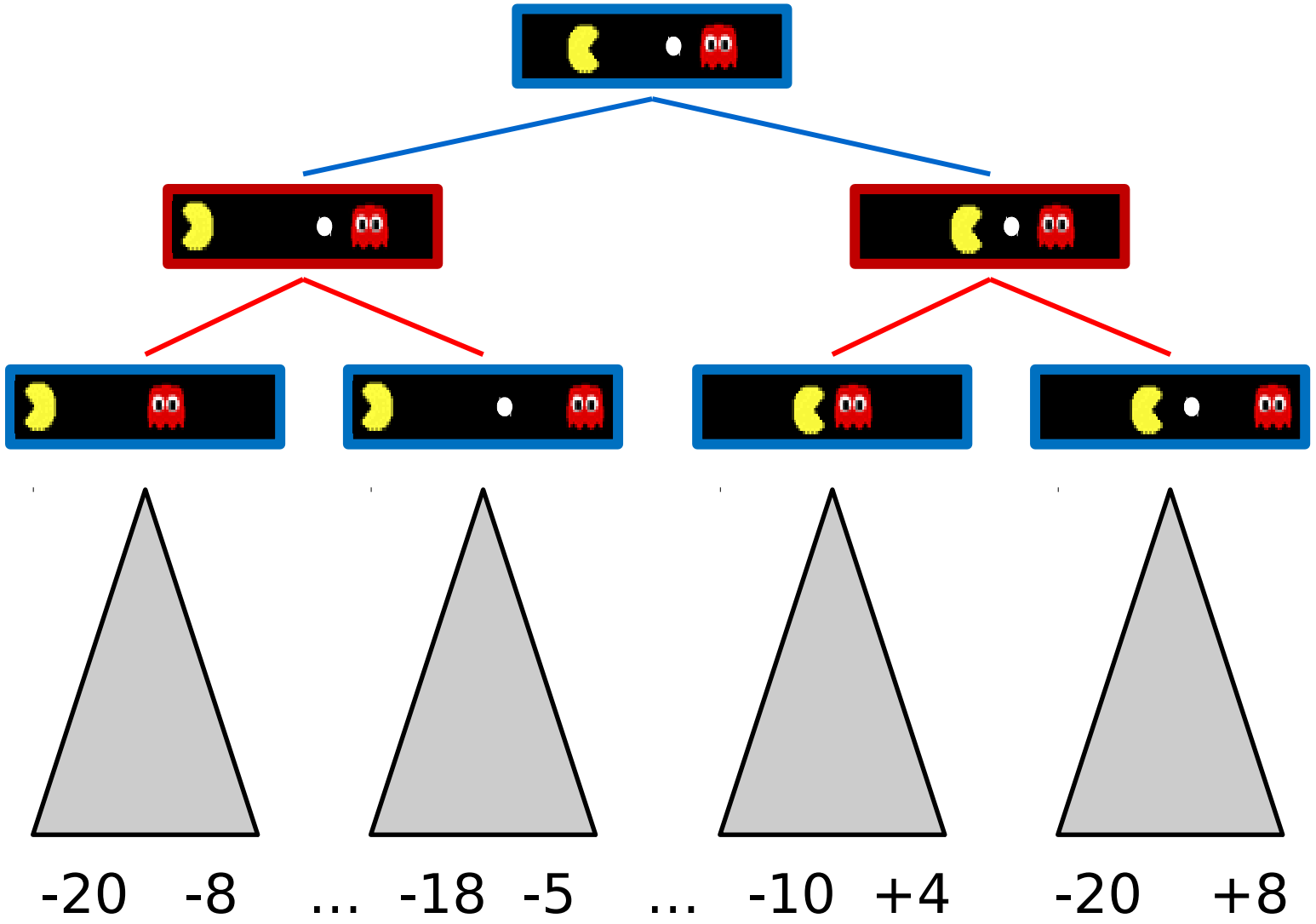
$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



Estados terminales:

V(s) = conocido

Árboles de estados Adversariales



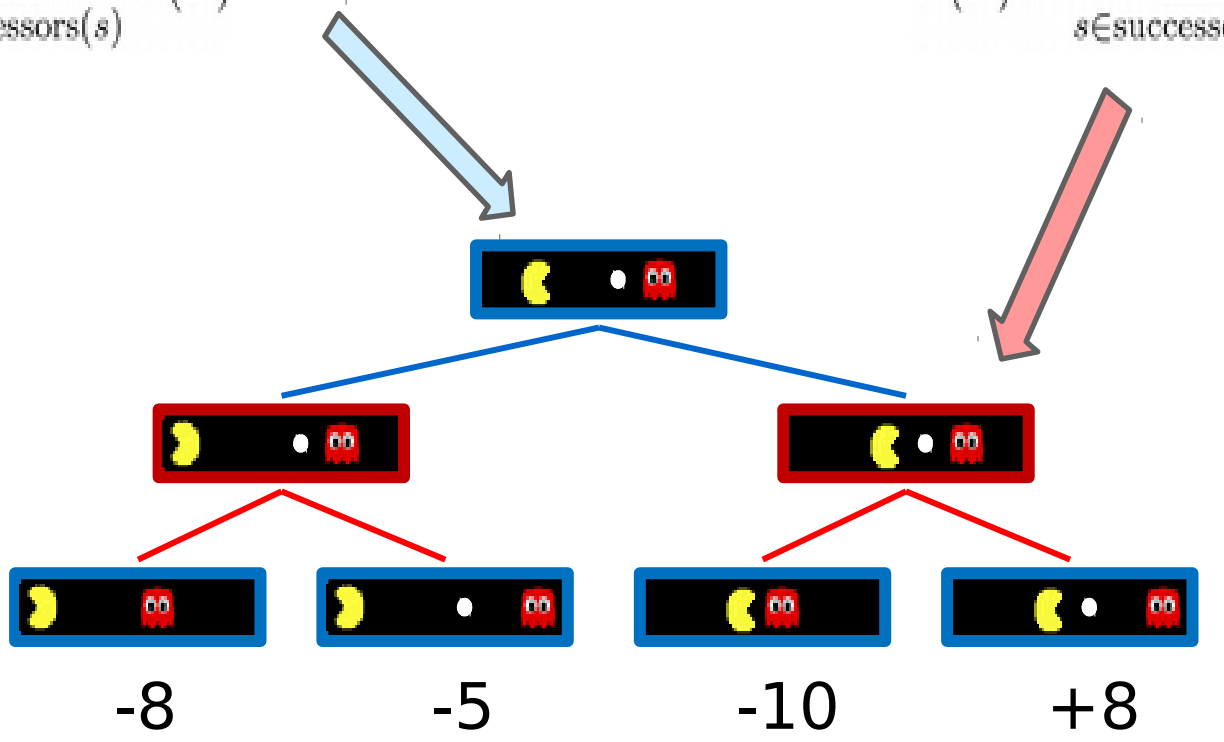
Valores Minimax

Estados bajo el control del agente:

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

Estados bajo el control del oponente:

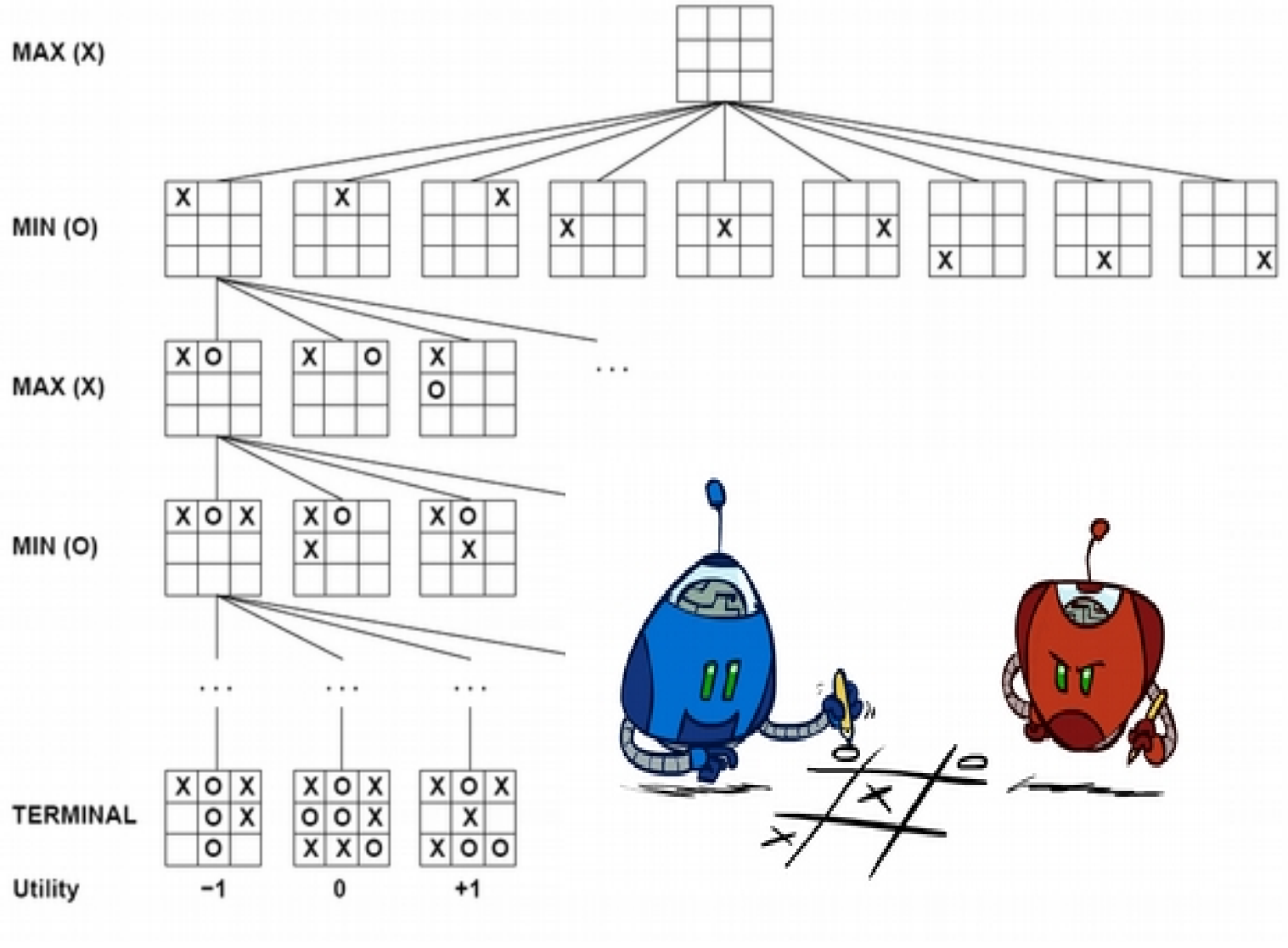
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Estados terminales:

V(s) = conocido

Árbol de juegos de Tic-Tac-Toe

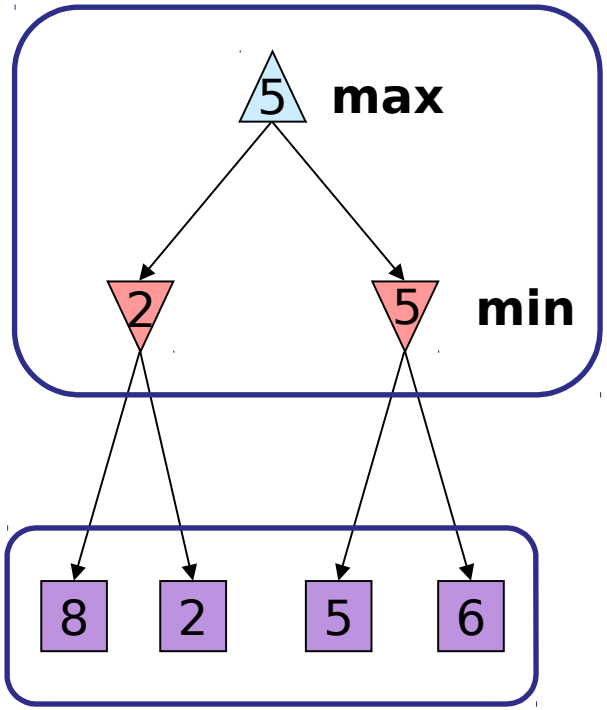


Búsqueda Adversarial (Minimax)

- 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)

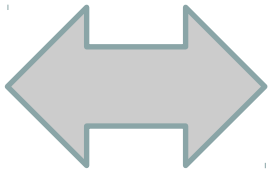
Valores Minimax:
calculados **recursivamente**



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

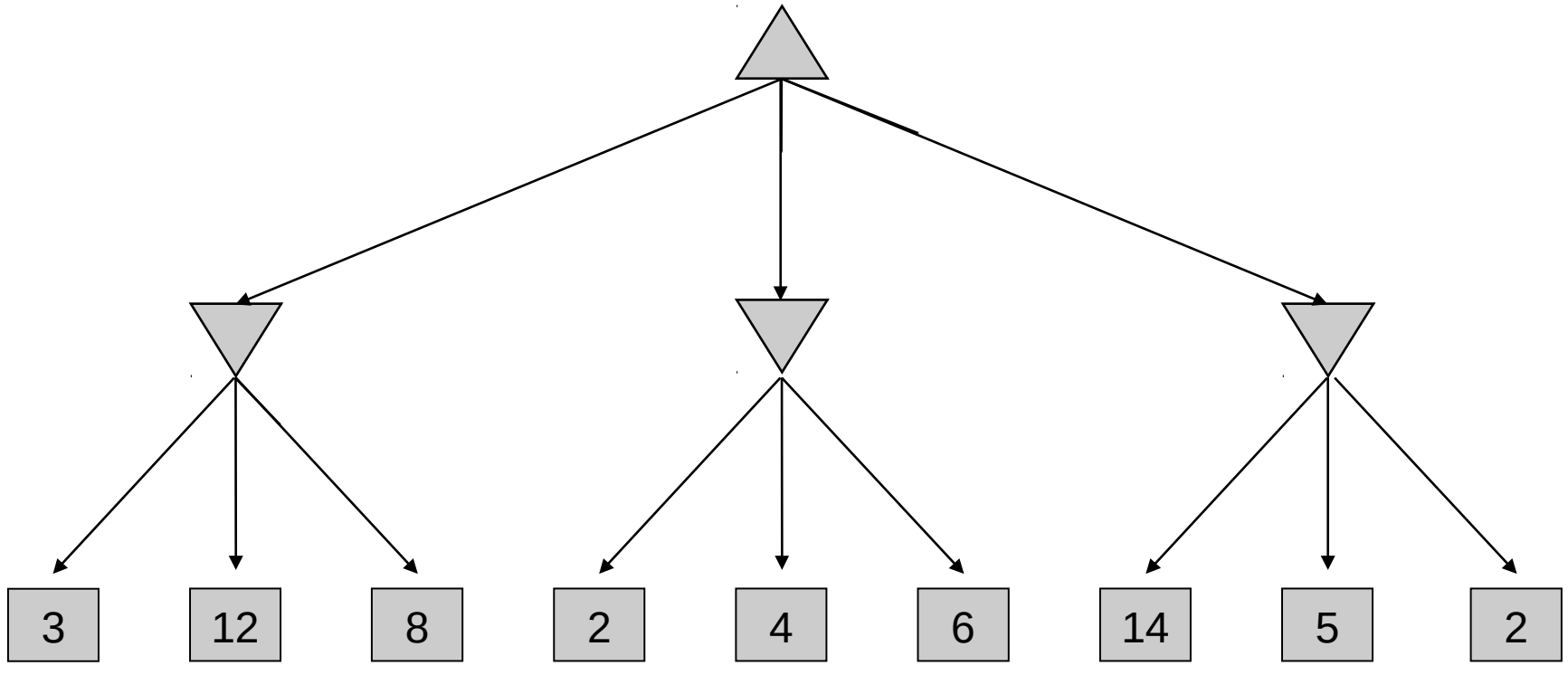
Implementación de Minimax (Dispatch)

```
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)
```

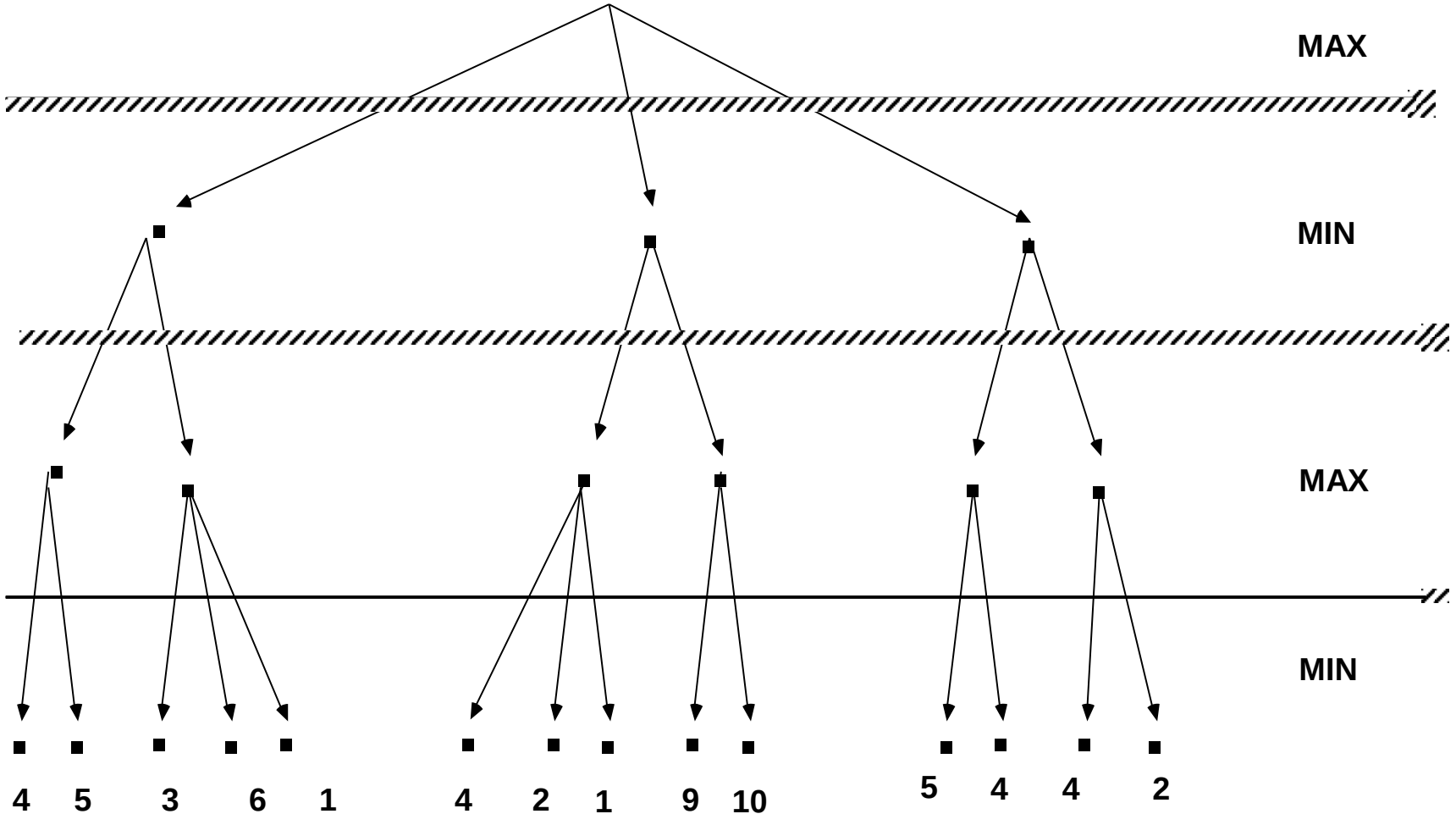
```
def max-value(state):
        initialize v = -∞
        for each successor of state:
                v = max(v,
                        value(successor))
        return v
```

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

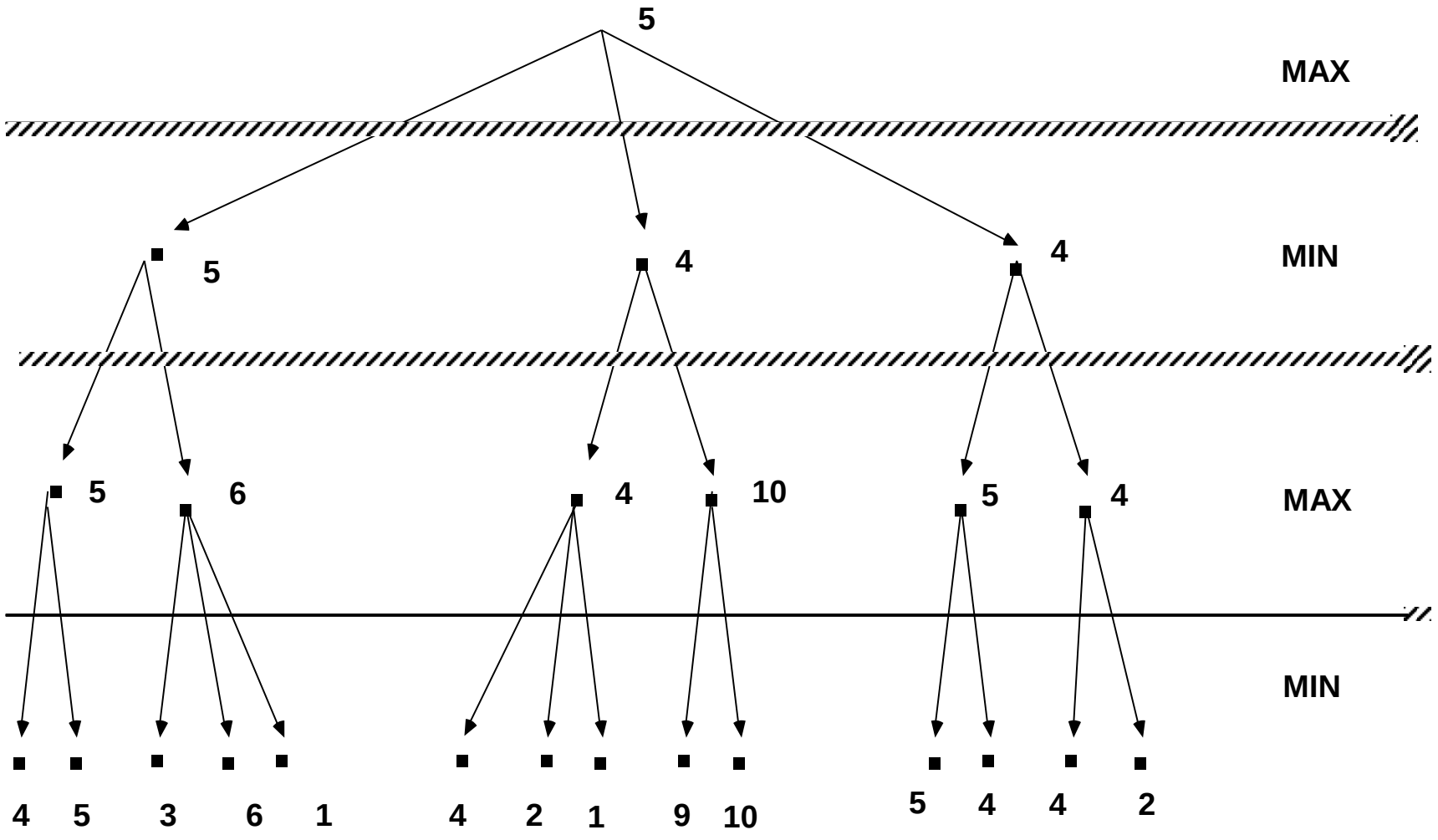
Ejemplo de Minimax



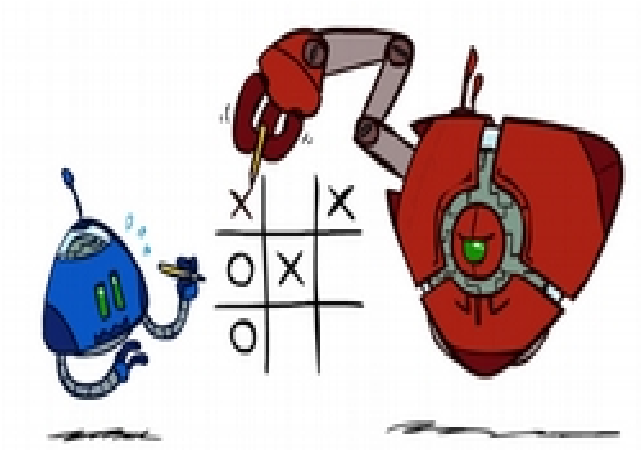
Árbol de una aplicación del mini-max

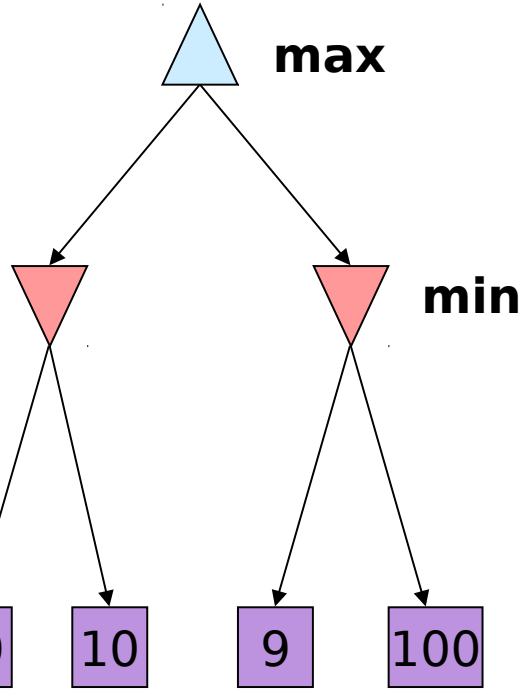


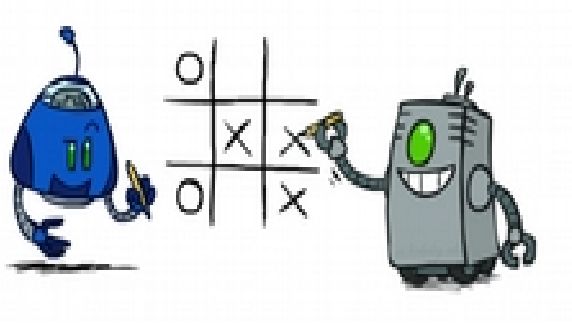
Árbol de una aplicación del mini-max



Propiedades de Minimax







Óptimo contra un jugador perfecto. ¿En otro caso?

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

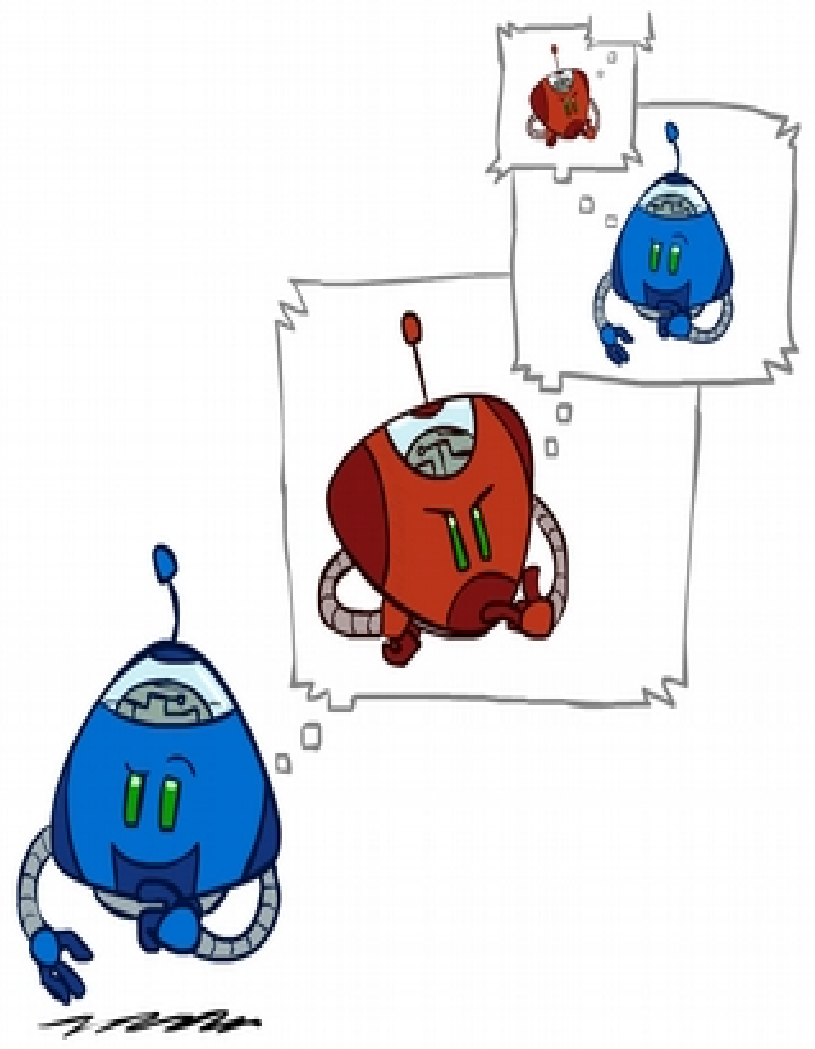
Video Demo Minimax vs. Exp (Minimax)

Video Demo Minimax vs. Exp (Exp)

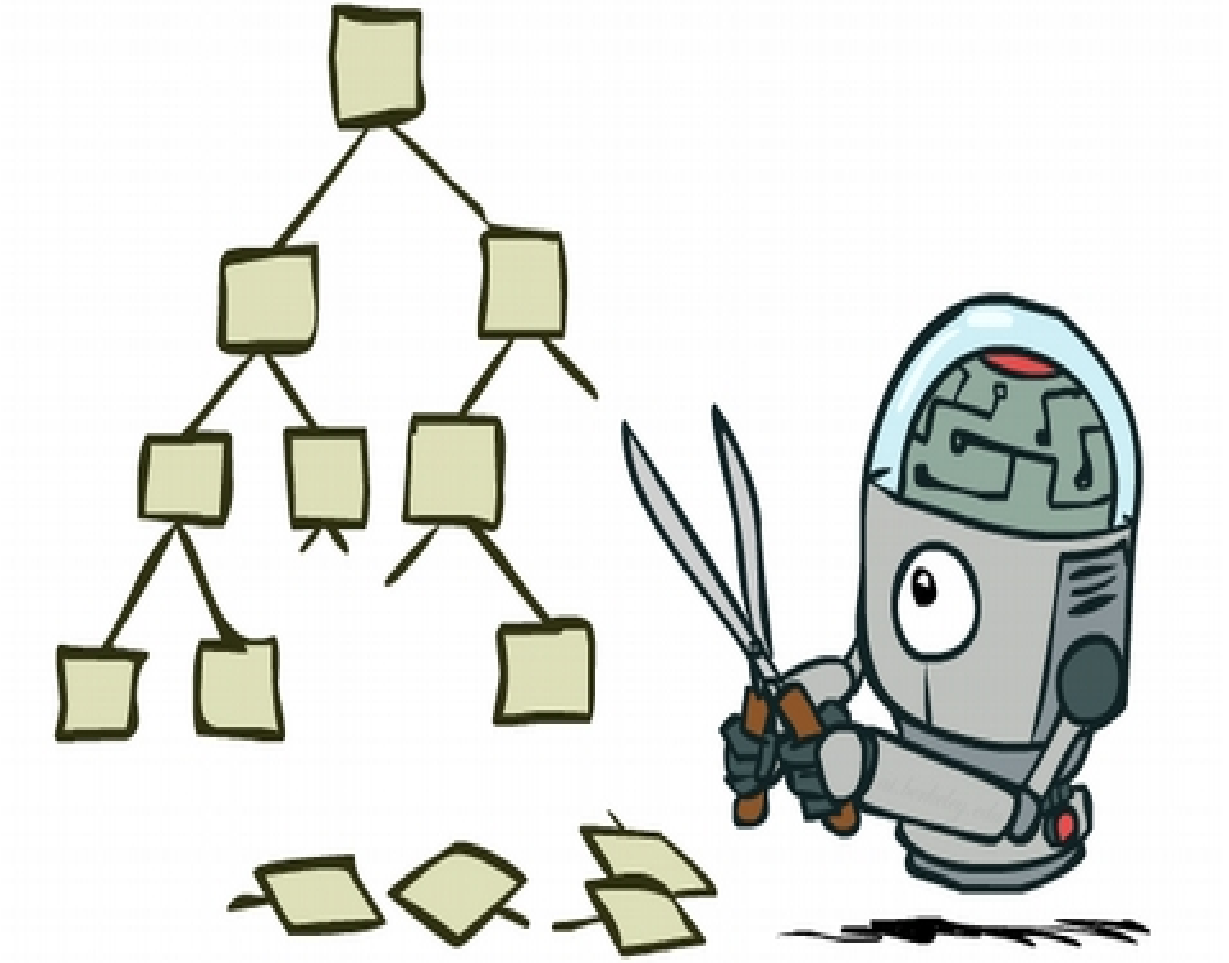
Eficiencia de Minimax

- ¿Cómo de eficiente es minimax?
  - Igual que (exhaustivo) DFS
  - Tiempo: $O(b^m)$
  - Espacio: $O(b^m)$

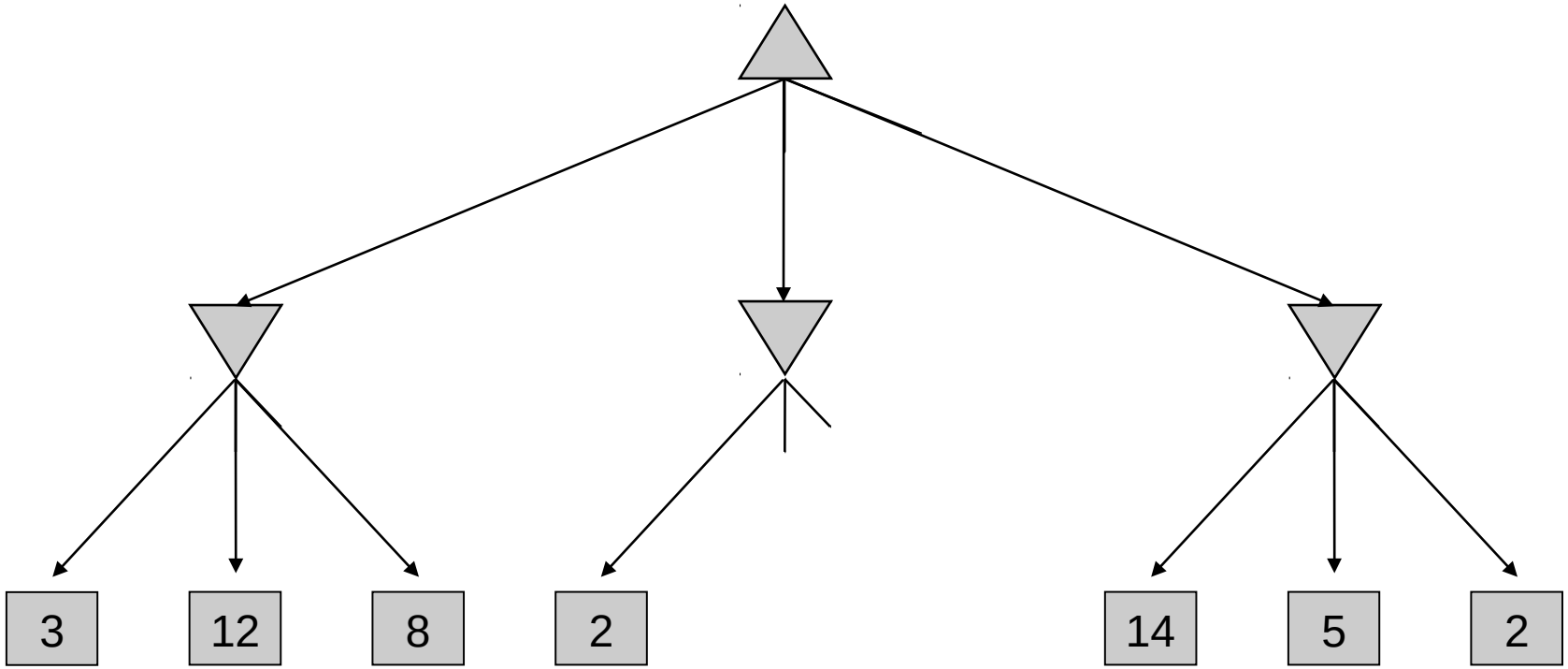
- Ejemplo: para ajedrez, b ~ 35, m ~ 100
  - Una solución exacta es inviable
  - Pero, ¿Tenemos que explorar el árbol entero?



Podado del árbol de juego



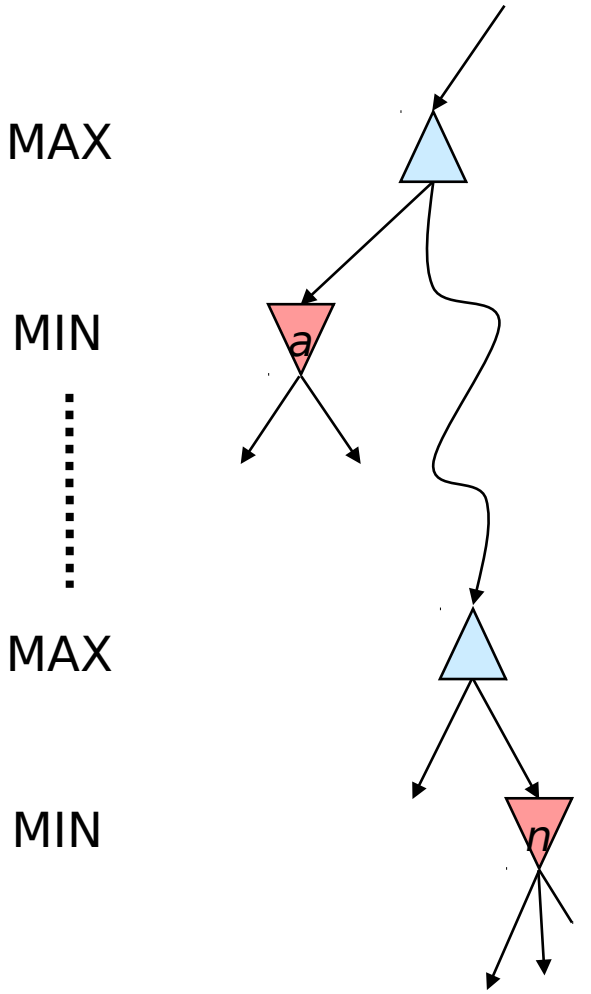
Podado de Minimax



Podado Alpha-Beta

- Configuración General (versión MIN)
  - Estamos calculando un valor-MIN en un nodo ***n***
  - Estamos iterando sobre los hijos de ***n***
  - Al estar minimizando, el valor de ***n*** irá reduciéndose
  - ¿Dónde se usa el valor de ***n***? MAX
  - Sea ***a*** el mejor valor que MAX puede obtener a lo largo del camino actual desde la raíz
  - Si ***n*** es peor que ***a***, MAX lo evitará, por ello podemos evitar el considerar los demás hijos de ***n*** (es suficientemente malo para saber que no se hará esa jugada).

- La versión MAX es simétrica



Implementación de Alfa-Beta

α: la mejor opción de MAX's en el camino a la raíz
β: la mejor opción de MIN's en el camino a la raíz

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

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

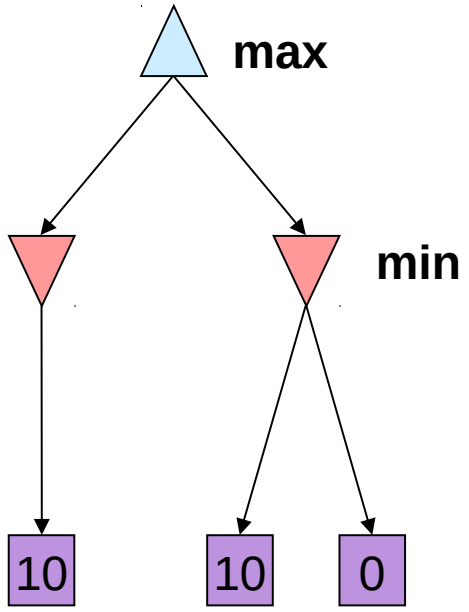
Propiedades de podado Alfa-Beta

- ¡El podado no tiene efecto sobre el valor minimax calculado para la raíz!

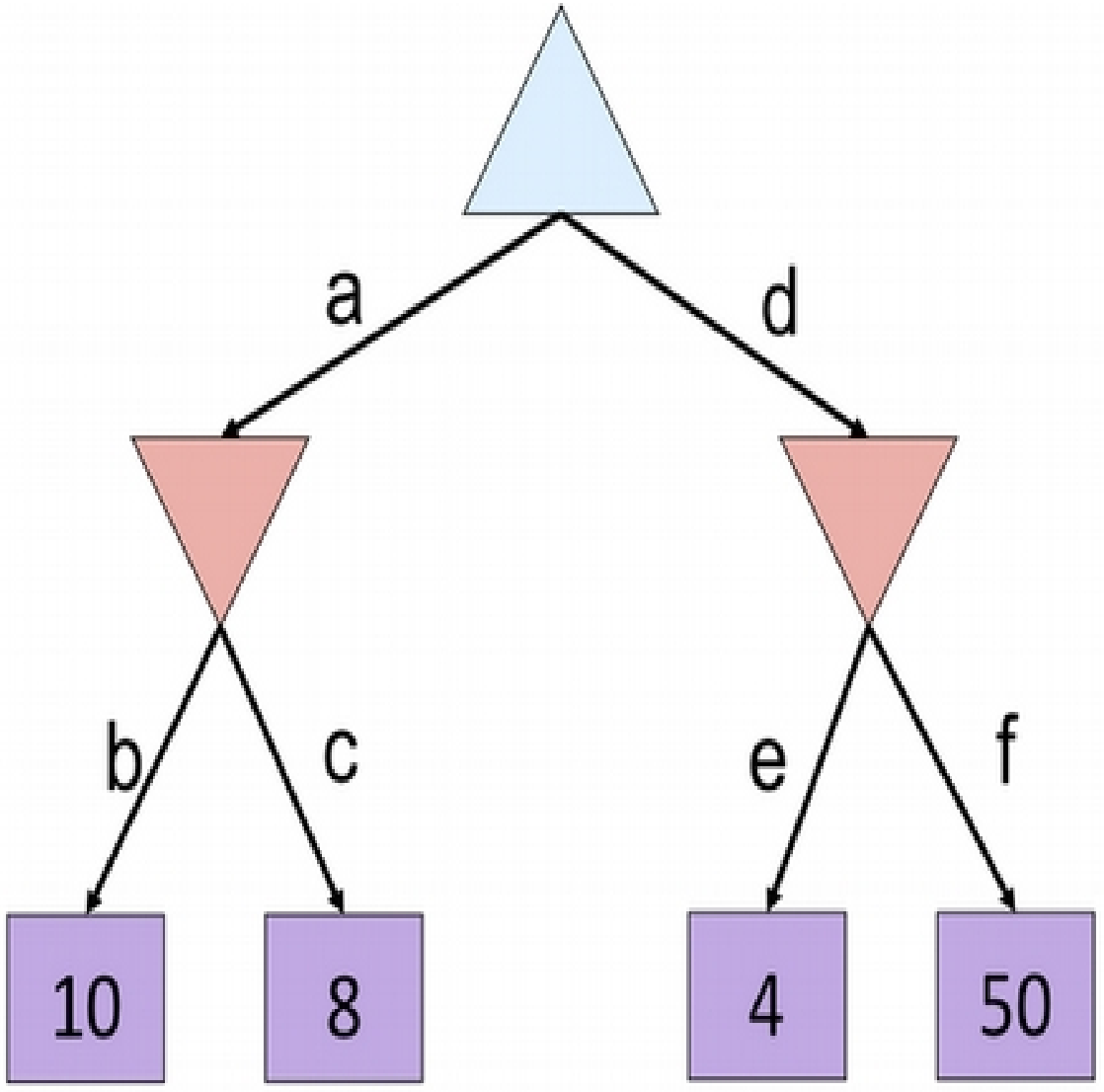
- Los valores de nodos intermedios pueden ser incorrectos
  - Importante: los hijos de la raíz pueden tener un valor incorrecto
  - Por ello, una versión simple no permitirá elegir una acción (hijos de la raíz)

- Una buena ordenación de los hijos mejora la efectividad del podado

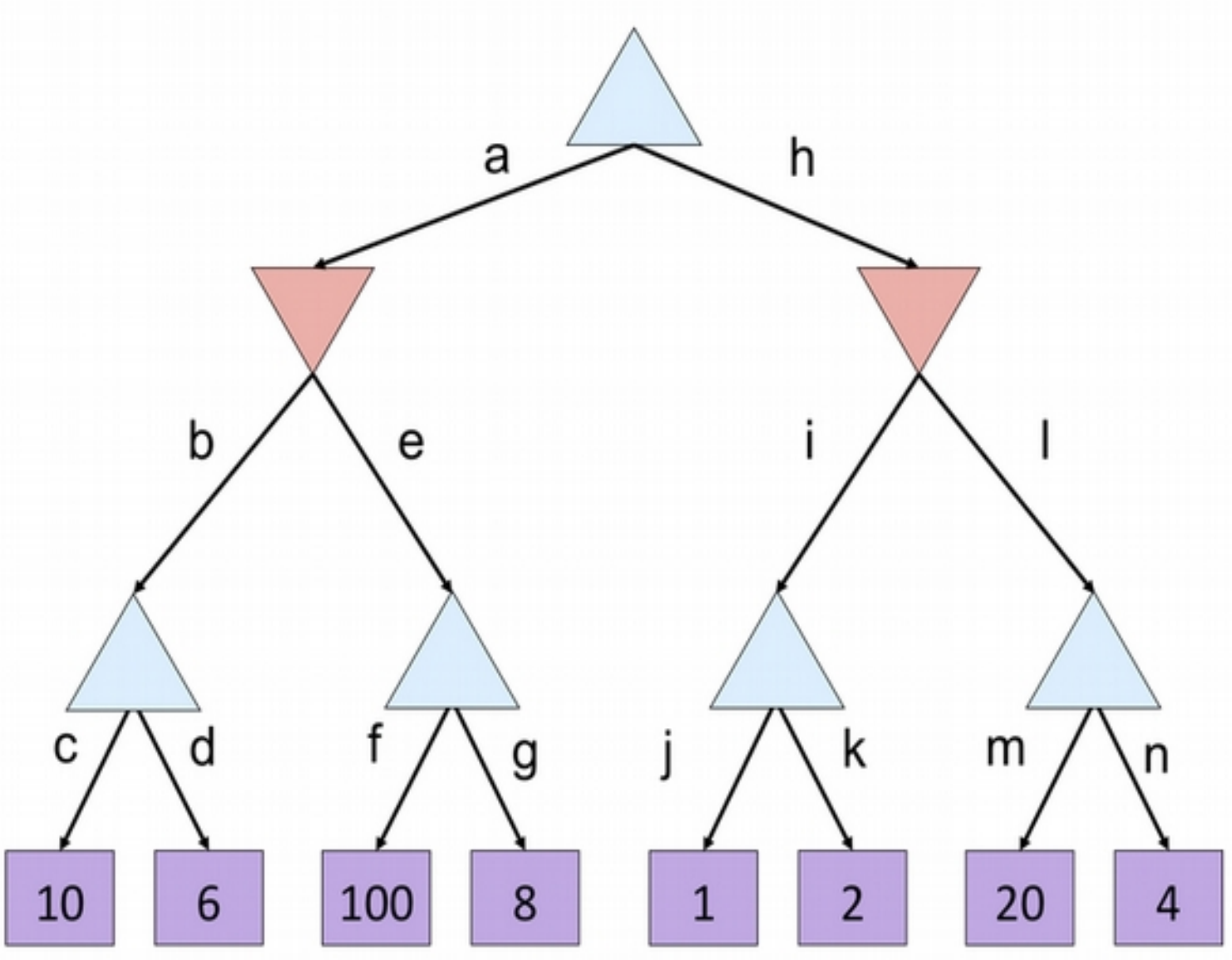
- Con una “ordenación perfecta”:
  - La complejidad en tiempo se reduce a $O(b^{m/2})$
  - ¡Dobla la profundidad que se puede explorar!



Alpha-Beta Quiz



Alpha-Beta Quiz 2

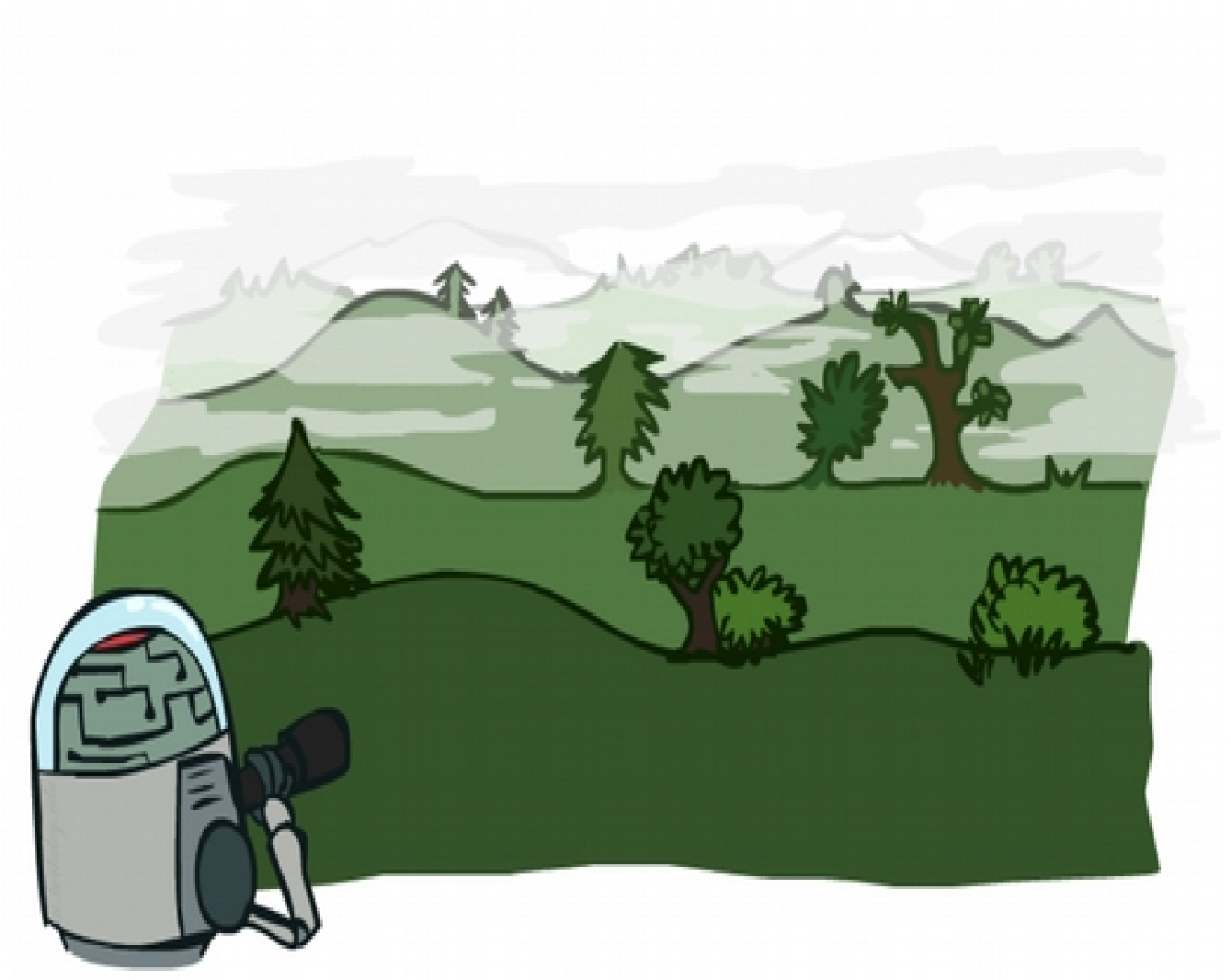


Alpha-Beta Demo

- Demo: minimax game search algorithm with alpha-beta pruning (using html5, canvas, javascript, css)

  - **http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html**

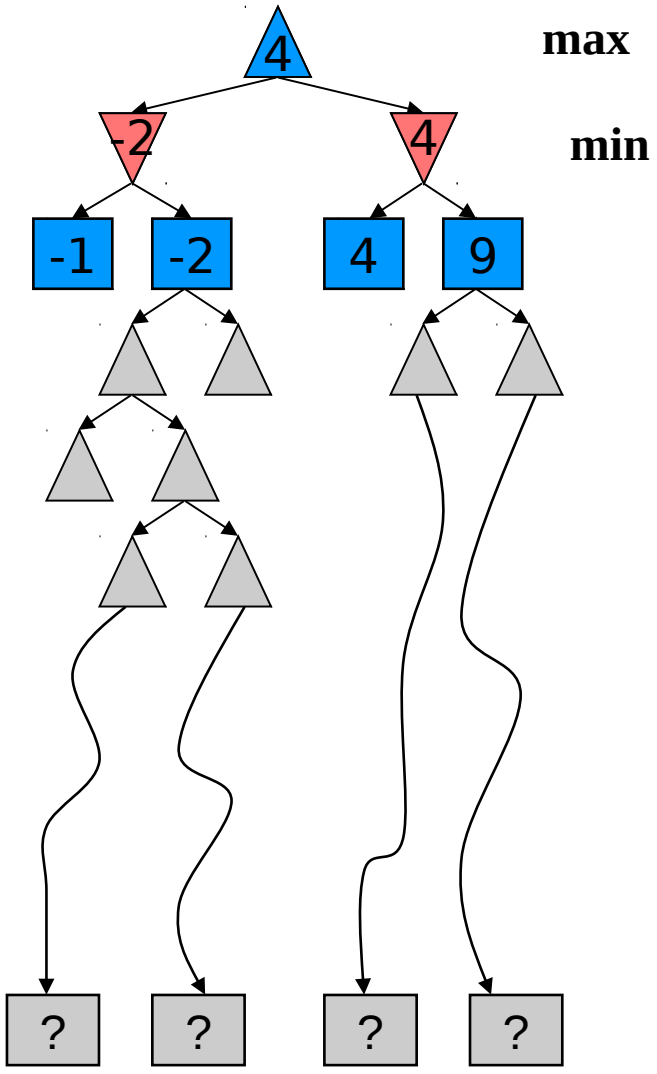
Recursos limitados



Recursos limitados

- Problema: en juegos reales, ¡no podemos buscar hasta las hojas!
- Solución: búsqueda limitada en profundidad (Depth-limited search)
  - Buscar solo una profundidad limitada del árbol
  - Reemplazar las utilidades terminales con una función de evaluación para posiciones no terminales
- Ejemplo:
  - Suponer que tenemos 100 segundos, y podemos explorar 10K nodos / seg
  - Podemos explorar 1M nodos por movimiento
  - α-β llega a profundidad 8 – programa de ajedrez decente

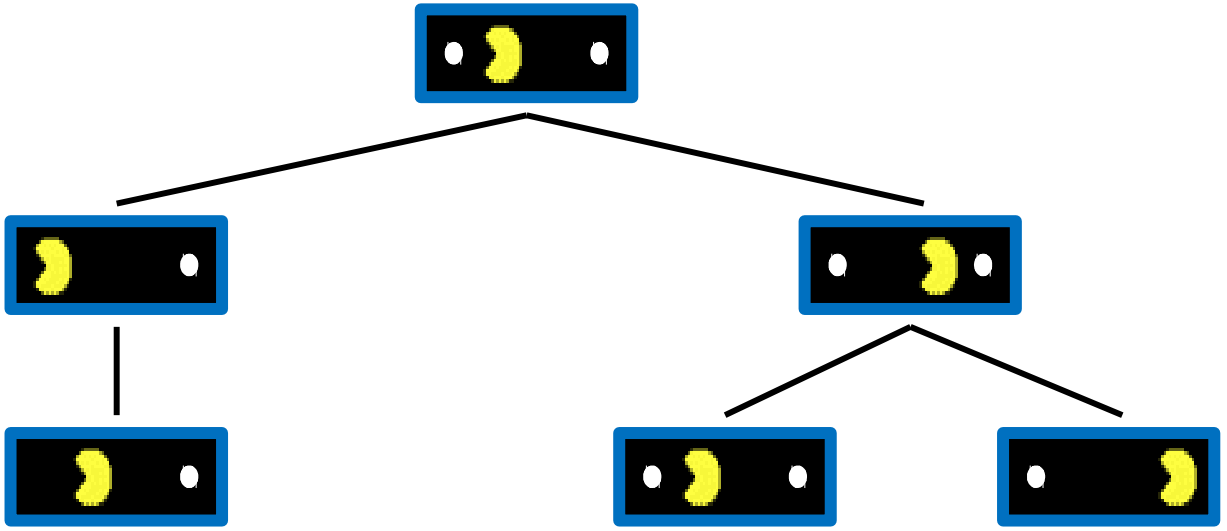
- La garantía de juego óptimo se desvanece
- Usar iterative deepening para tener un anytime algorithm (lo mejor posible dado un tiempo limitado)



Vídeo of Demo Thrashing (d=2)

[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function) (L6D6)]

Por qué Pacman desfallece



- ¡Peligro de agentes que replanifican!
  - Pacman sabe que su puntuación aumentará yendo en las 2 direcciones comiendo un punto (west, east). P. ej.: función de evaluación: 10 por cada punto comido
  - Pero Pacman sabe que su puntuación también subirá si hace (east, west)
  - Después de comer el punto, no hay oportunidad de anotar más puntos (en el horizonte, en este caso, 2 jugadas (profundidad))
  - Por ello, la espera es tan buena como el comer: puede ir east, luego west en la siguiente ronda de planificación

Video Demo Thrashing – Fijado (d=2)

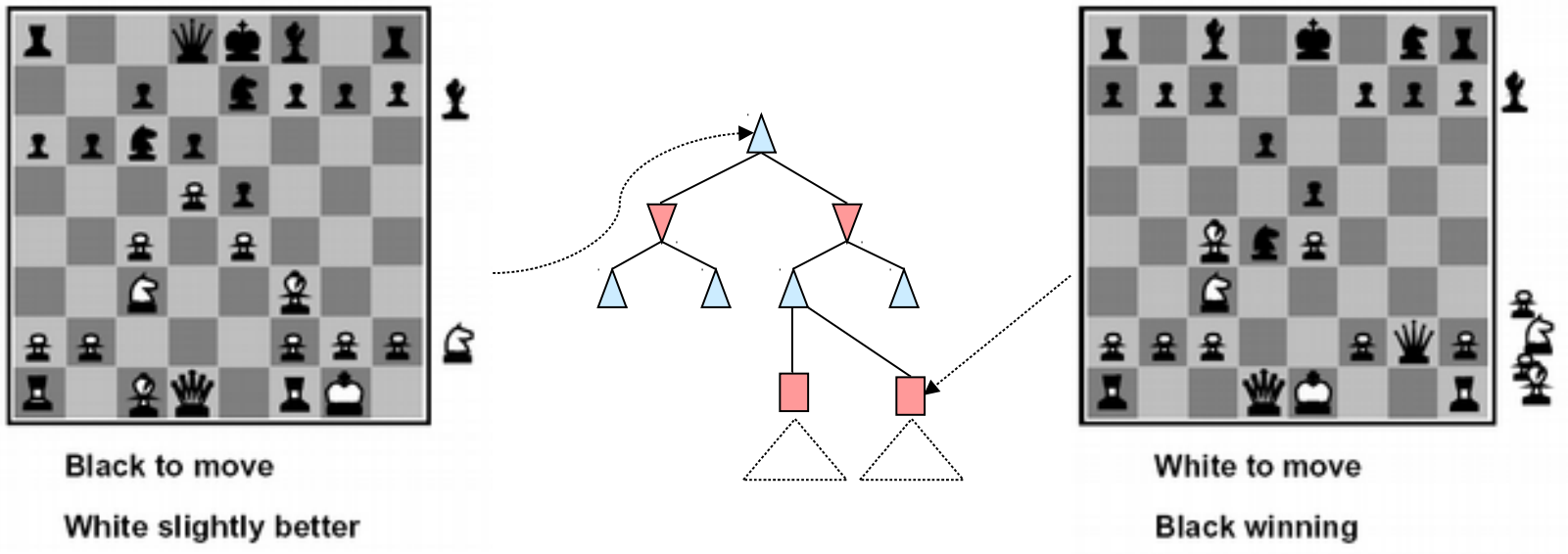
El problema no era el minimax sino la función de evaluación

[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function) (L6D7)]

Funciones de Evaluación

Funciones de Evaluación

- Las funciones de evaluación puntúan no terminales, con búsqueda de profundidad limitada



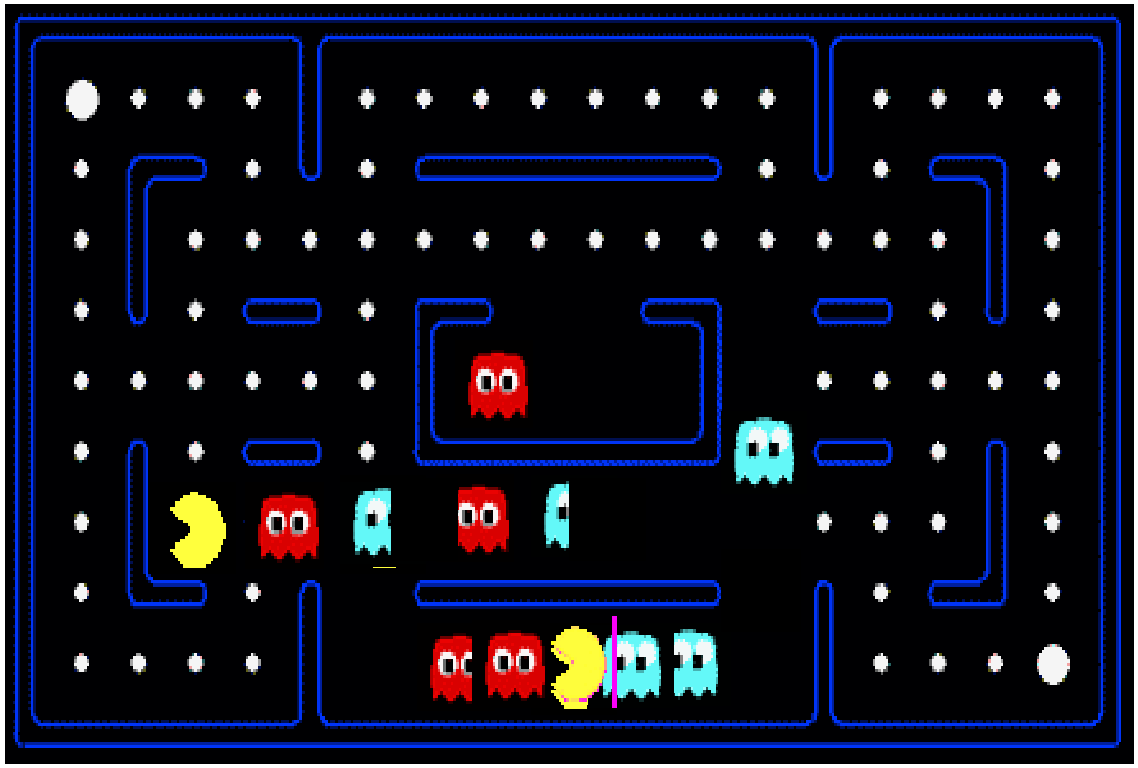
- Función Ideal: devuelve el valor real de minimax en esa posición
- En la práctica: normalmente suma lineal ponderada de características (features)

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

- Por ejemplo:

$f_1(s)$ = (cantidad de reinas blancas – cantidad reinas negras), etc.

Evaluación para Pacman



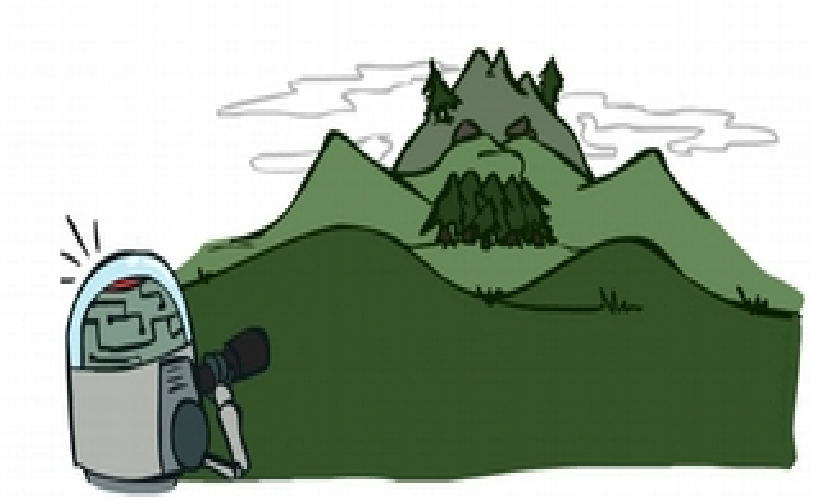
[Demo: thrashing d=2, thrashing d=2 (fixed evaluation function), smart ghosts coordinate (L6D6,7,8,10)]

Vídeo Demo Fantasmas inteligentes (Coordinación)

Vídeo Demo Fantasmas inteligentes (Coordinación) – con Zoom

La profundidad importa

- Las funciones de evaluación son siempre imperfectas
- Cuanto más profundamente en el árbol probemos la función de evaluación, es menos importante la calidad de la función de evaluación
- Un ejemplo interesante de compensación (tradeoff) entre complejidad de la función de evaluación y complejidad de cálculo

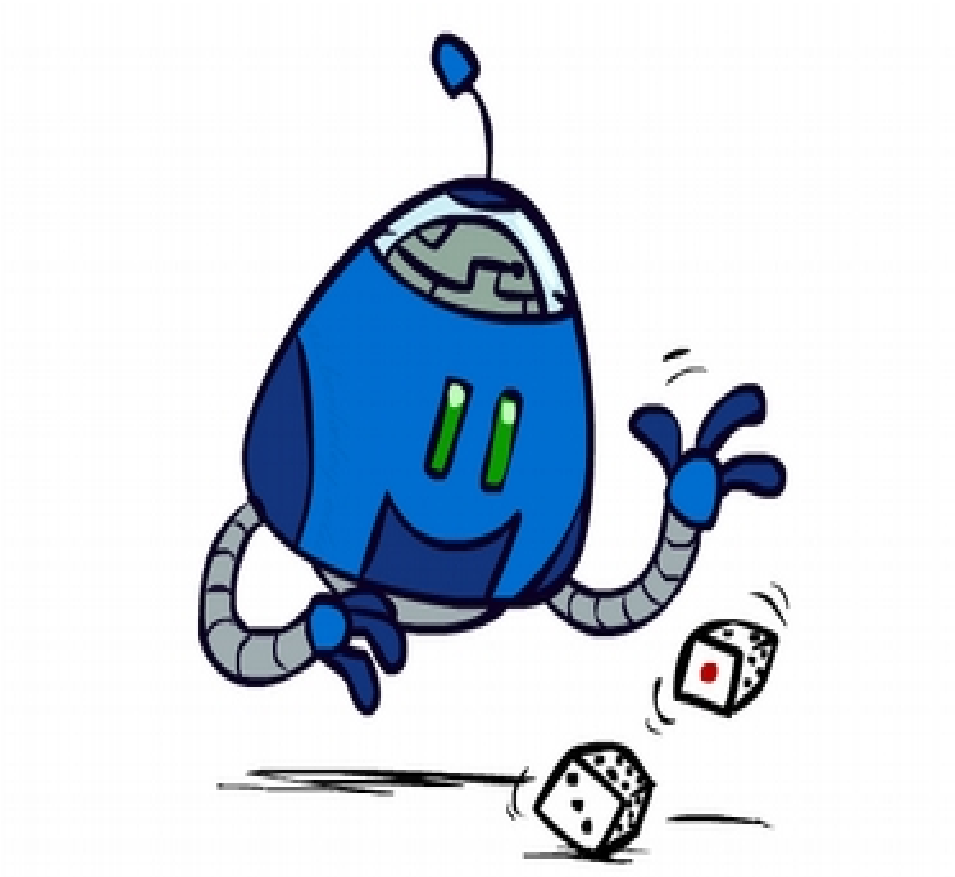


[Demo: depth limited (L6D4, L6D5)]

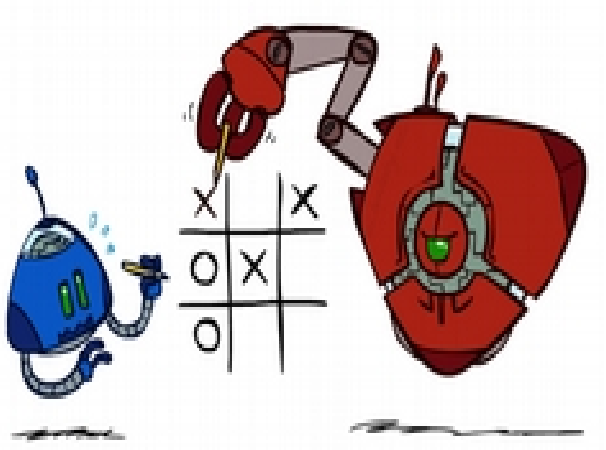
Vídeo Demo Profundidad limitada (2)

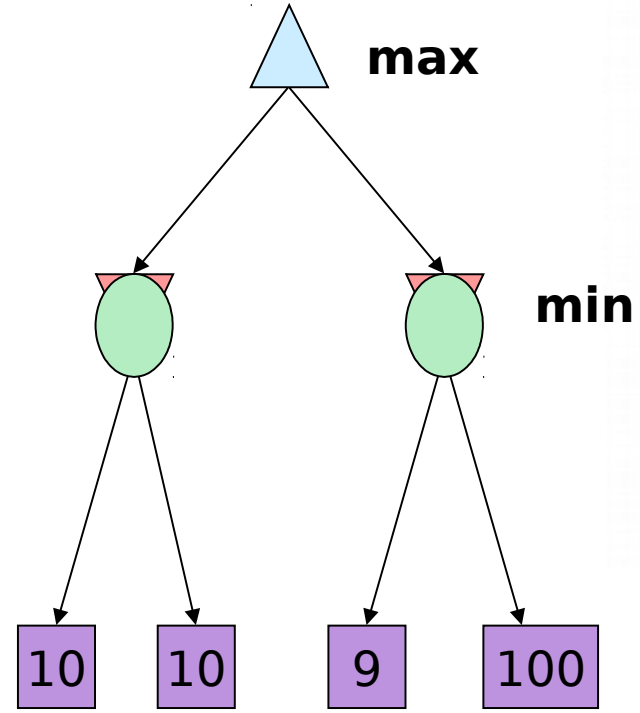
Vídeo Demo Profundidad limitada (10)

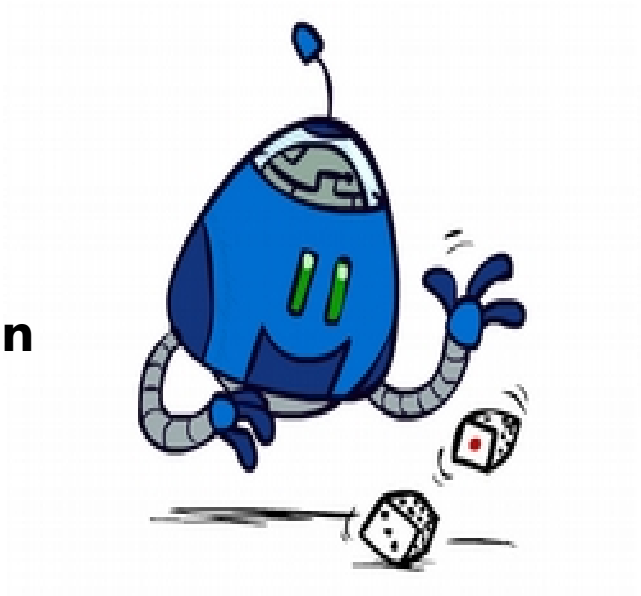
Resultados inciertos



Caso peor vs. caso medio



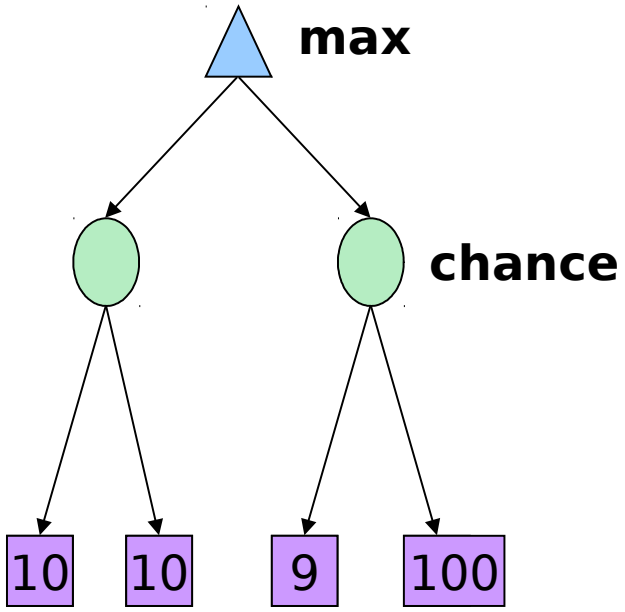




Idea: los resultados inciertos están controlados por el azar, no por el adversario!

Búsqueda Expectimax

- ¿Por qué podemos no conocer el resultado de una acción?
  - Aleatoriedad explícita: echar los dados
  - Oponentes impredecibles: los fantasmas responden aleatoriamente
  - Las acciones pueden fallar: al mover un robot, las ruedas pueden patinar
- Los valores deberían reflejar resultados medios (expectimax), no resultados en el caso peor (minimax)
- Búsqueda Expectimax: calcular la puntuación media con un juego óptimo
  - Nodos Max como en búsqueda minimax
  - Los nodos aleatorios son como los nodos min pero el resultado es incierto
  - Se calcularán las utilidades esperadas
  - P. ej. Tomar la media ponderada (expectativa) de los hijos



Pseudocódigo de Expectimax

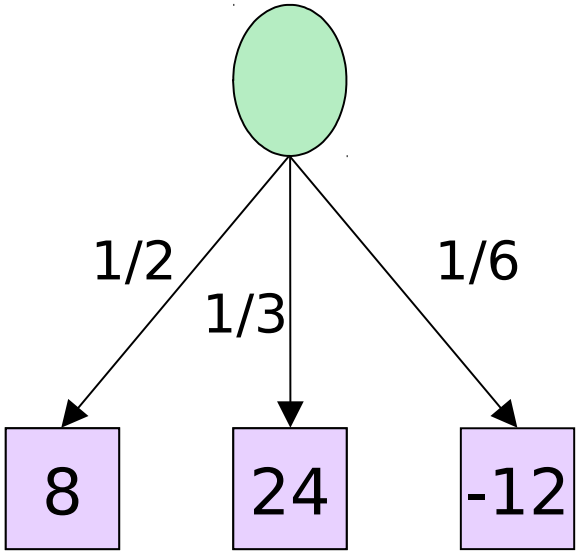
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

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

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

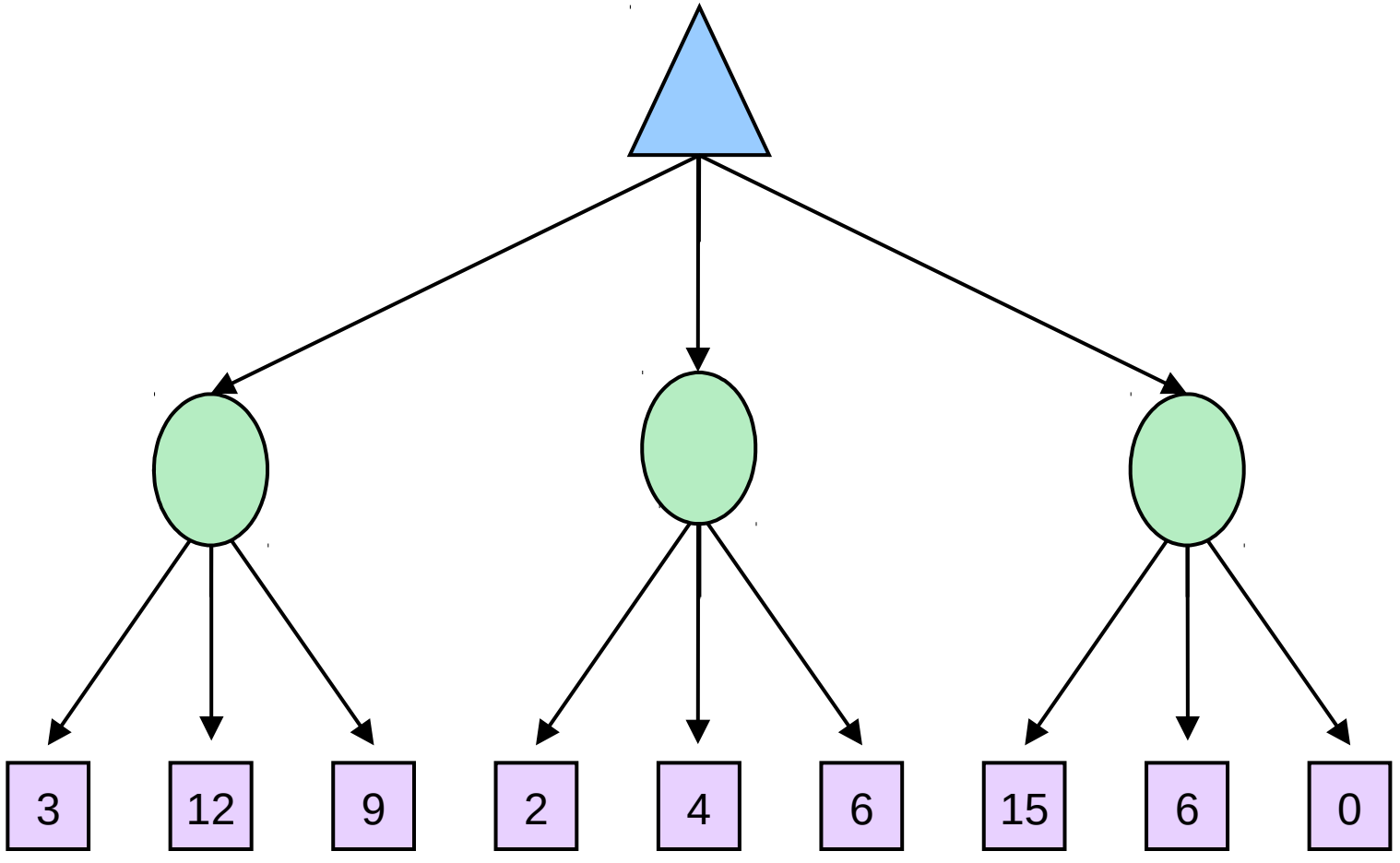
Pseudocódigo de Expectimax

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```



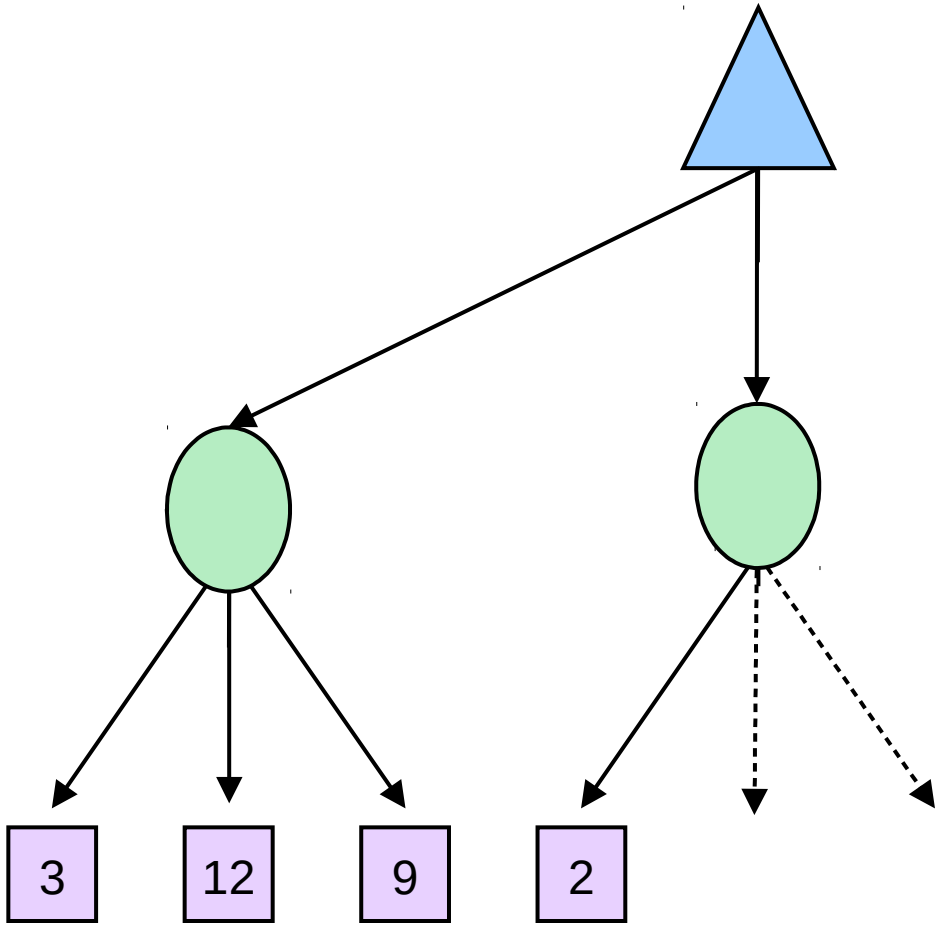
$$v = (1/2)\,(8) + (1/3)\,(24) + (1/6)\,(-12) = 10$$

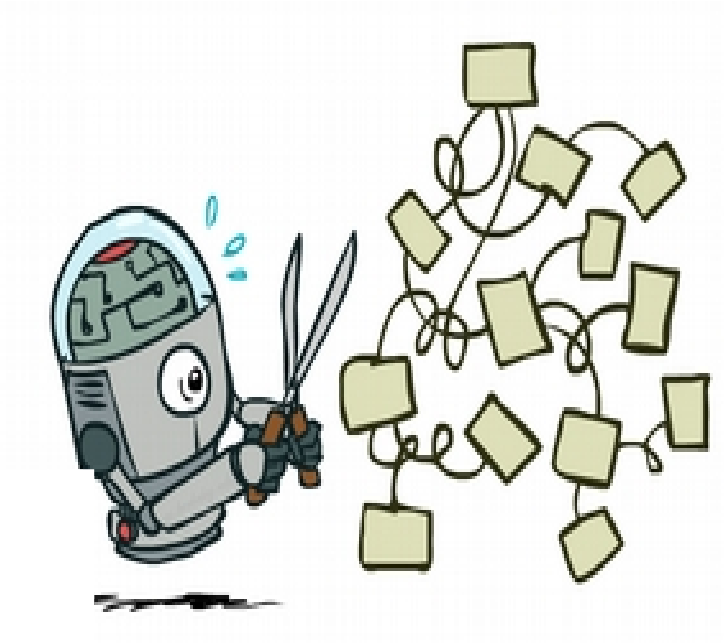
Ejemplo de Expectimax



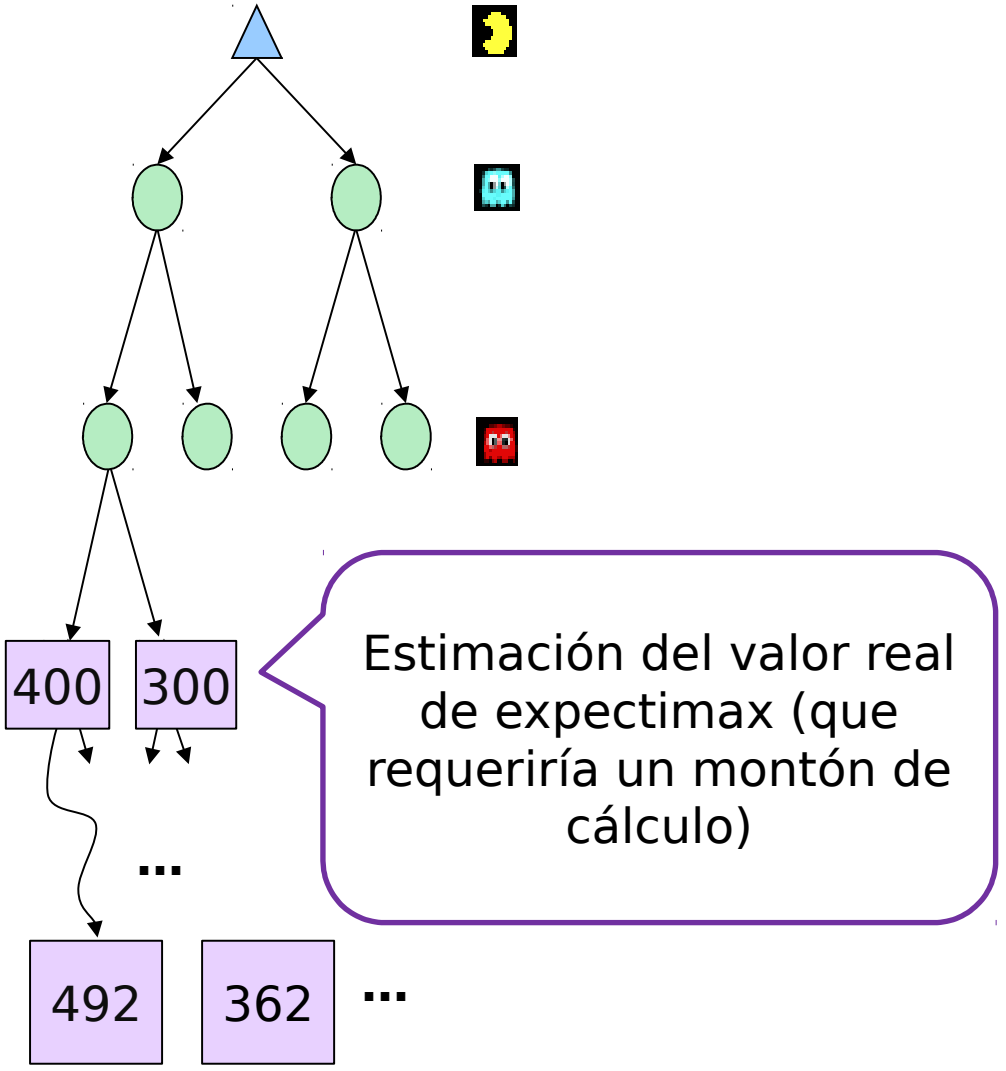
Vídeo Demo Expectimax

¿Poda de Expectimax?

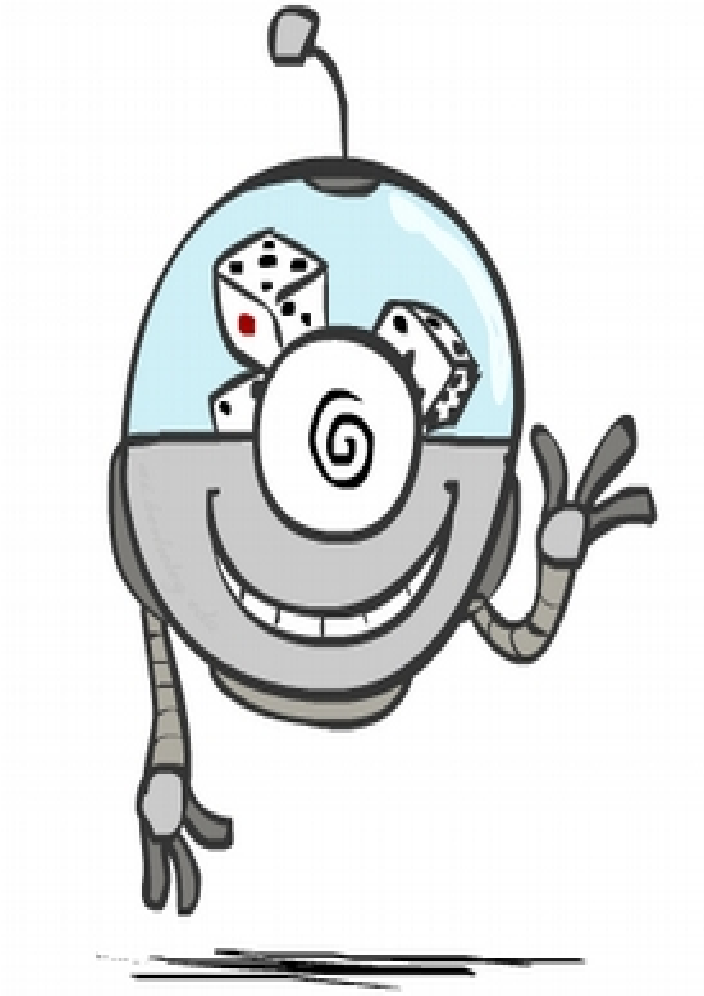




Expectimax con profundida limitada

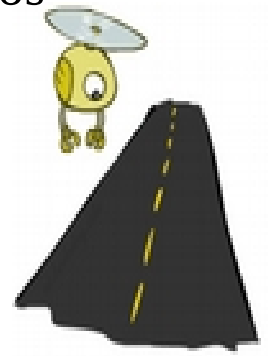


Probabilidades



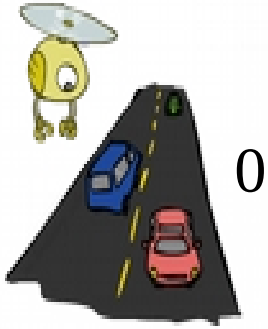
Recuerdo: Probabilidades

- Una **variable aleatoria (random)** representa un evento cuyo resultado es desconocido
- Una **distribución de probabilidad** es una asignación de pesos a resultados



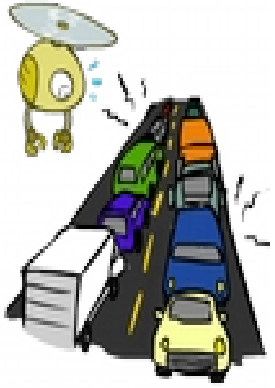
0.25

- Ejemplo: Tráfico en la autovía
  - Variable aleatoria: T = hay tráfico o no
  - Valores: T en {nada, ligero, mucho}
  - Distribución: P(T=nada) = 0.25, P(T=ligero) = 0.50, P(T=mucho) = 0.25



0.50

- Algunas leyes de probabilidad:
  - Las Probabilidades son siempre no negativas
  - La suma de Probabilidades sobre todos los valores posibles suma uno

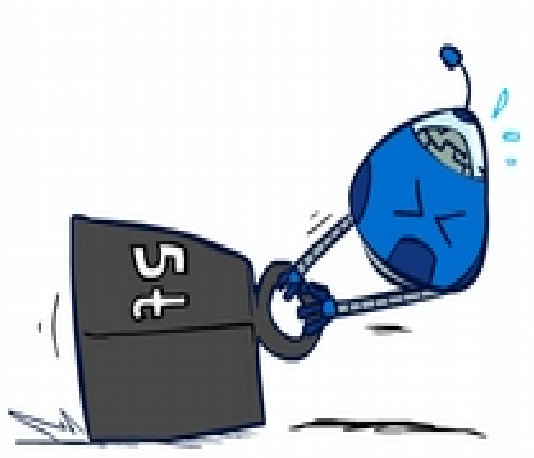


0.25

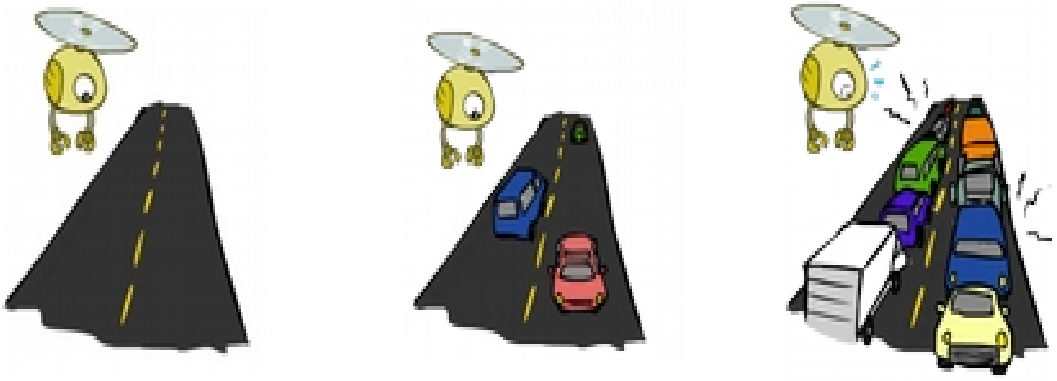
Recuerdo: Expectativas

- El valor esperado de una función de una variable aleatoria es la media, ponderada por la distribución de probabilidad de los resultados

- Ejemplo: ¿Cuánto tardaré en llegar al aeropuerto?



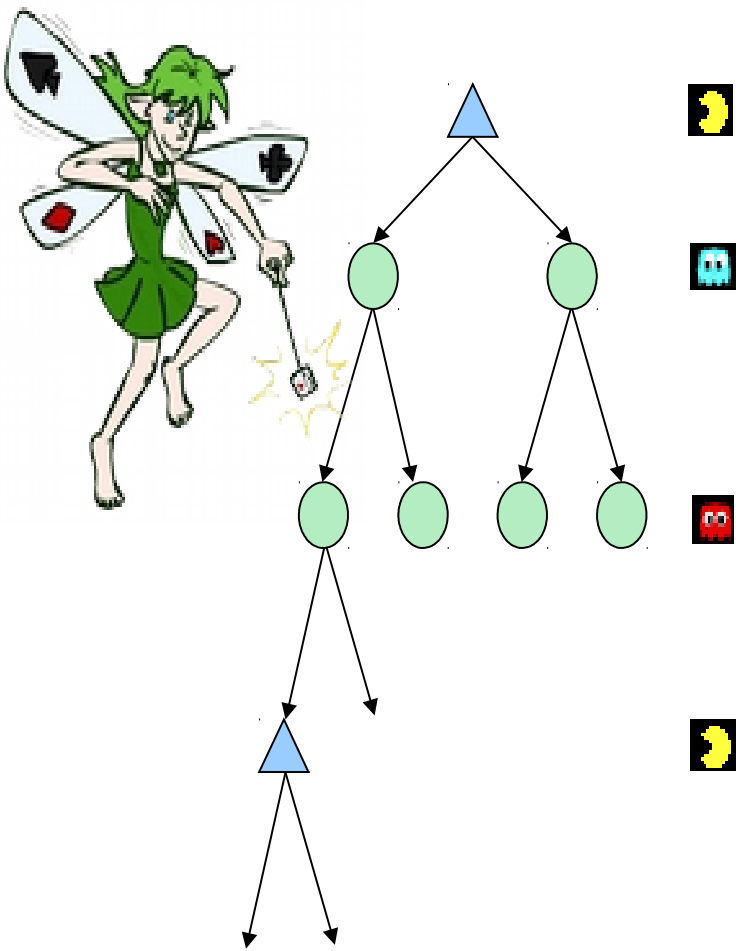
Tiempo:	20 min		30 min		60 min		
	x	+	x	+	x	→	35 min
Probabilidad:	0.25		0.50		0.25		



¿Qué probabilidades usamos?

- En la búsqueda expectimax, tenemos un modelo probabilístico de cómo el oponente (o entorno) se comportará en cualquier estado
  - El Modelo podría ser una distribución uniforme (echar los dados)
  - El Modelo podría ser sofisticado y requerir un montón de computación
  - Tendremos un nodo aleatorio por cada situación fuera de nuestro control: oponente o entorno
  - ¡El modelo podría decir qué acciones adversariales son probables!

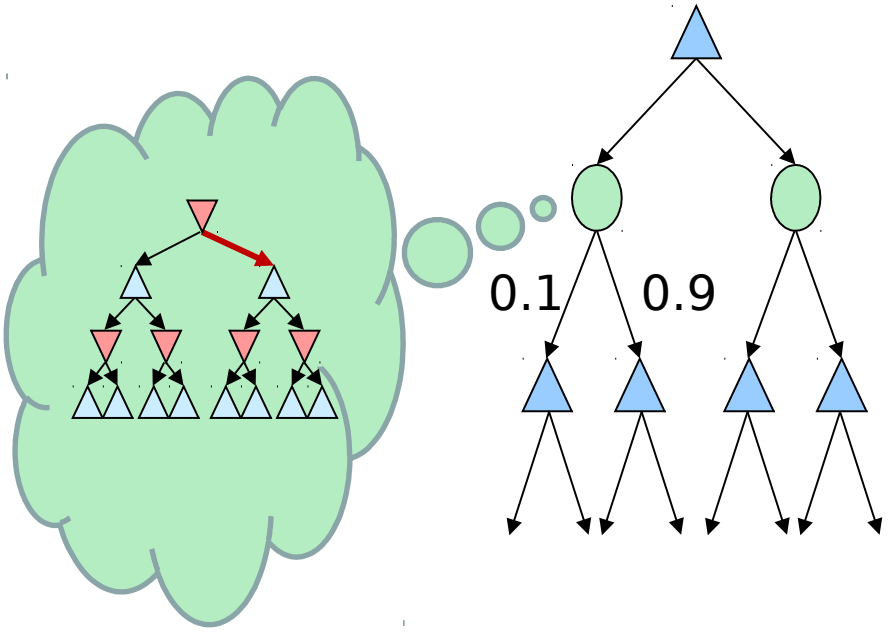
- Por ahora, asumiremos que cada nodo viene mágicamente con probabilidades que especifican la distribución respecto a sus valores



¡Tener una suposición probabilística sobre la acción de otro agente no quiere decir que ese agente esté echando una moneda!

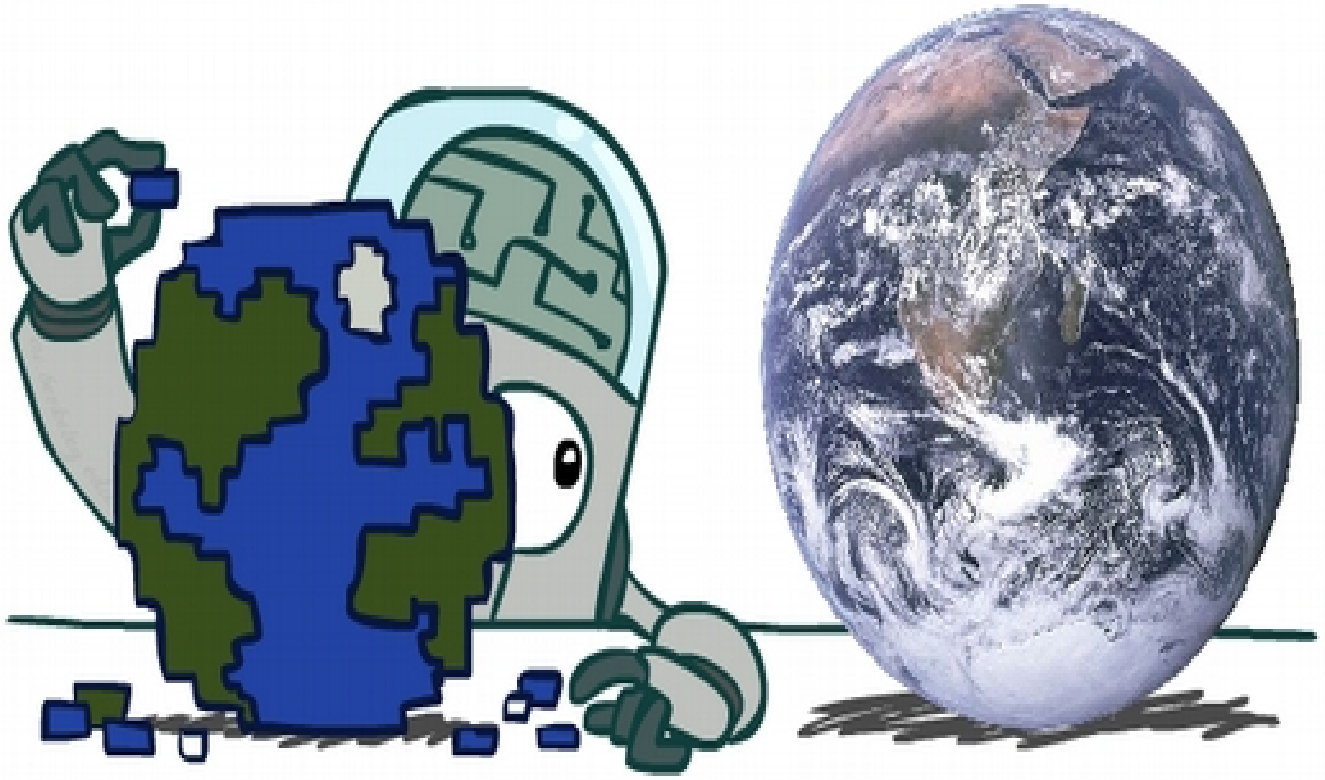
Quiz: Probabilidades informadas

- Supongamos que nuestro oponente está ejecutando un minimax de profundidad 2, usando ese resultado un 80% de las veces, y moviéndose aleatoriamente en otro caso
- Pregunta: ¿Qué tipo de búsqueda en árbol usaríamos?



- Respuesta: ¡Expectimax!
  - Para estimar las probabilidades de CADA nodo aleatorio, tendríamos que ejecutar una simulación de nuestro oponente
  - Esto nos puede llevar rápidamente a ineficiencia (tiempo)
  - Peor si tenemos que simular a nuestro oponente simulándonos a nosotros ...
  - … excepto para minimax, que tiene la (buena) propiedad de que todo se junta en un árbol de juegos

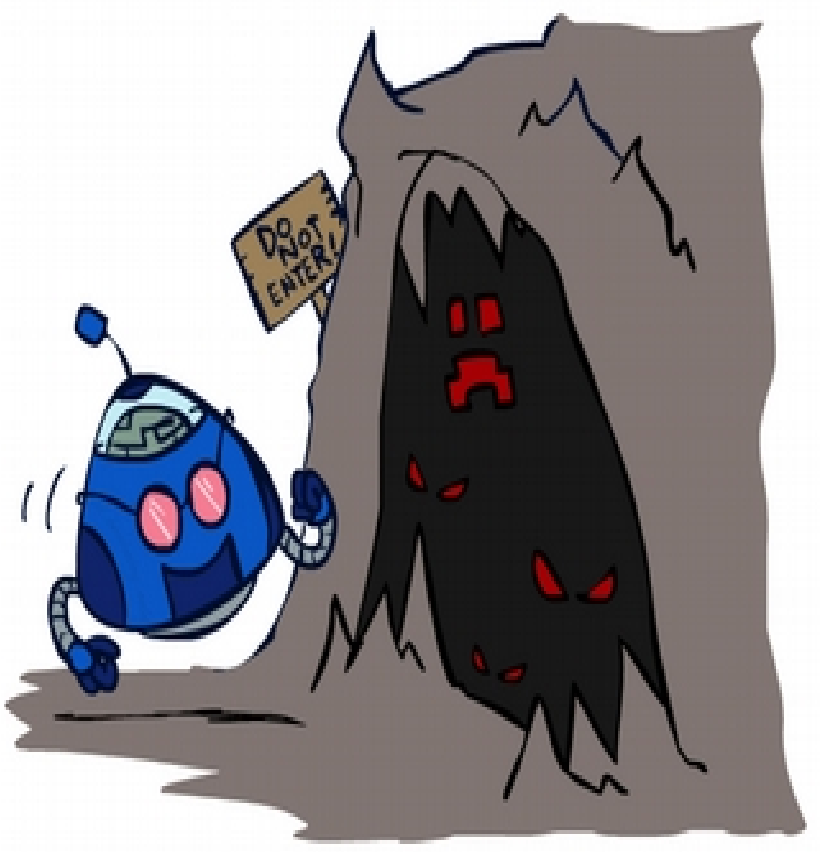
Modelando Asunciones



Los peligros del optimismo y el pesimismo

Optimismo peligroso

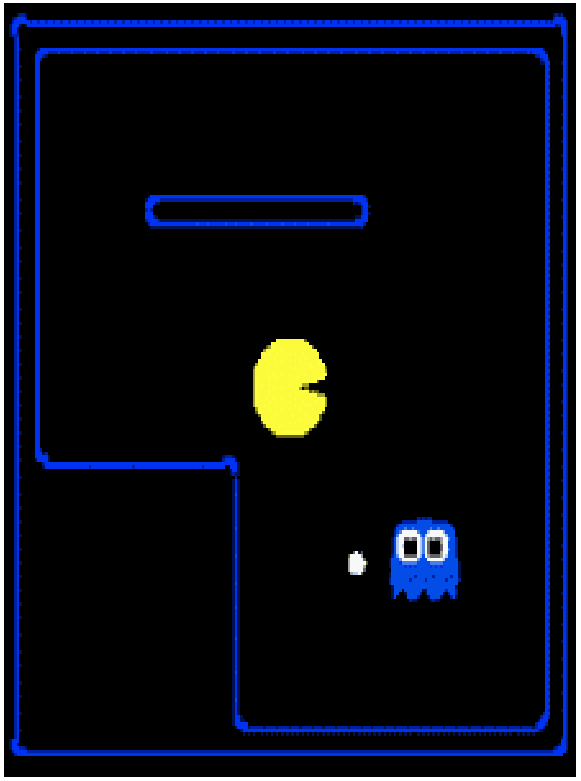
Asumiendo azar cuando el mundo es adversarial



Pesimismo peligroso

Asumiendo el caso peor cuando es poco probable

Asunciones vs. Realidad



	Fantasma Adversarial	Fantasma Aleatorio
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expecti max Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

Resultados jugando 5 juegos

- Pacman usó búsqueda de profundidad 4 con una función de evaluación que evita problemas
- El fantasma usó búsqueda de profundidad 2 con una función de evaluación que busca a Pacman

[Demos: world assumptions (L7D3,4,5,6)]

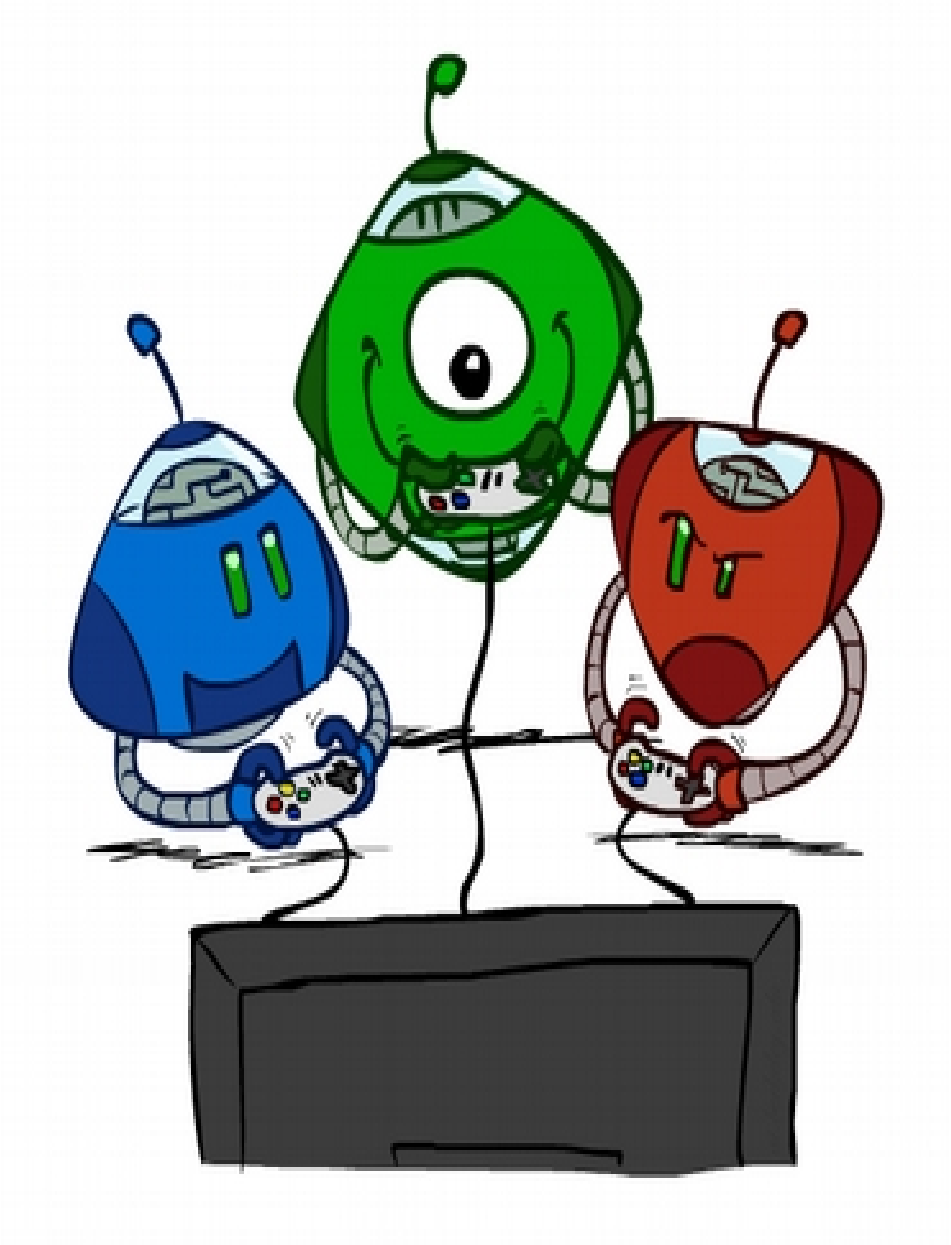
Vídeo Demo Asunciones sobre el mundo Fantasma aleatorio – Pacman Expectimax

Vídeo Demo Asunciones sobre el mundo Fantasma Adversarial – Pacman Minimax

Vídeo Demo Asunciones sobre el mundo Fantasma Adversarial – Pacman Expectimax

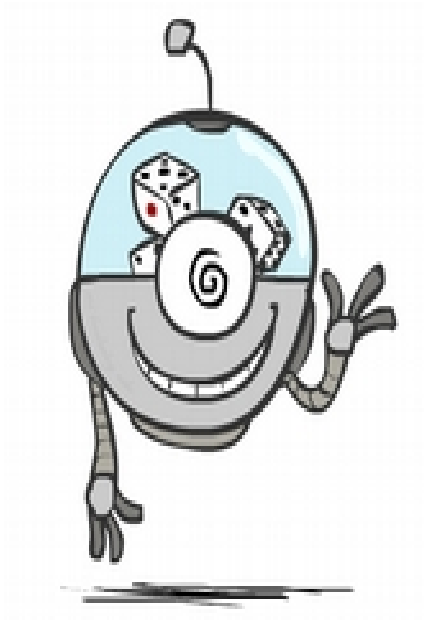
Vídeo Demo Asunciones sobre el mundo Fantasma aleatorio – Pacman Minimax

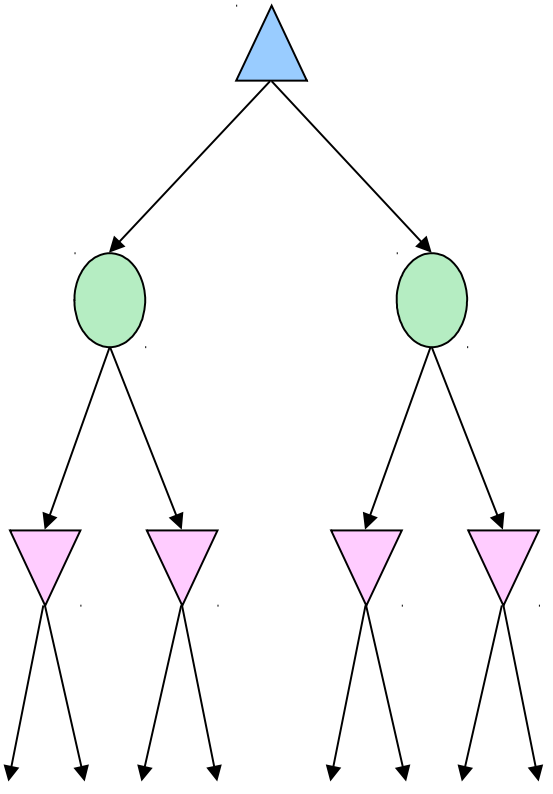
Otros tipos de juegos



Tipos de nivel mixtos

- P. ej. Backgammon
- Expectiminimax
  - El entorno es un “agente aleatorio” extra que mueve después de cada agente min/max
  - Cada nodo calcula la combinación apropiada para sus hijos





Ejemplo: Backgammon

- Al echar los dados se incrementa b: 21 resultados posibles con 2 dados
  - Backgammon ~ 20 movimientos legales
  - Profundidad 2 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$

- Al aumentar la profundidad, la probabilidad de llegar a un nodo disminuye
  - Por ello disminuye la utilidad de la búsqueda
  - Por ello limitar la profundidad es menos dañino
  - Pero el podado es más complicado …

- AI histórica: TDGammon usa búsqueda de profundidad 2 search + muy buena función de evaluación + reinforcement learning: nivel de juego de campeón mundial

- ¡Primer campeón mundial automático en un juego!

Image: Wikipedia

Utilidades Multi-agente

- ¿Qué pasa si el juego no es de suma cero, o hay varios jugadores?

- Generalización de minimax:
  - Los terminales tienen tuplas de utilidades
  - Los valores de los nodos son también
  - tuplas de utilidades
  - Cada jugador maximiza su componente
  - Puede dar lugar a competición y
  - cooperación dinámicamente