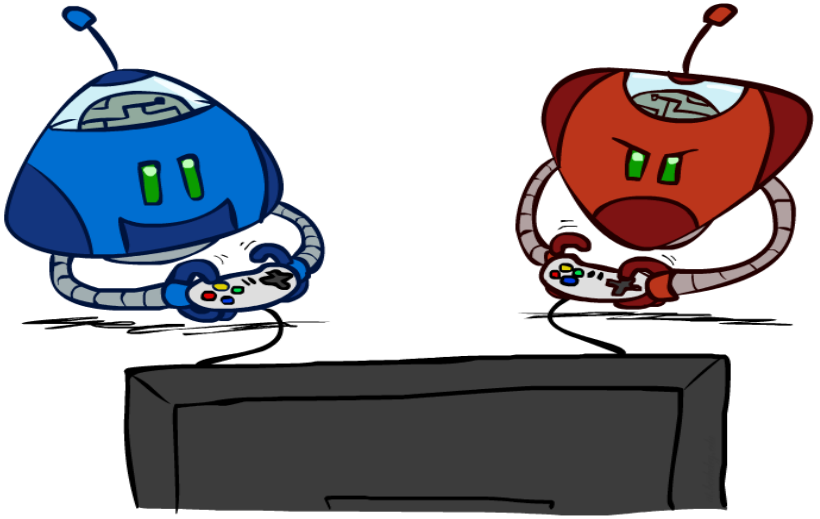


Inteligencia Artificial

Búsqueda adversarial y Árboles de juegos

ROBOTIKA & SISTEMA AUTONOMOEN IKERKETA TALDEA

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

Índice

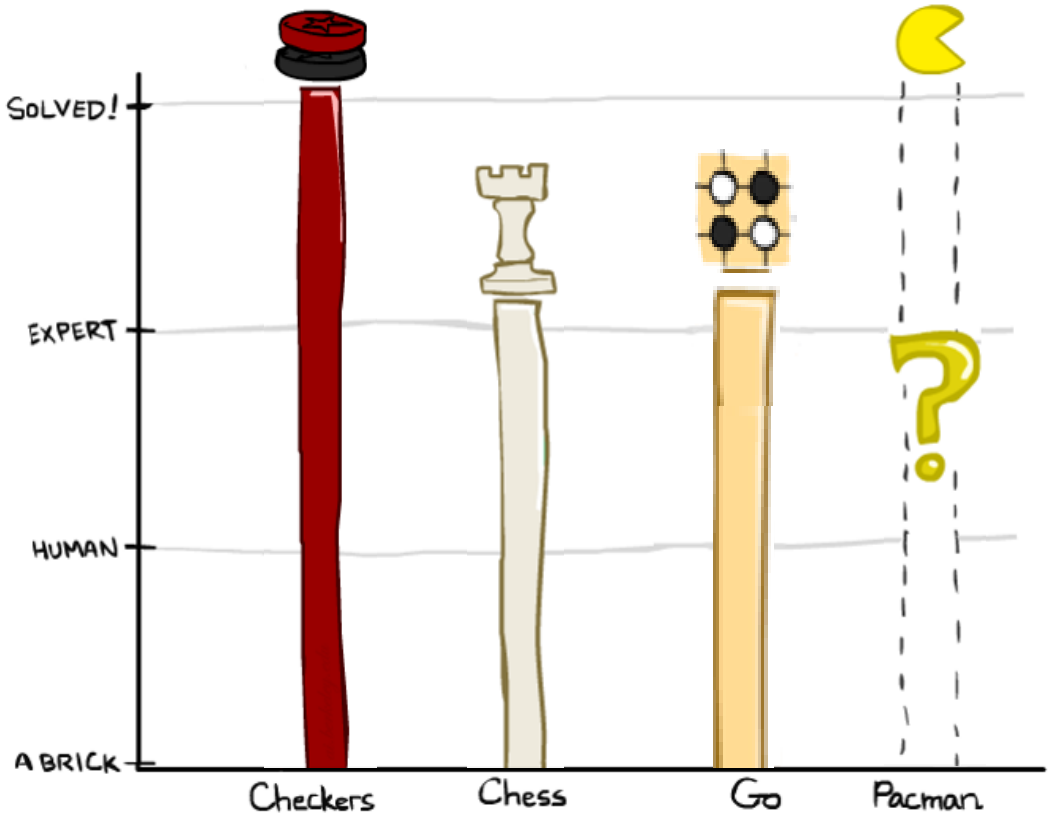
Estado del arte en algunos juegos

- **Damas 1989 - 2007**: Primer campeón automático (Chinook 1990 campeonato mundial) acabó con el reinado de 40 años del campeón Marion Tinsley. En ¿2007?: **¡Damas resueltas!**

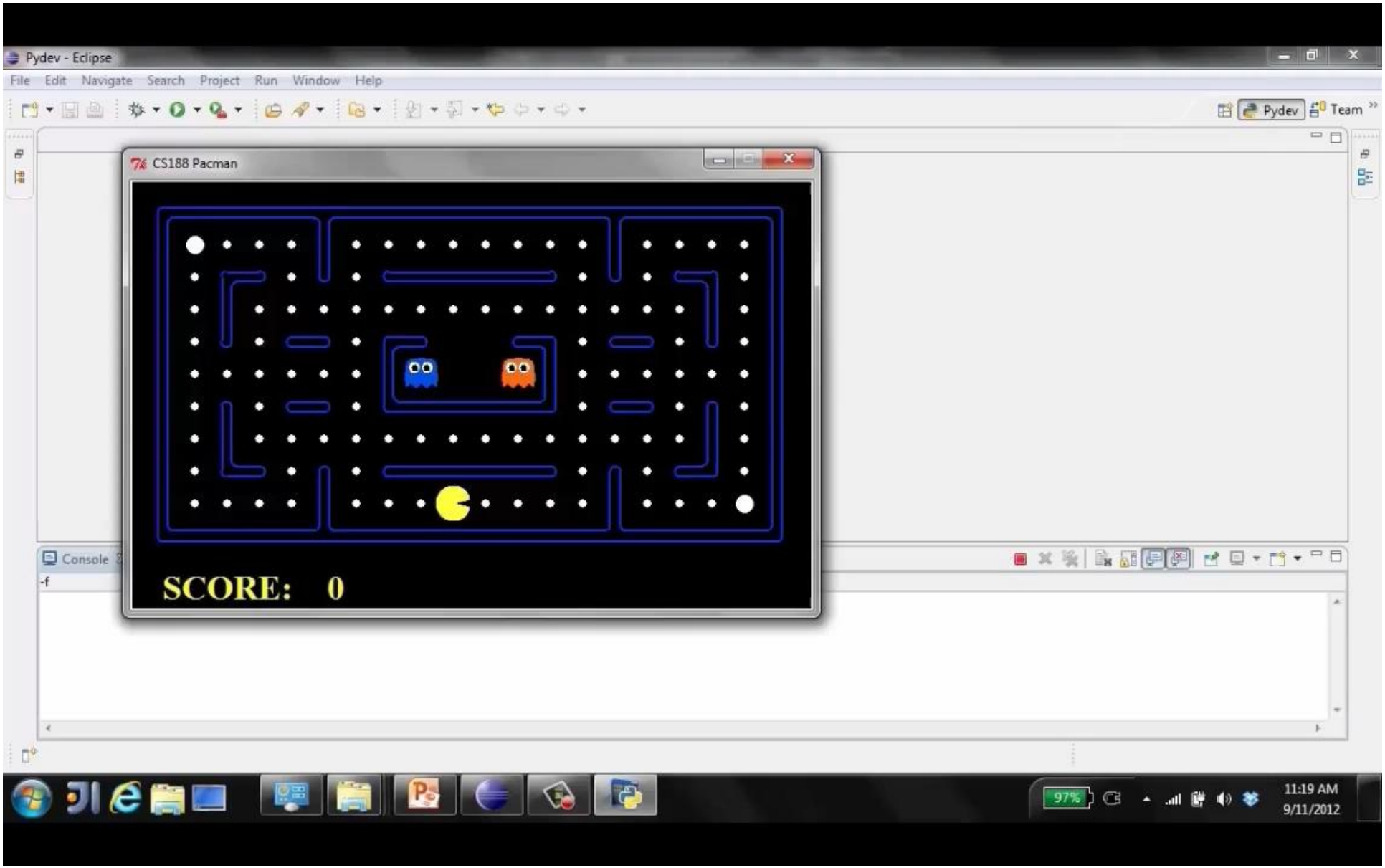
- **Ajedrez 1997**: Deep Blue derrota al campeón Gary Kasparov en un torneo de 5 partidas. Deep Blue examinó 200M posiciones por segundo. Los programas actuales son todavía mejores.

- **Go 2016**: Los campeones humanos empiezan a ser superados por las máquinas **AlphaGO**. En GO, **b > 300 ramificaciones**! Los programas clásicos usan bases de conocimiento de patrones, pero recientes avances usan métodos de expansión Monte Carlo, Redes Neuronales Profundas (**AlphaZero? …**)

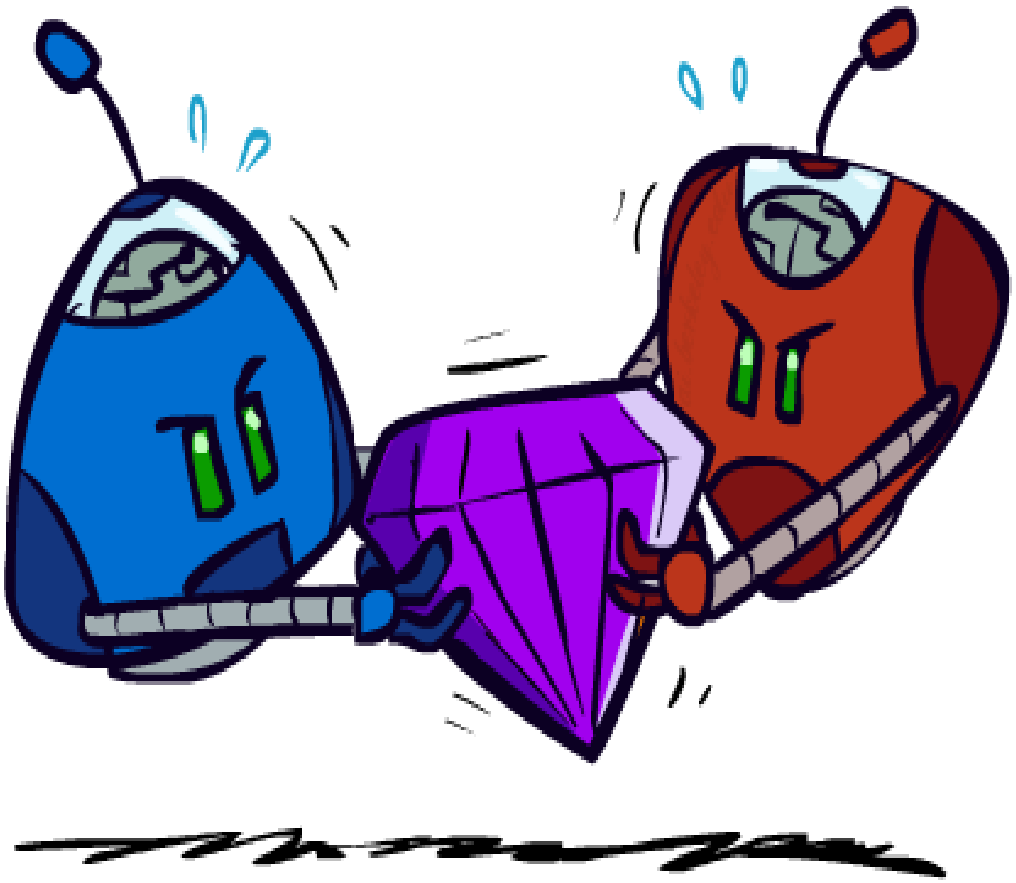
- **Pacman**



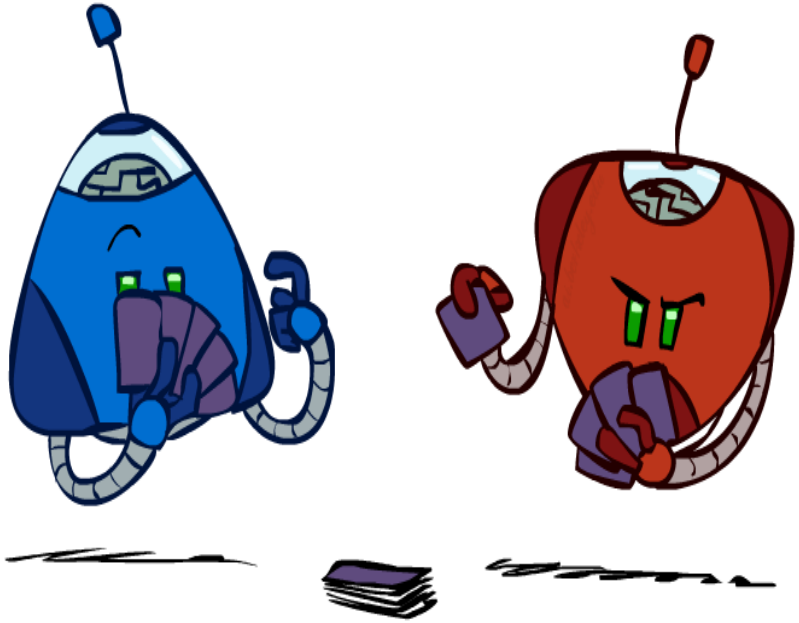
Video Demo Mystery Pacman



Juegos Adversariales



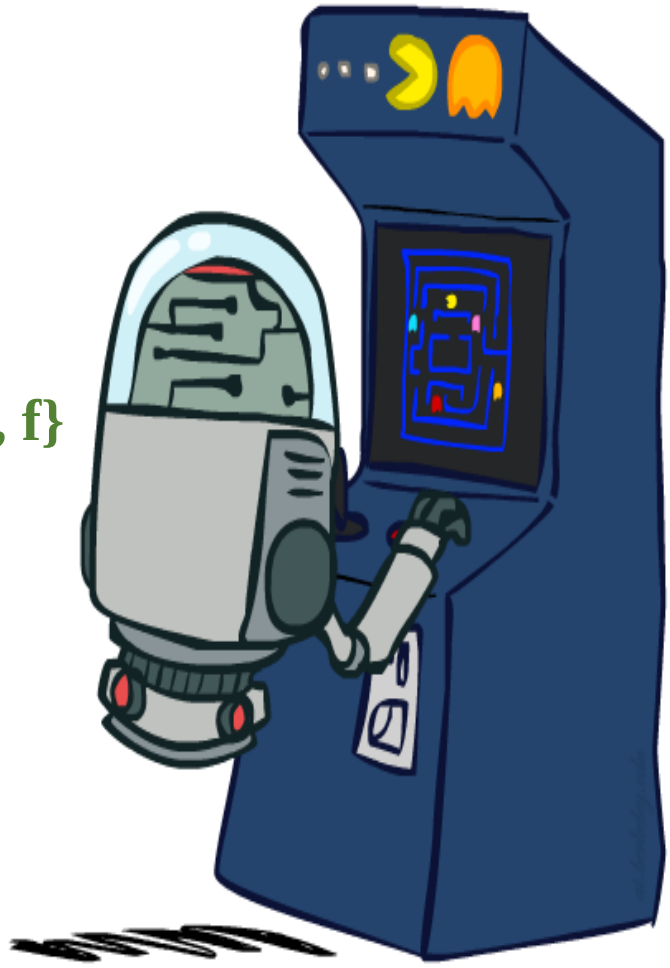
Tipos de juegos



- ¡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**

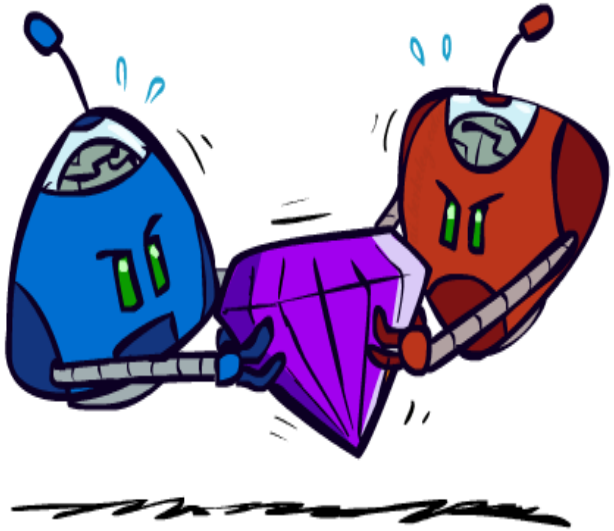
Juegos determinísticos

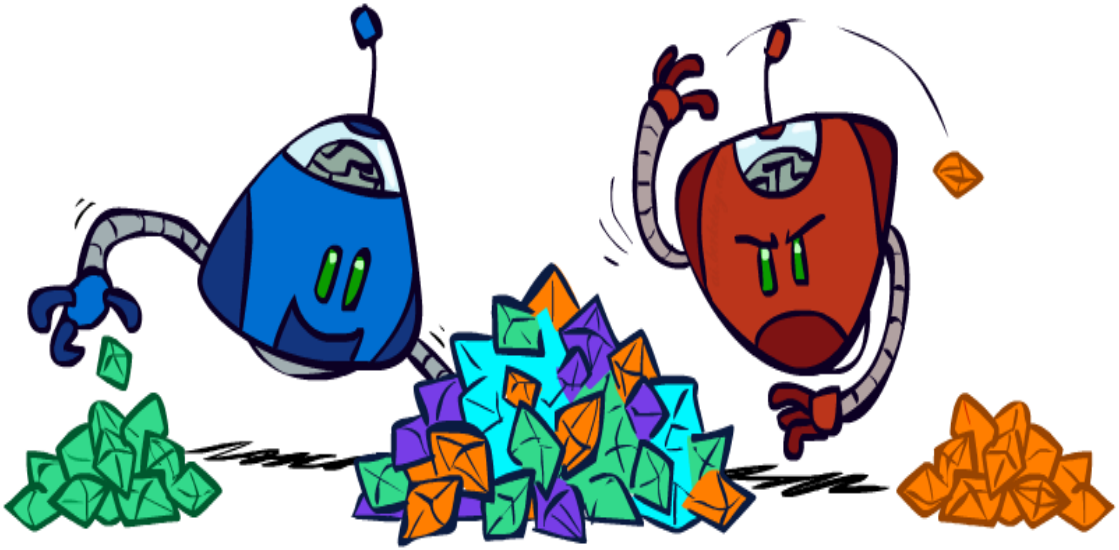
- **Muchas formalizaciones posibles**, una es:
  - Estados: **S** (inicio en s_0)
  - Jugadores: **P**={1...N} (normalmente a turnos)
  - Acciones: **A**
  - Función de transición: **S x A → S**
  - Test de terminación (estado objetivo o final): **S → {t , f}**
  - **Función de utilidad** para estados terminales (**borde**)



- La solución para un jugador es una política: **S → A**

Juegos de suma cero

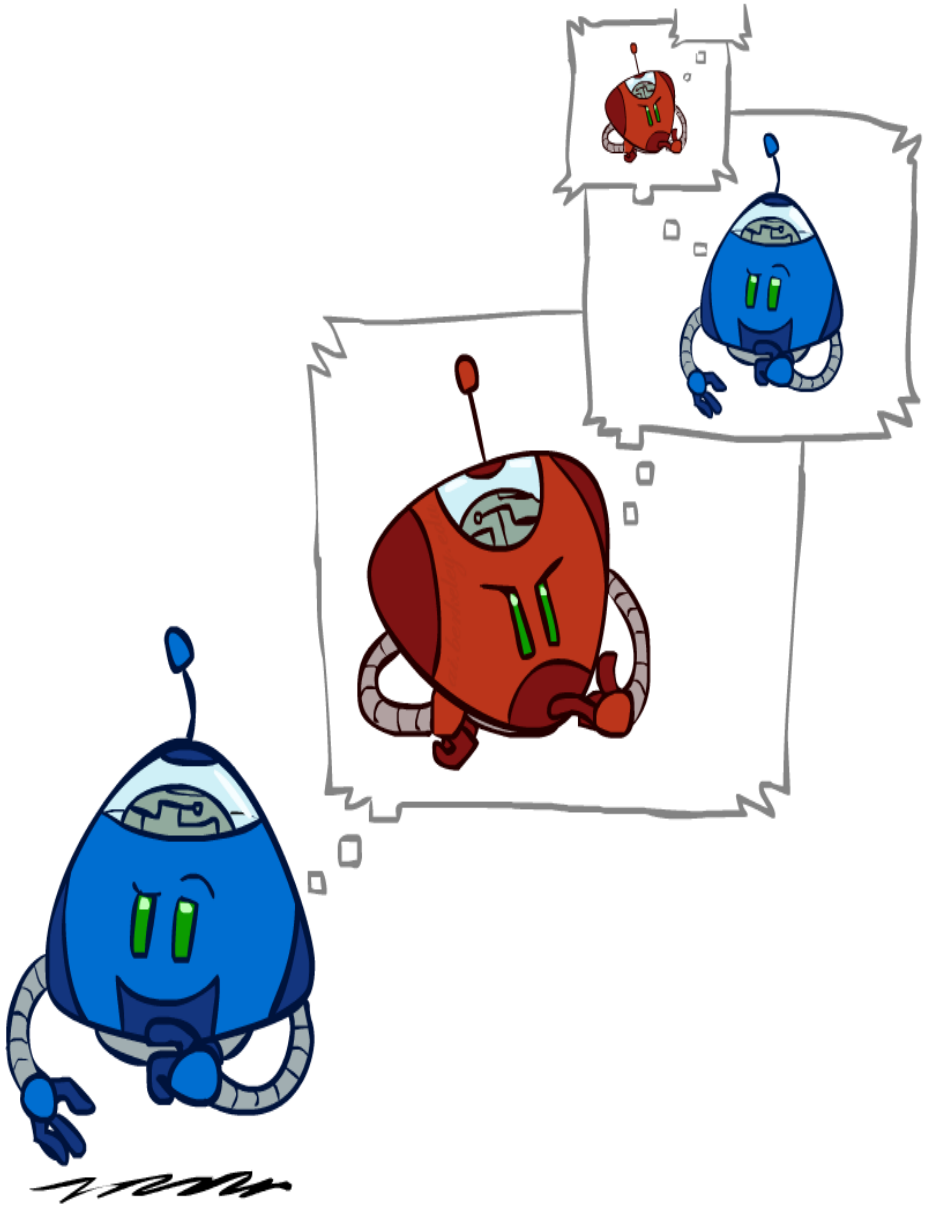




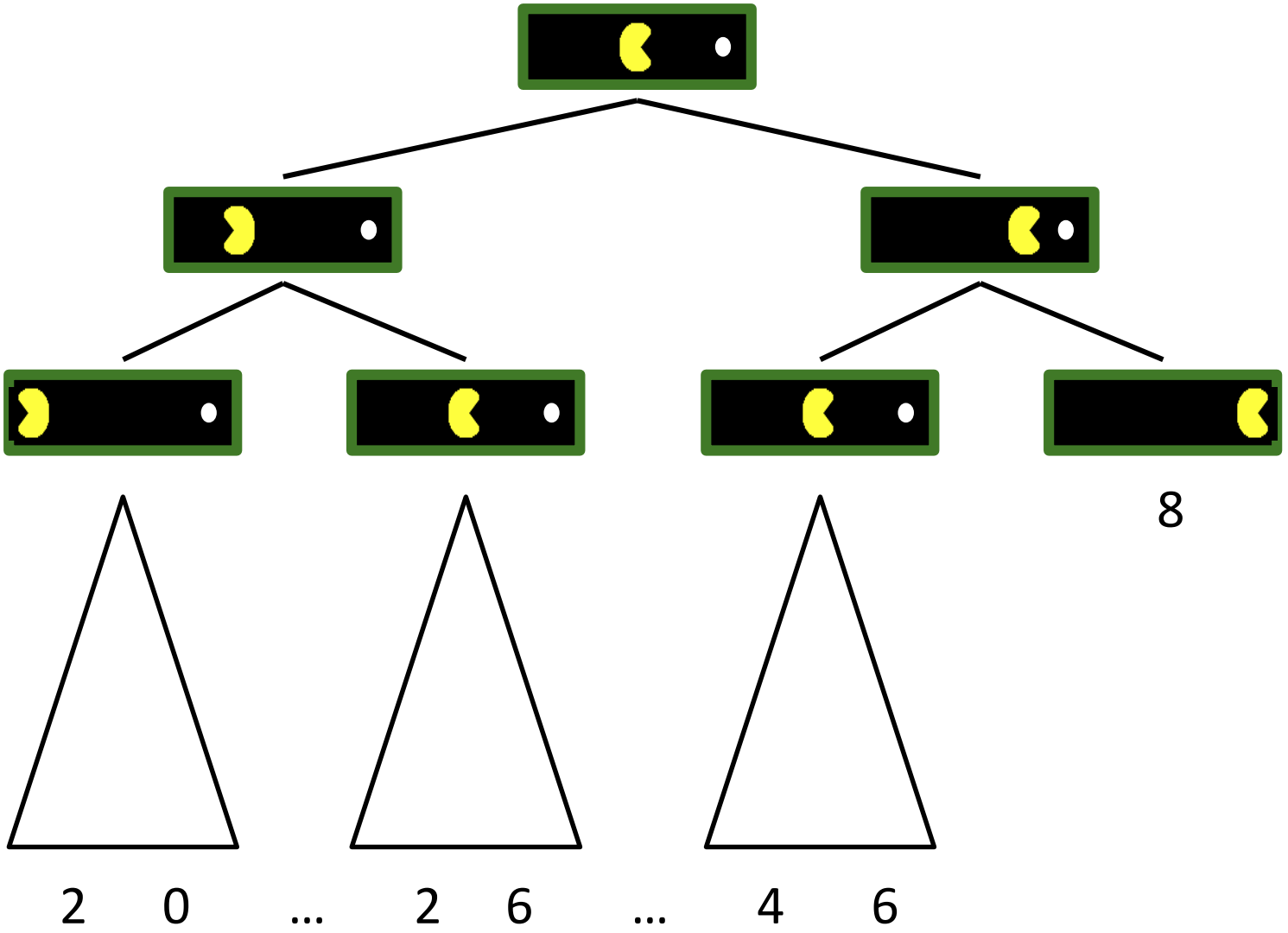
- Juegos de suma cero
  - Los agentes tienen utilidades opuestas (valores)
  - Podemos pensar en un único valor que uno maximiza y el otro minimiza
  - **Adversarial, competición pura**

- Juegos generales
  - Los agentes tienen utilidades independientes (valores)
  - **Cooperación, competición, y más, todo es posible**

Búsqueda Adversarial: suma cero



Árboles con un solo Agente

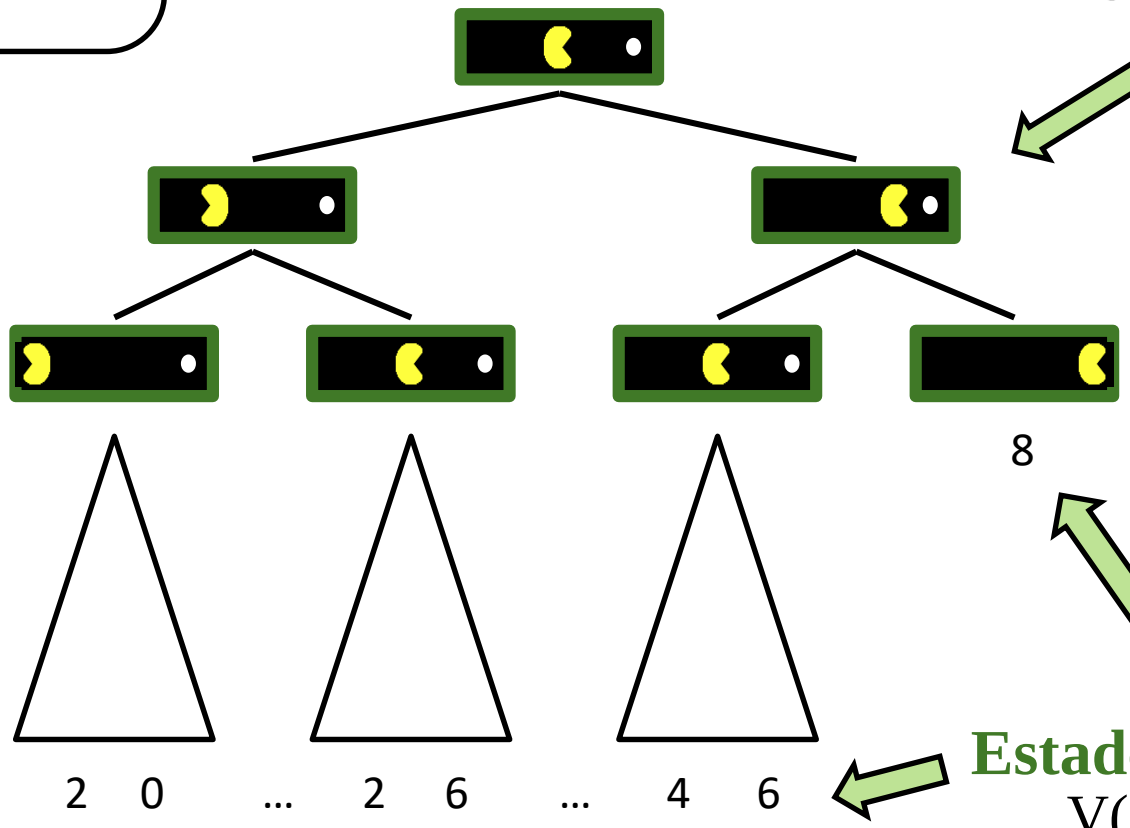


Valor de un estado

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

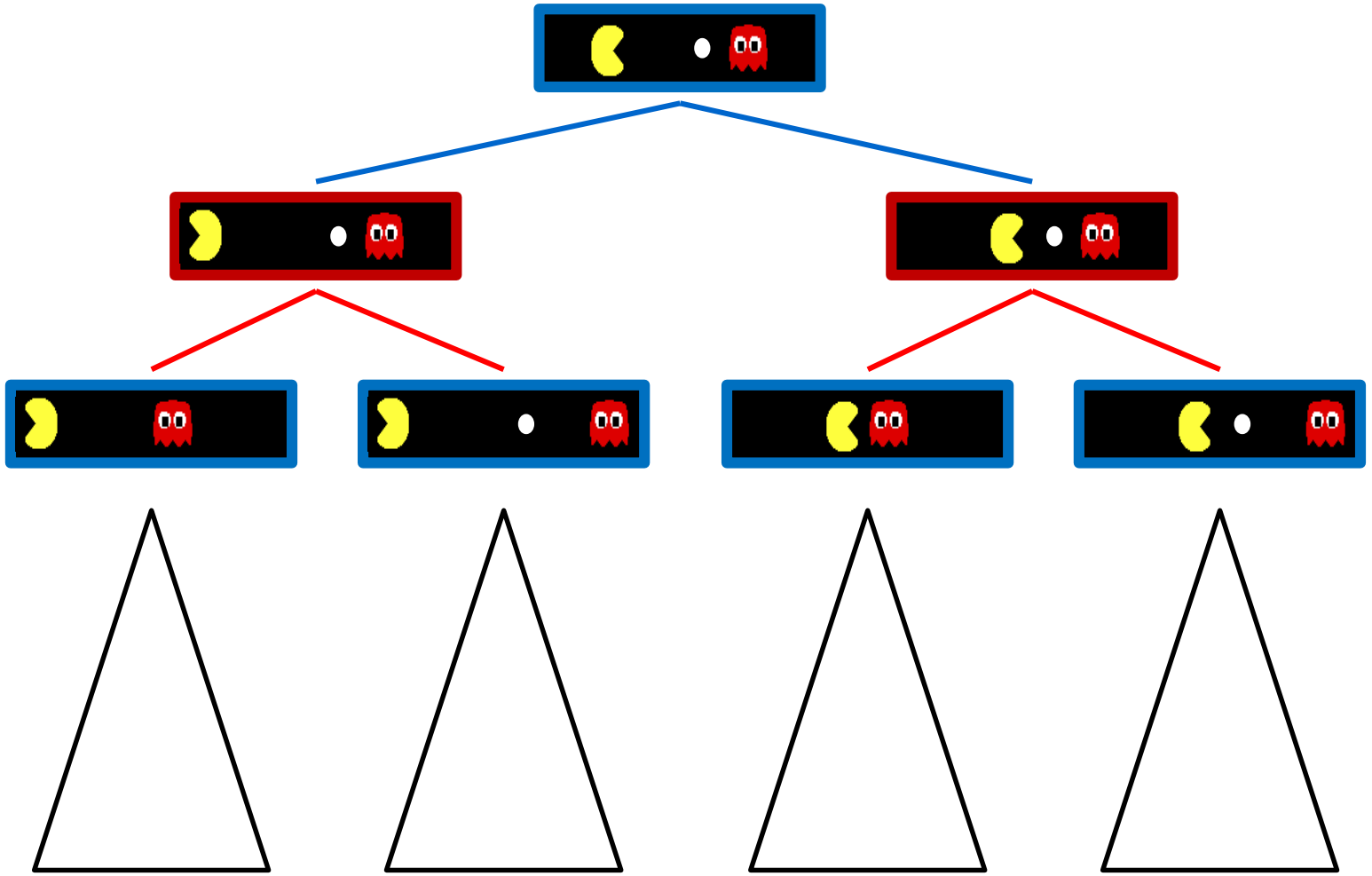
Estados no terminales:

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



Estados terminales:
V(s) = conocido

Árboles de estados Adversariales: 1 adversario



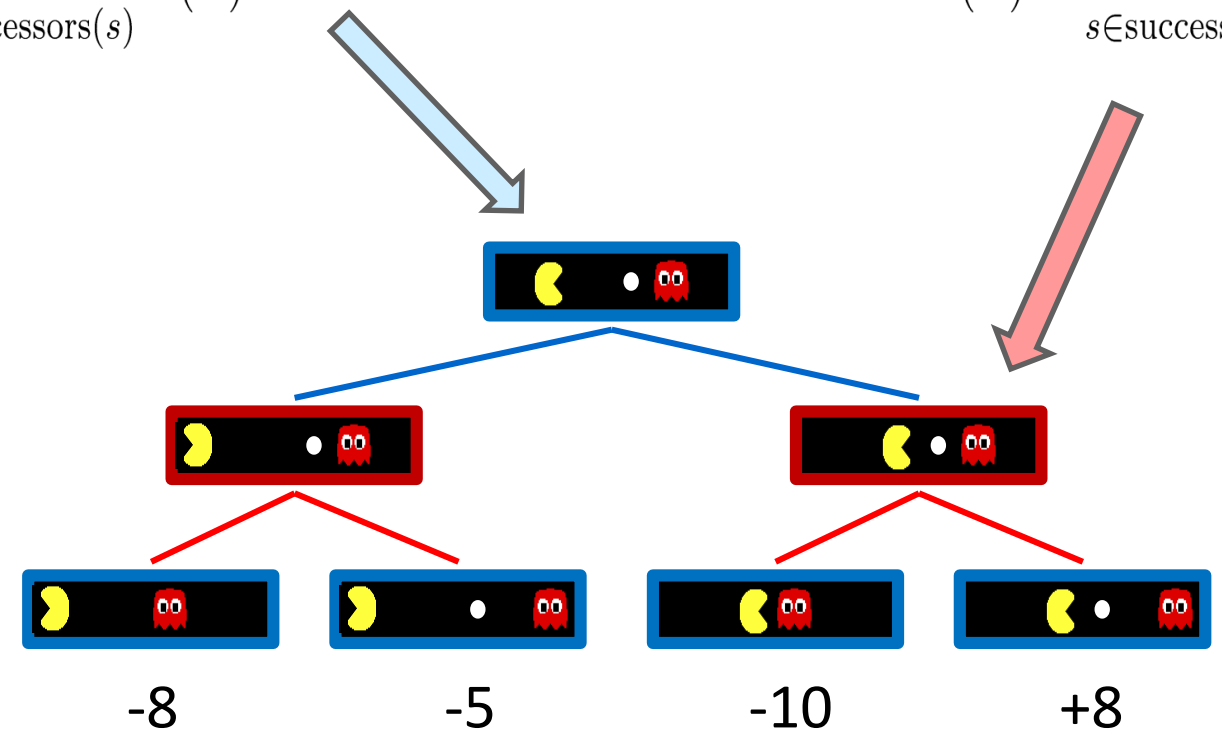
Valores Minimax: 1 adversario

Estados bajo el control del agente:

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

Estados bajo el control del oponente:

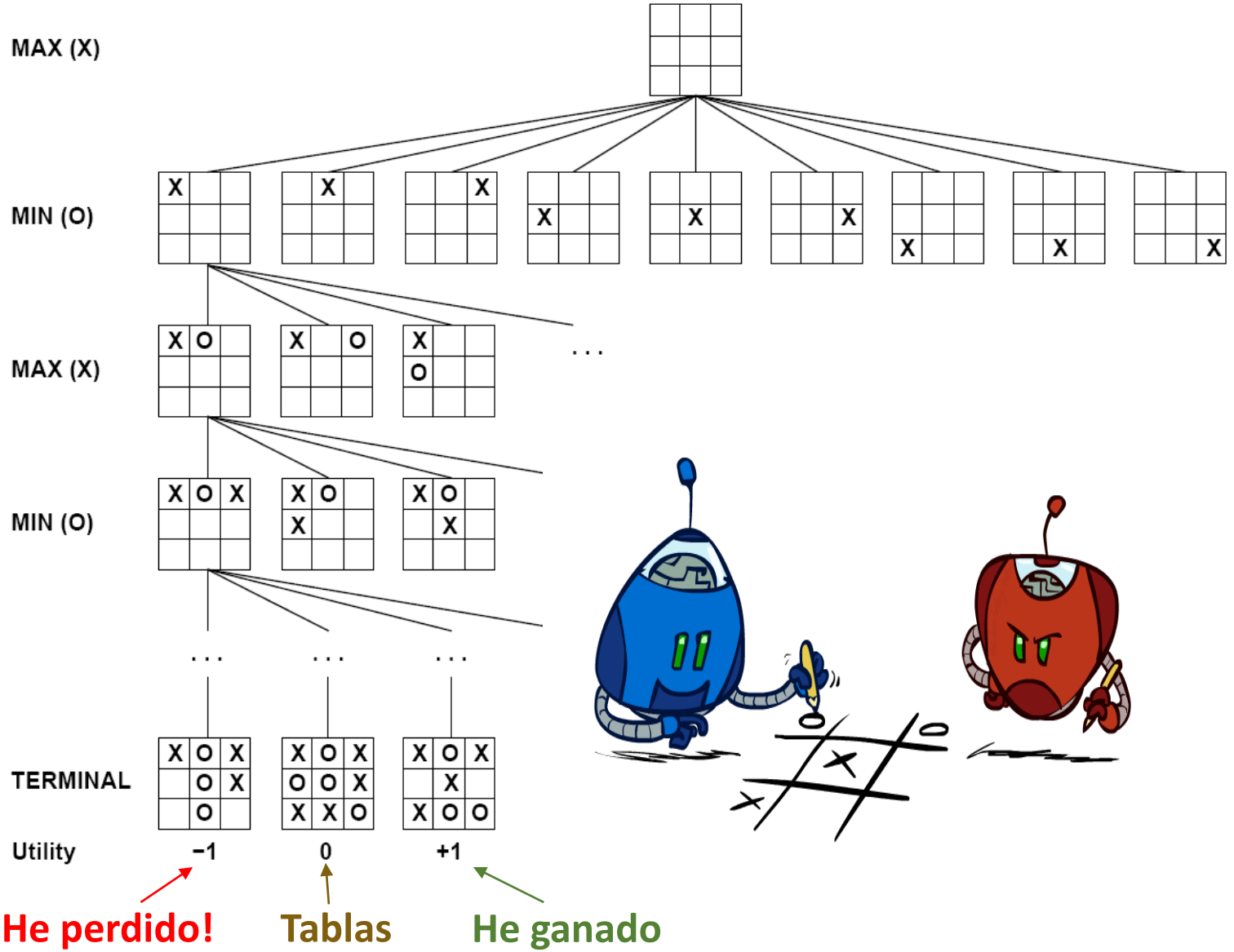
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Estados terminales:

V(s) = conocido

Árbol de juegos de Tic-Tac-Toe

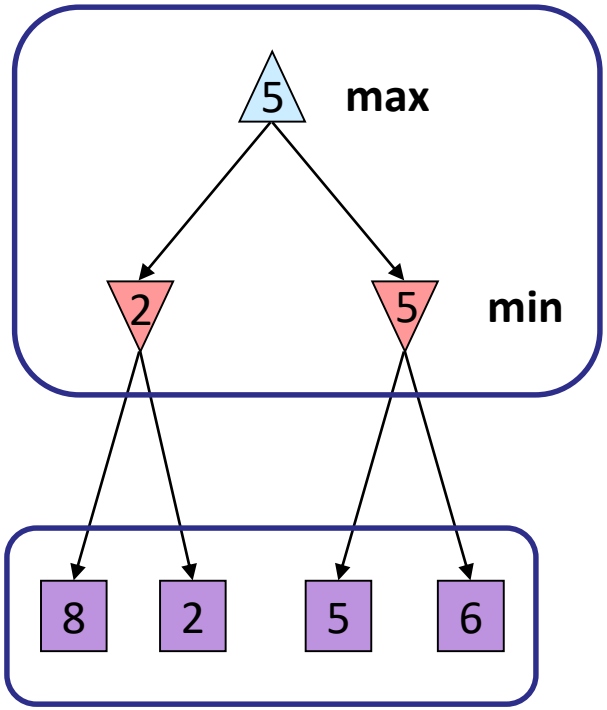


Búsqueda Adversarial (Minimax): 1 adversario

- 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 estado: el máximo resultado (**utility**) posible, contra un adversario racional (**óptimo**)

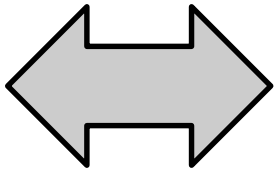
Valores Minimax:
calculados **recursivamente**



Valores terminales

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
```

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

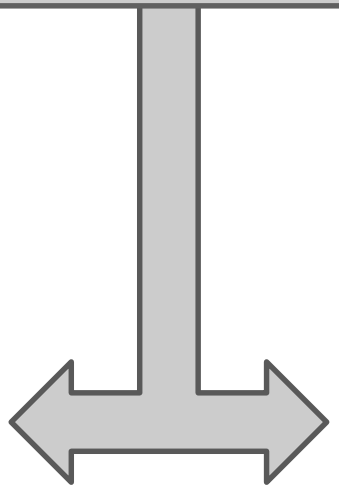
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

El valor **v empieza con -∞ y va aumentando**

El valor **v empieza con +∞ y va disminuyendo**

Implementación de Minimax (Dispatch)

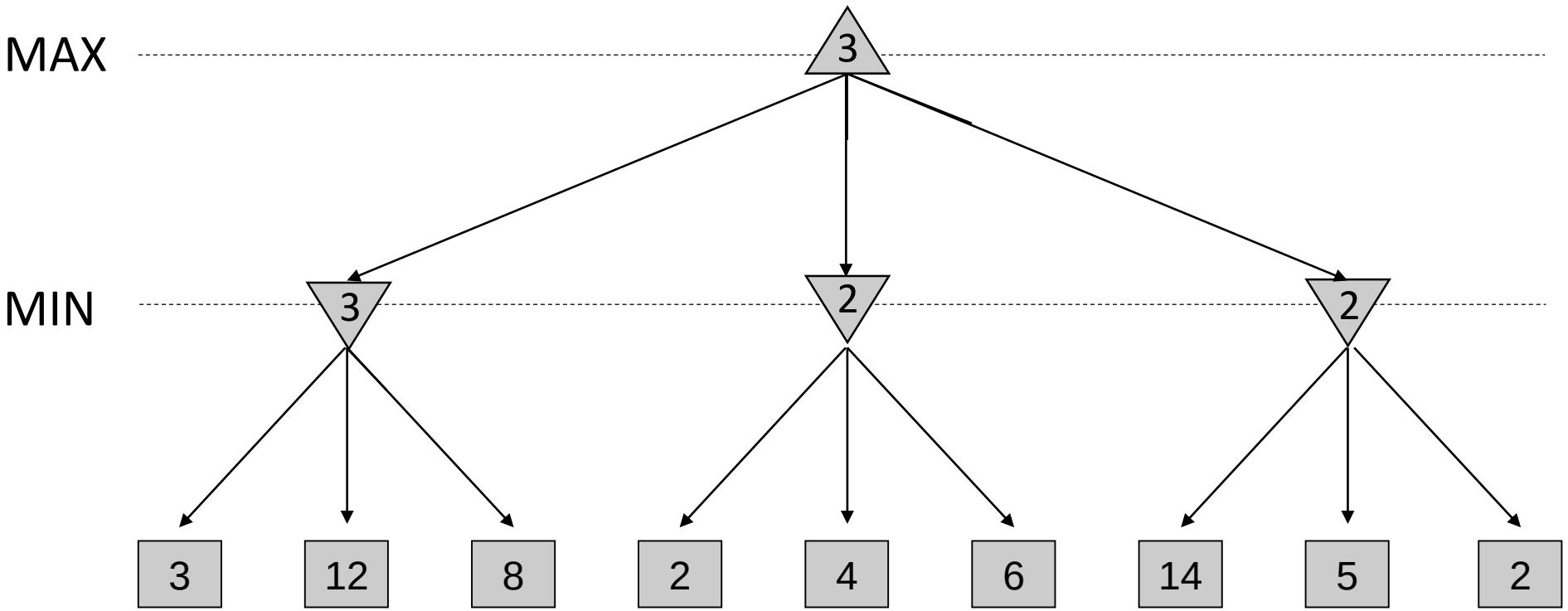
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)



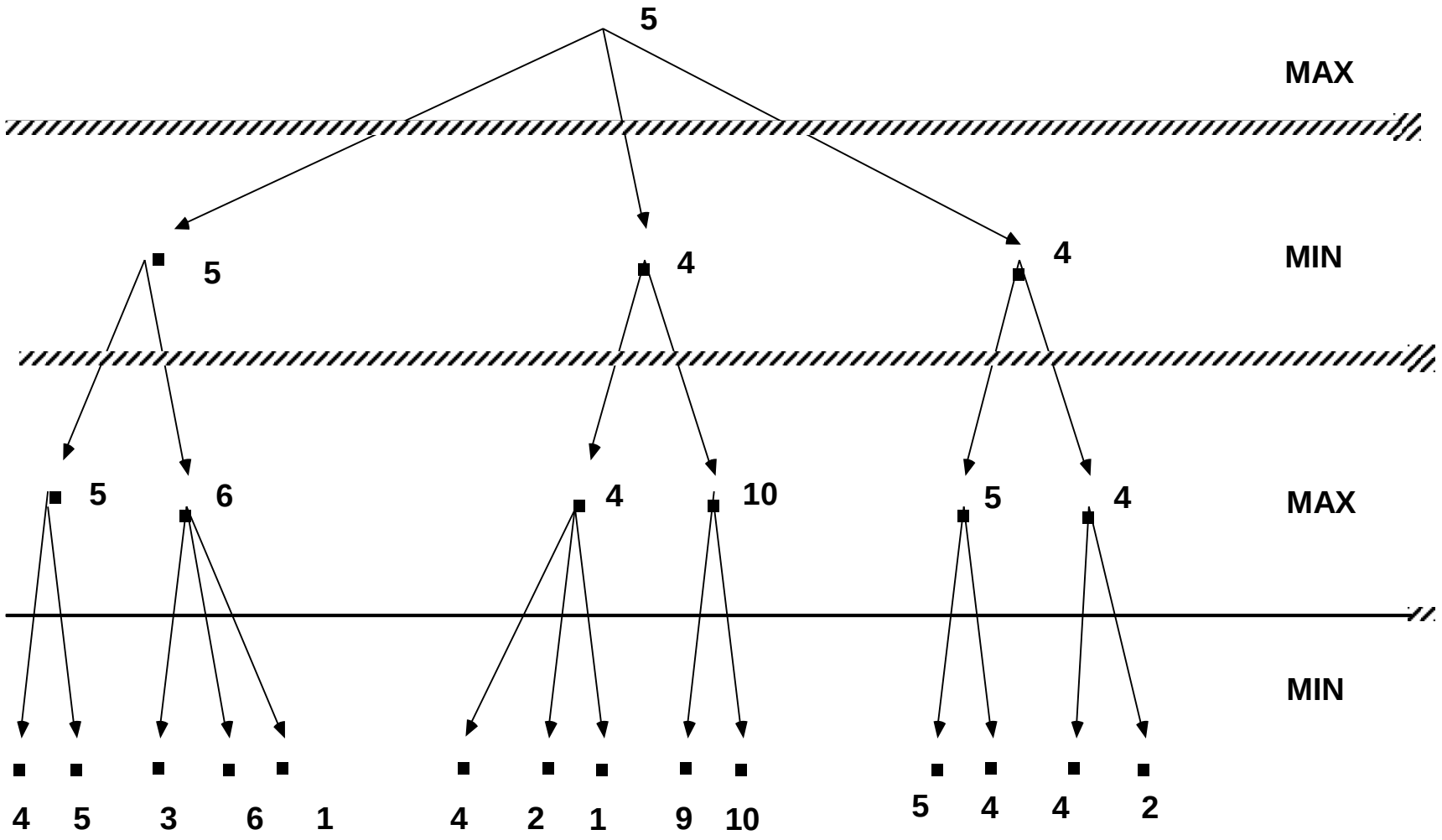
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v,
            value(successor))
    return v

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

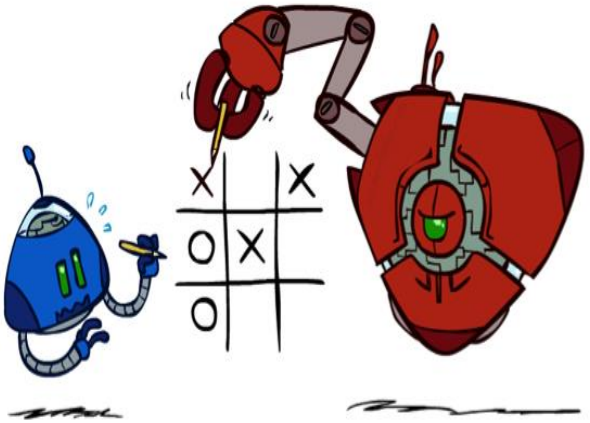
Ejemplo de Minimax

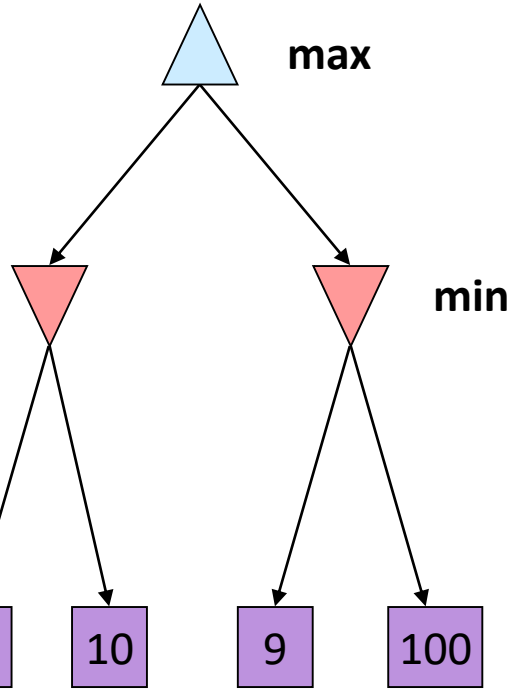


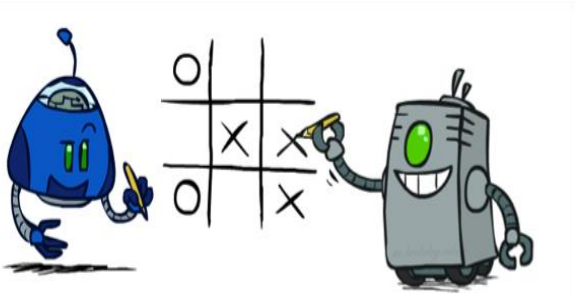
Árbol de una aplicación del mini-max



Propiedades de Minimax

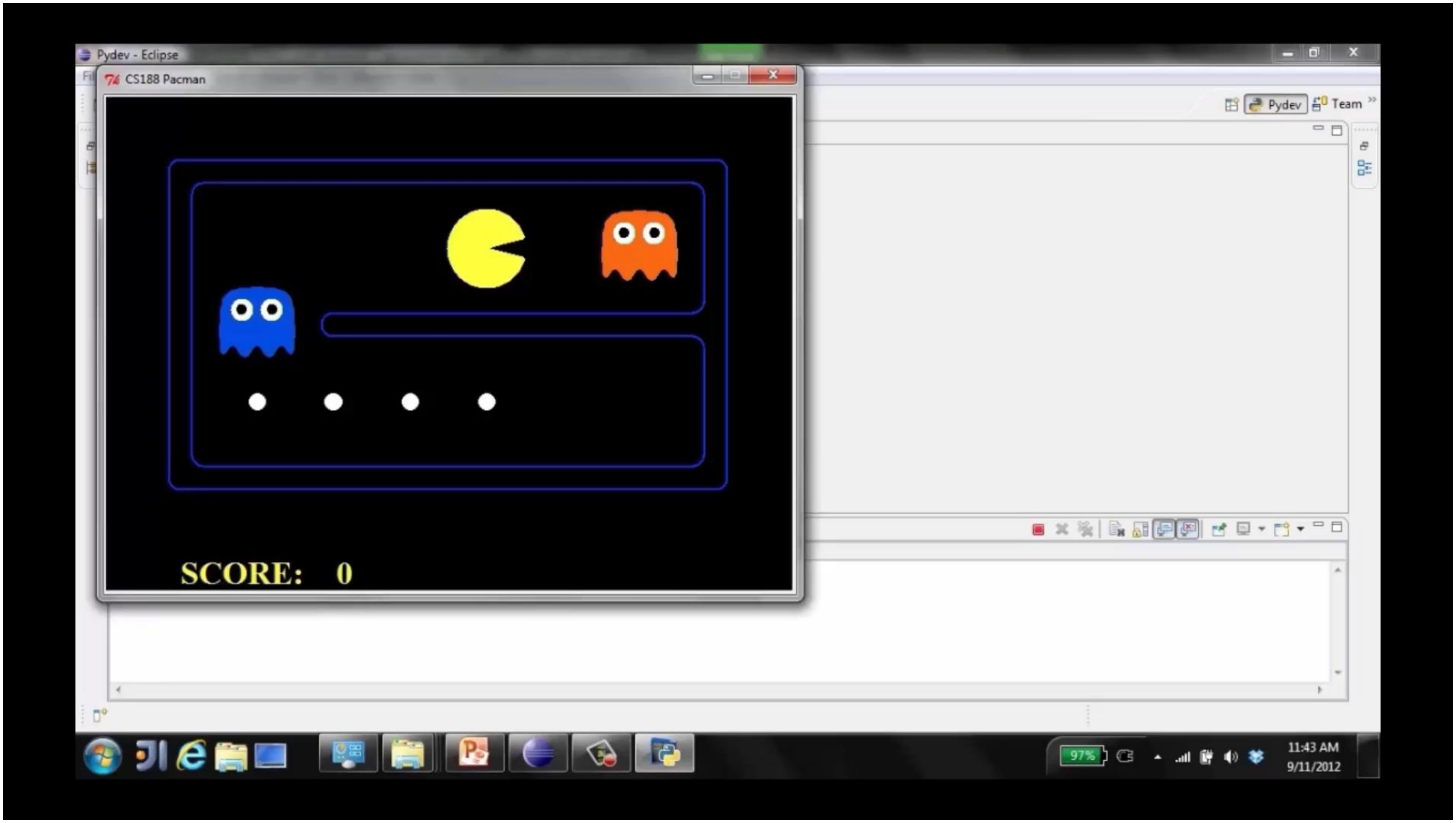






Óptimo contra un jugador perfecto. **¿En otro caso?**

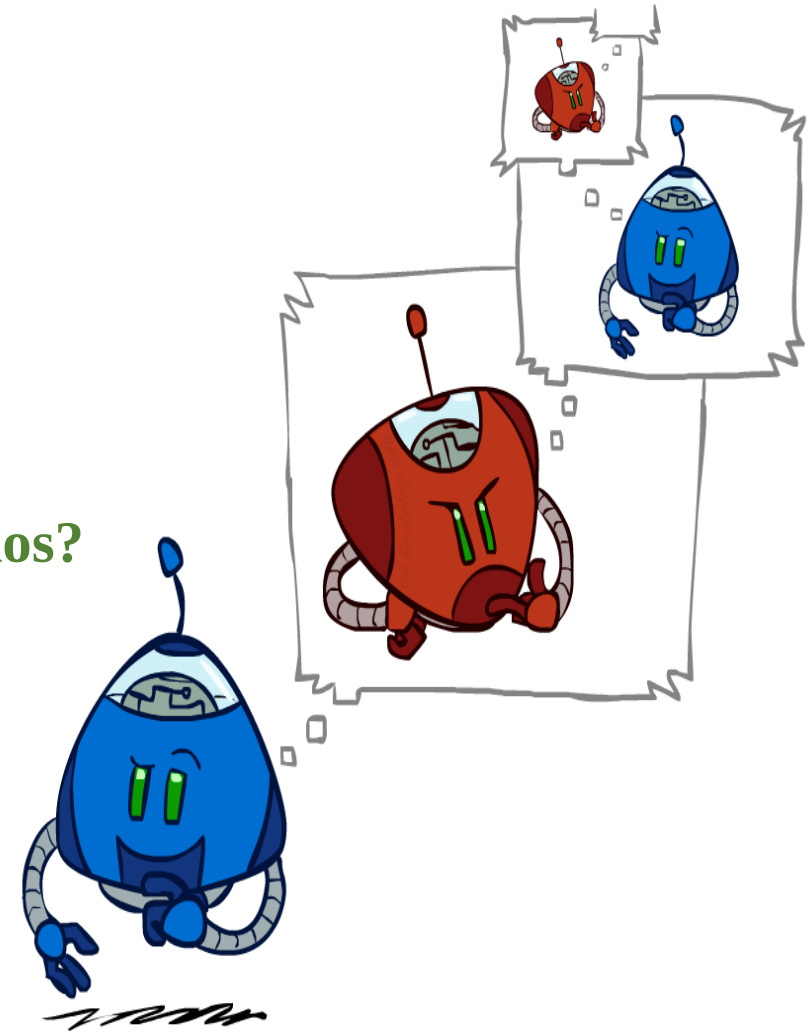
Video Demo Minimax: fantasmas racionales!



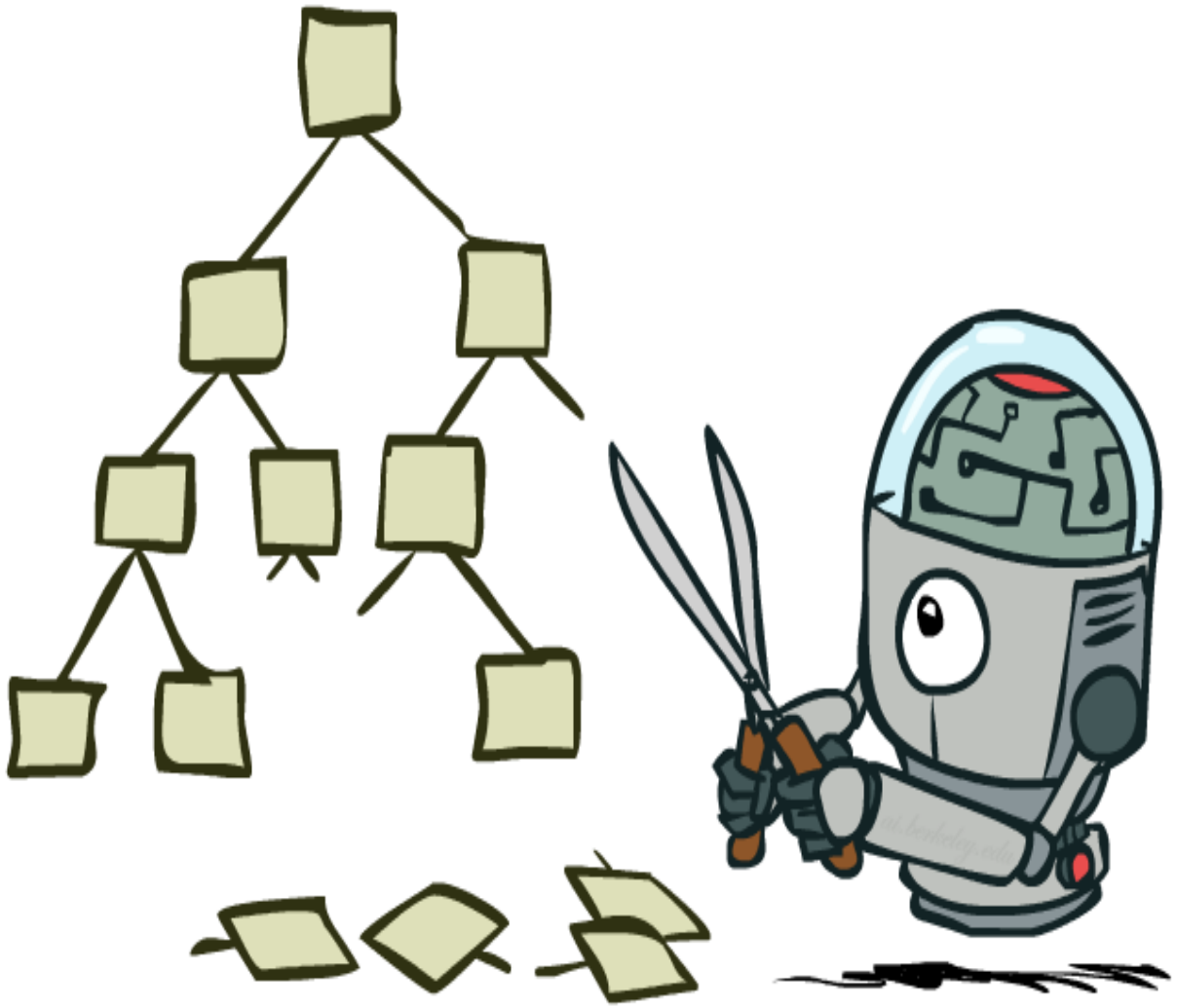
Eficiencia de Minimax

- ¿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 todos los estados?**



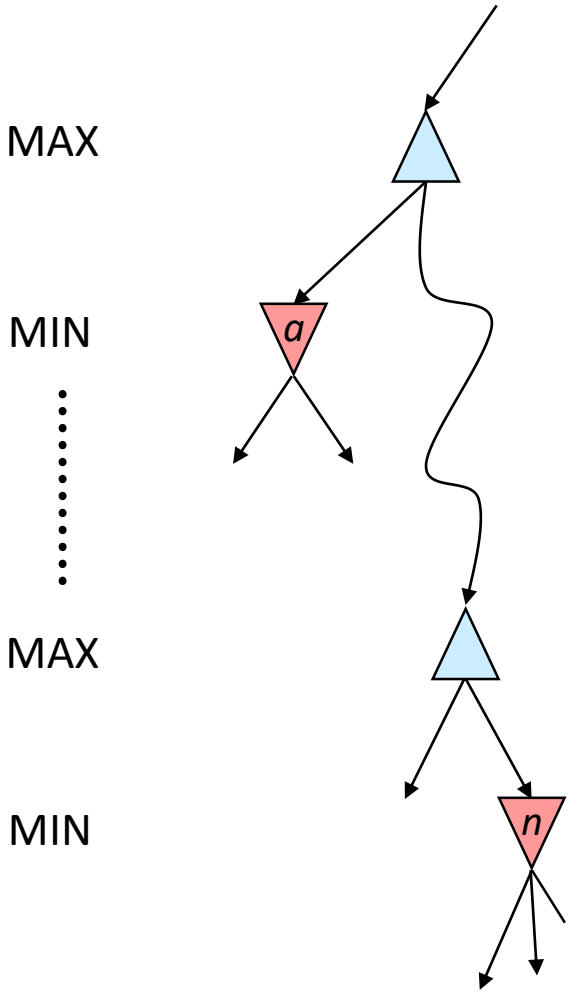
Podado del árbol de juego



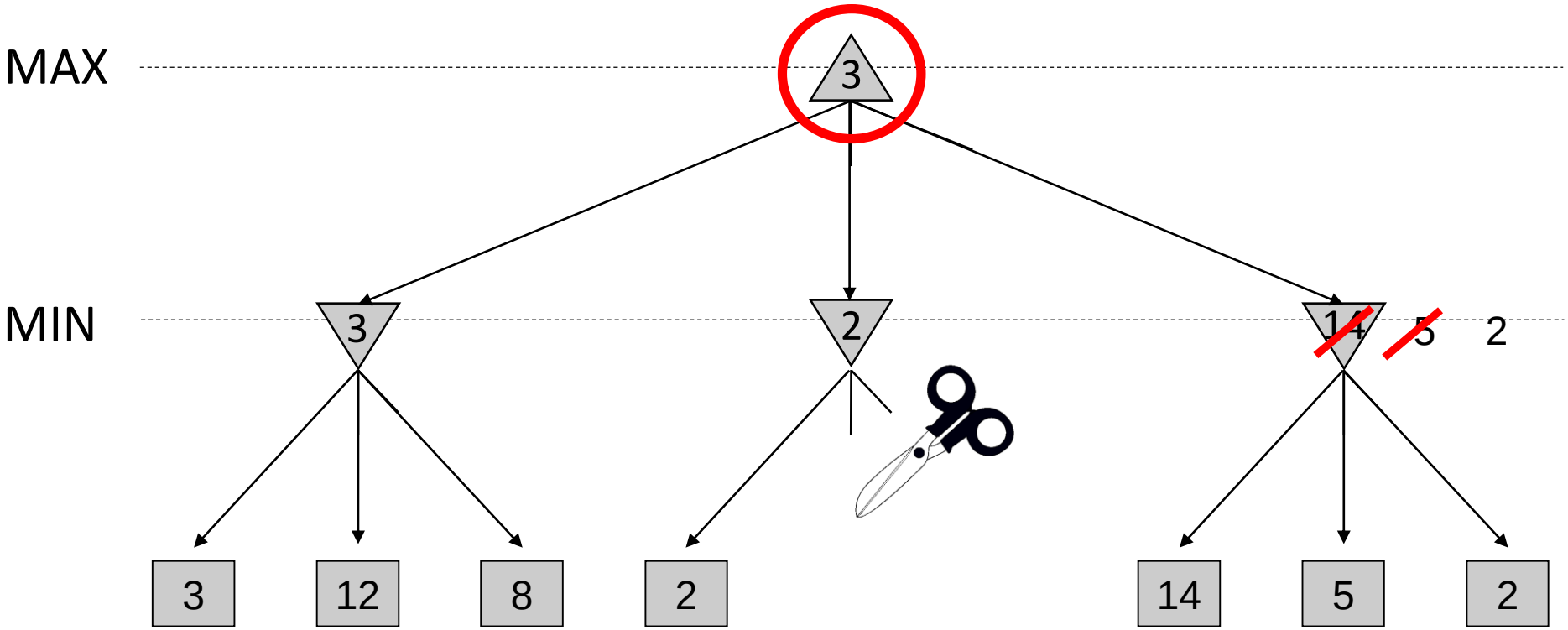
Podado Alpha-Beta

- 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



Podado de Minimax



Implementación de Alfa-Beta

α: la mejor opción de MAX's en el camino a la raíz: inicializar con -∞

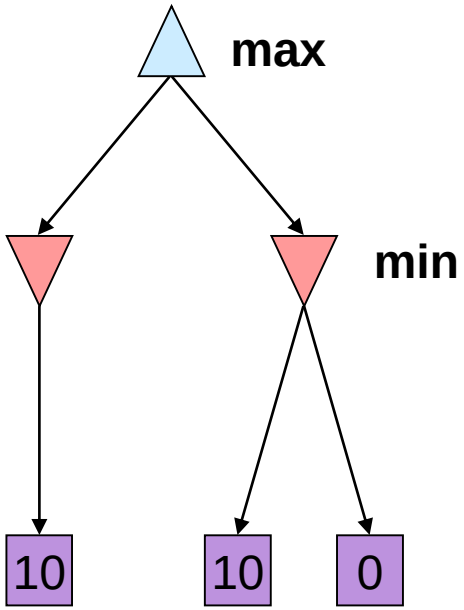
β: la mejor opción de MIN's en el camino a la raíz: inicializar con +∞

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

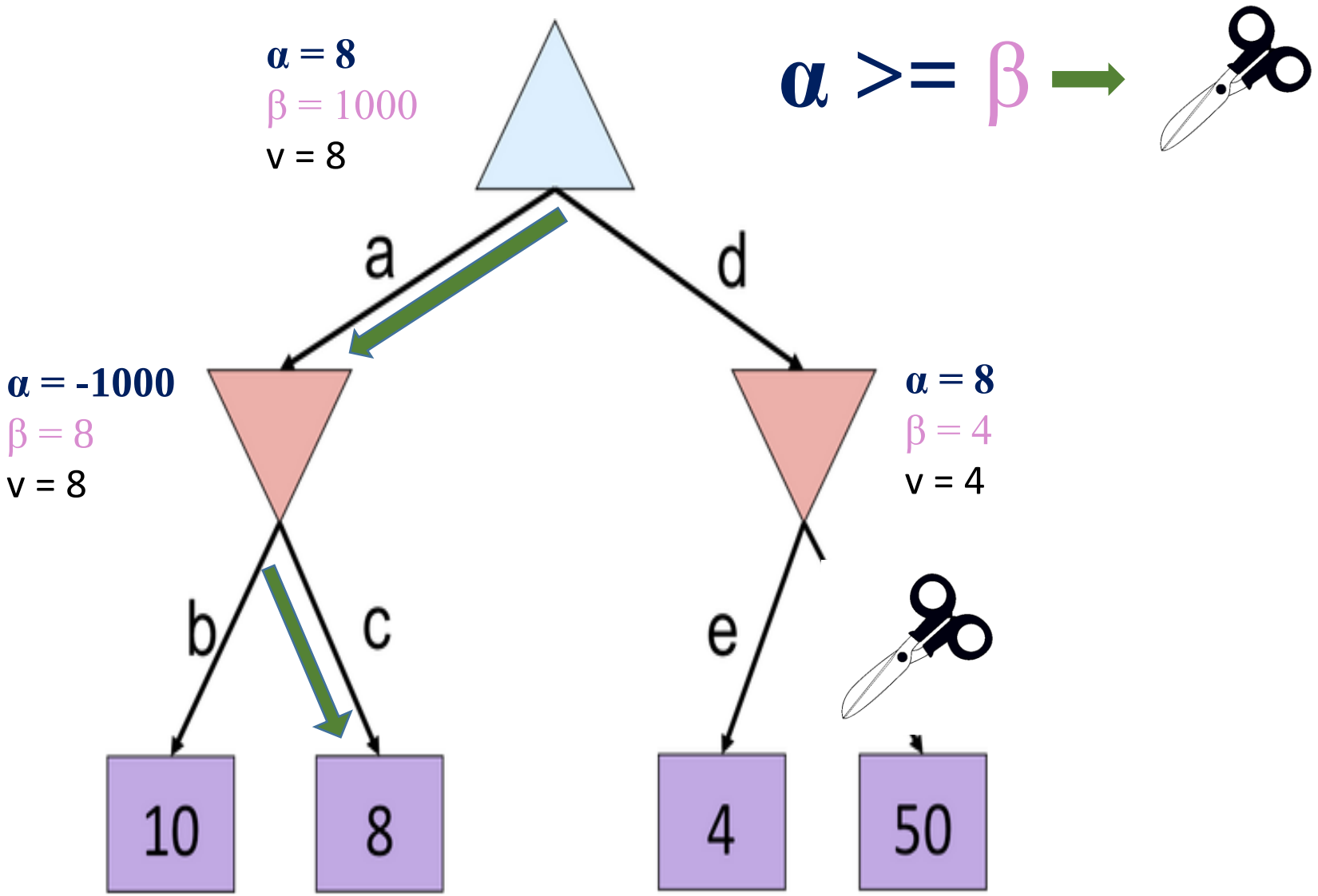
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, max_value(successor, α, β))
        if v < α:
            return v
        β = min(β, v)
    return v
```

Propiedades de podado Alfa-Beta

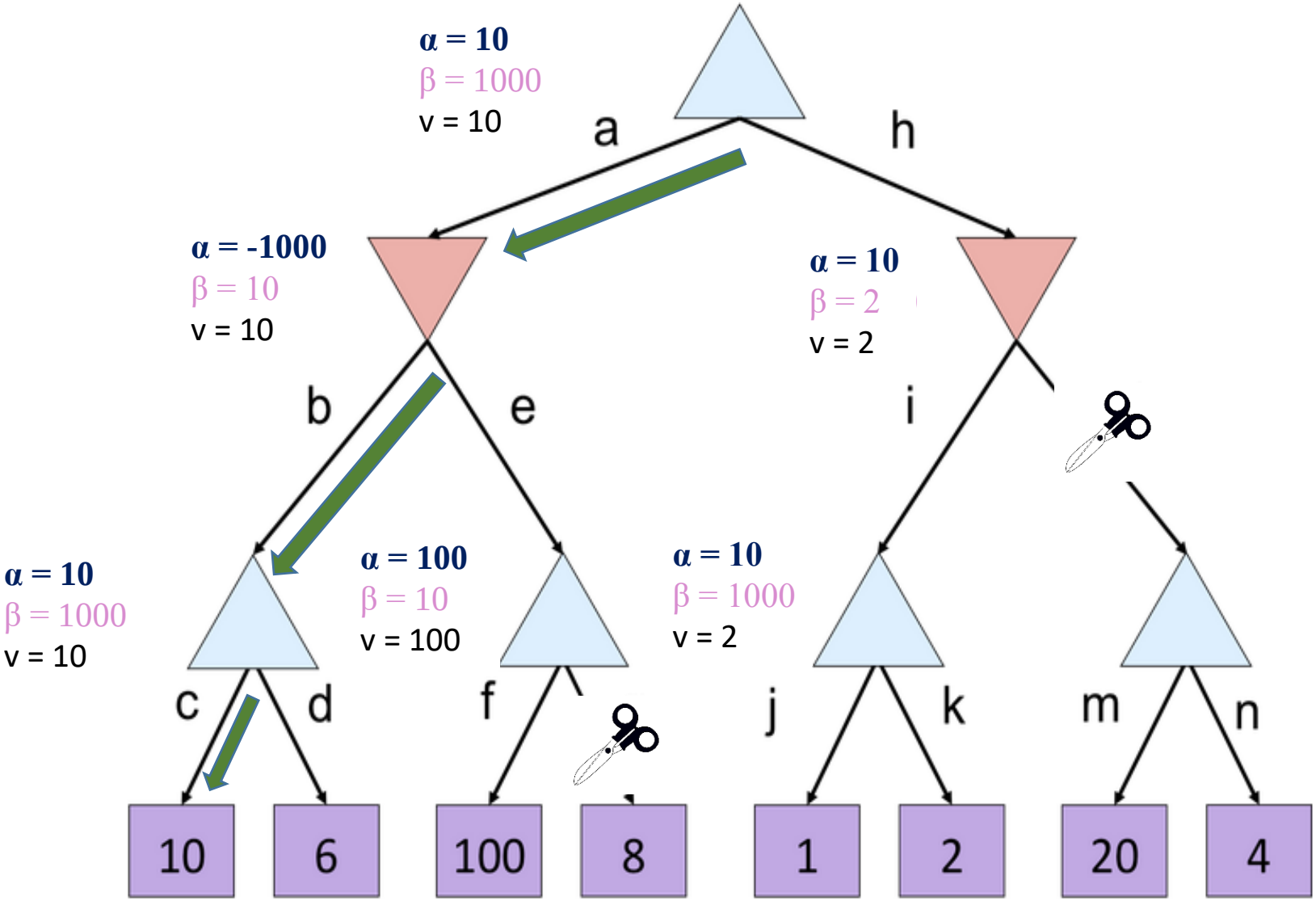
- ¡**El podado no tiene efecto sobre el valor minmax** calculado para la raíz!
- **Una buena ordenación de los hijos mejora la efectividad del podado**
- Con una **"ordenación perfecta"**:
  - La complejidad en tiempo **se reduce** a **$O(b^{m/2})$**



Alpha-Beta Quiz



Alpha-Beta Quiz 2



Alpha-Beta Demo

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

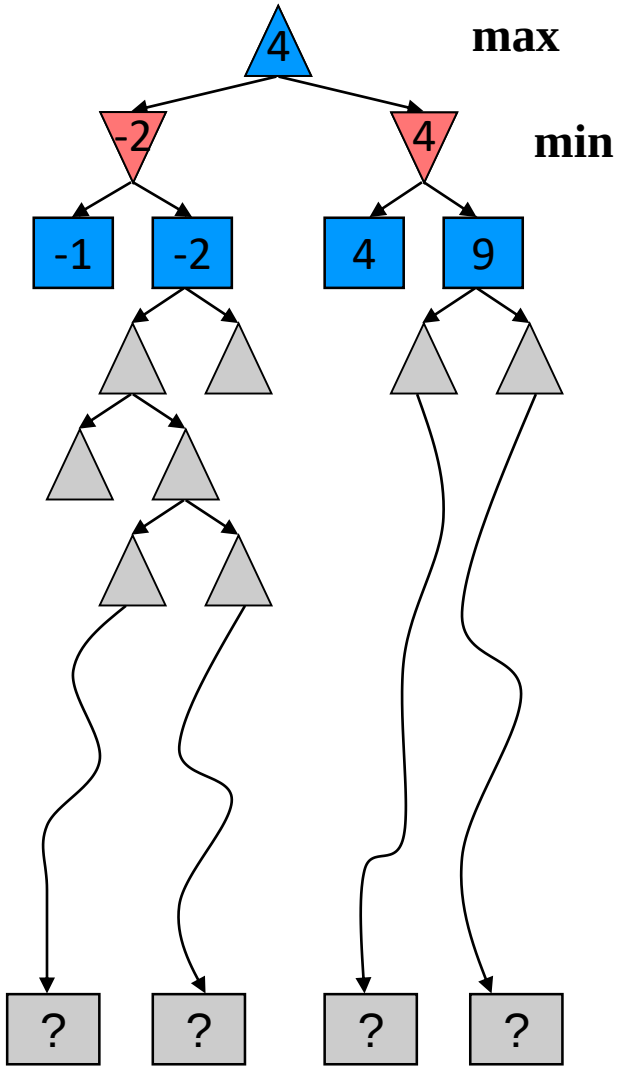
  - **http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html**

Recursos limitados

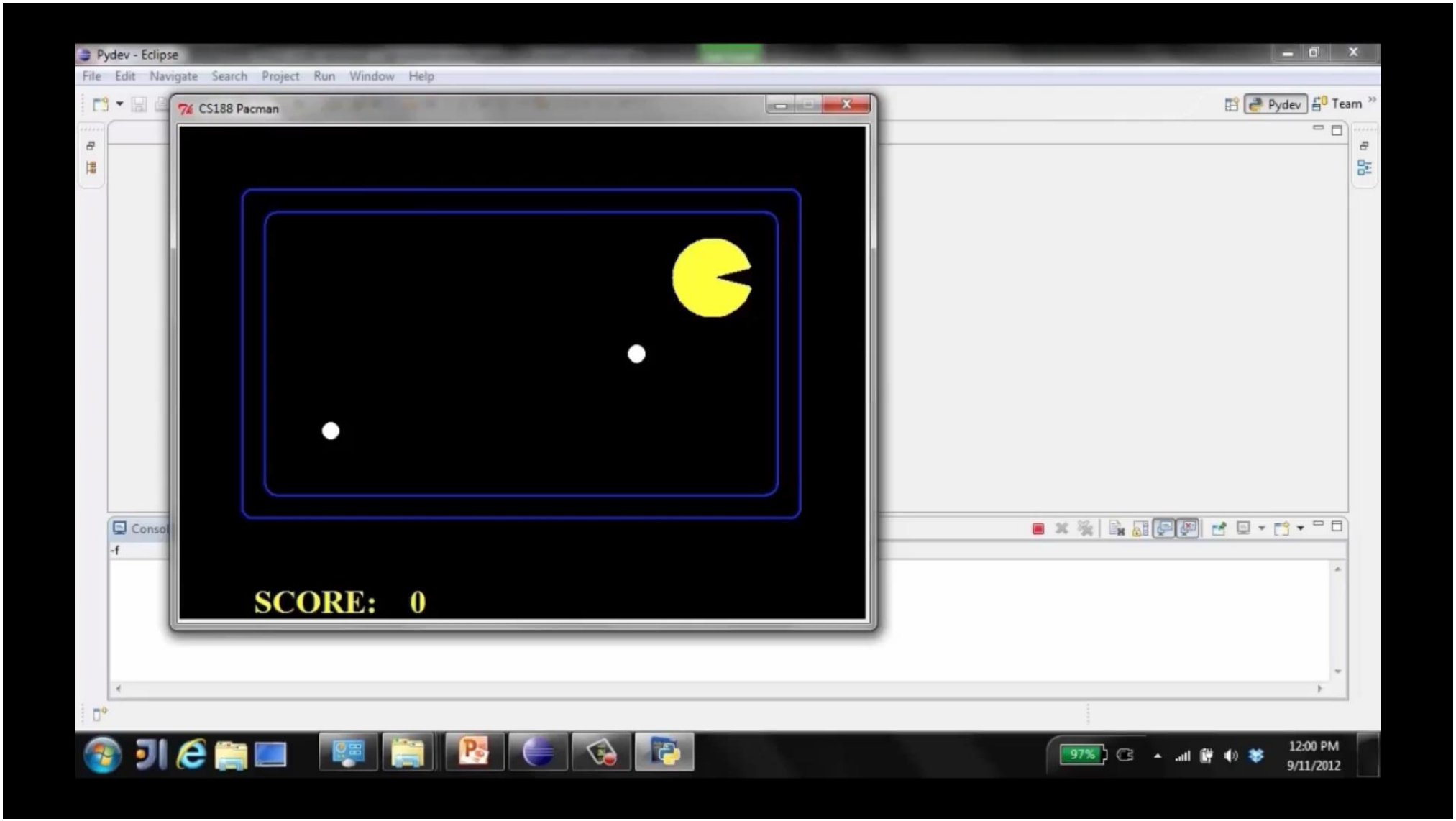
Recursos limitados

- Problema: en juegos reales, **¡no podemos buscar hasta estados terminales!**
- **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 estados no terminales

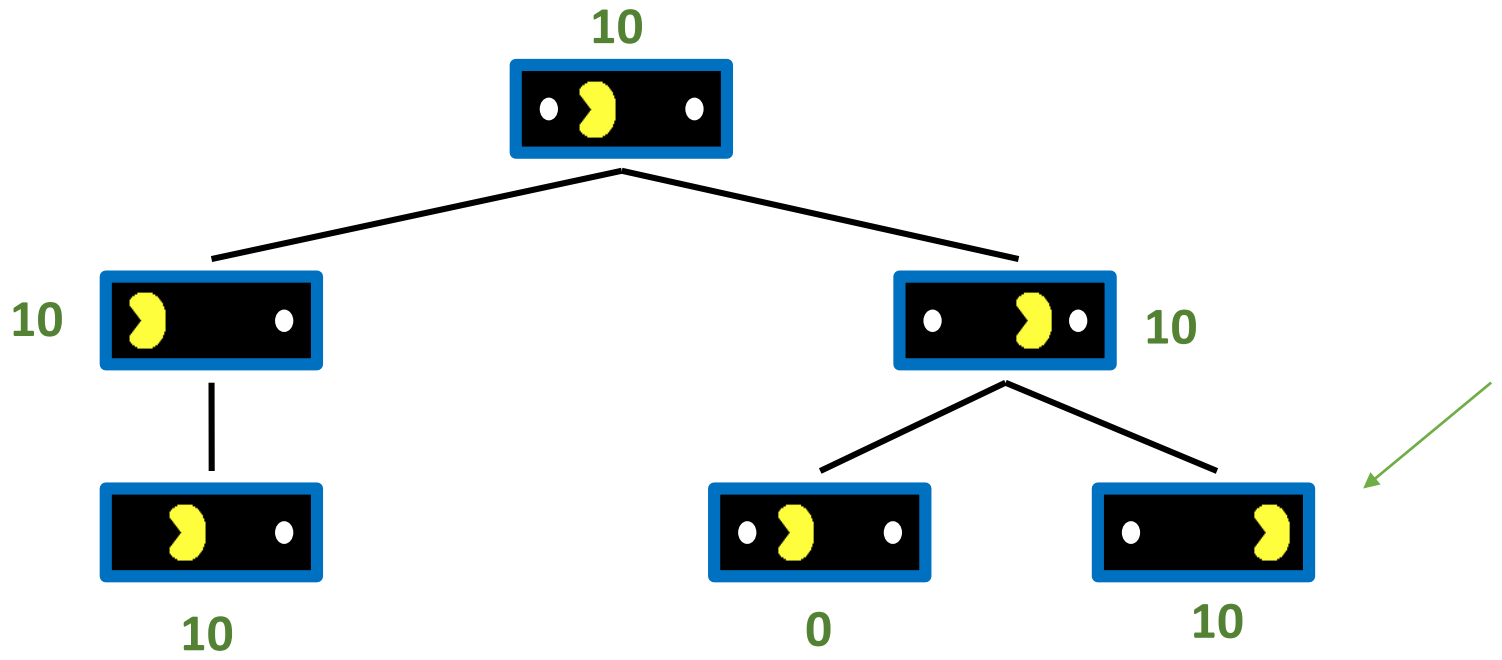
- **La garantía de juego óptimo se desvanece**



D = 2 (función de evaluación: movimientos)

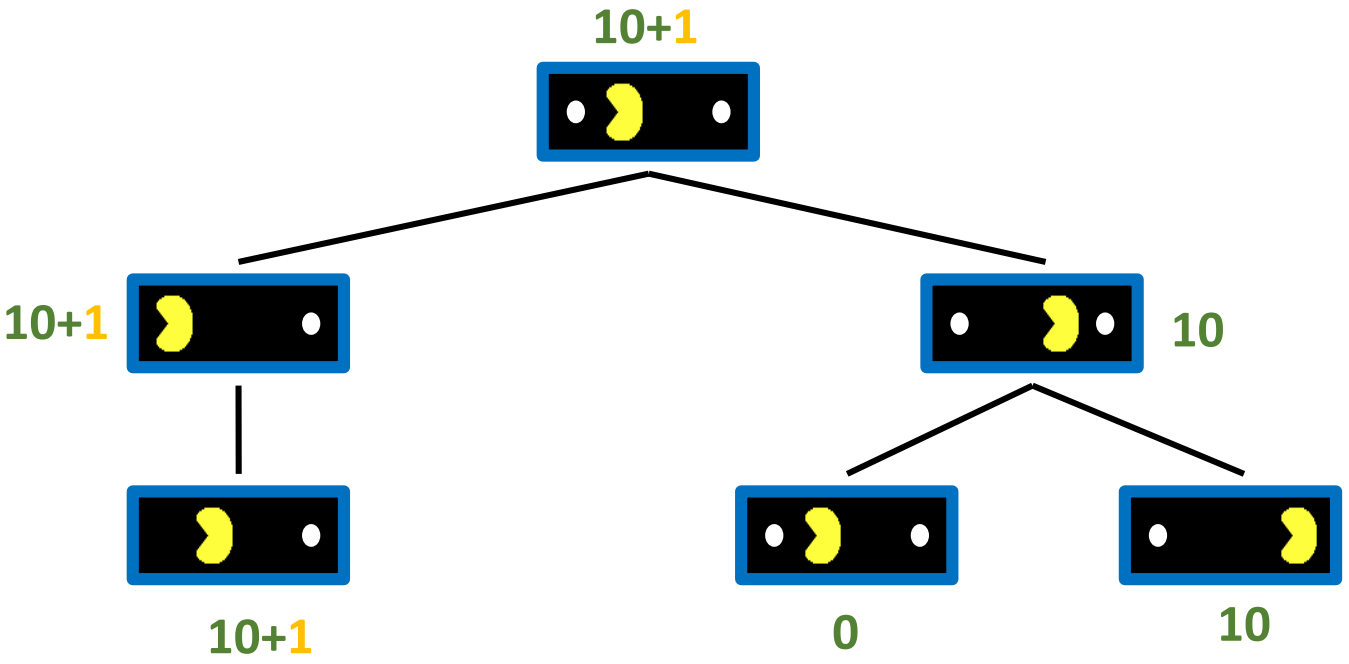


¿Por qué desfallece Pacman?



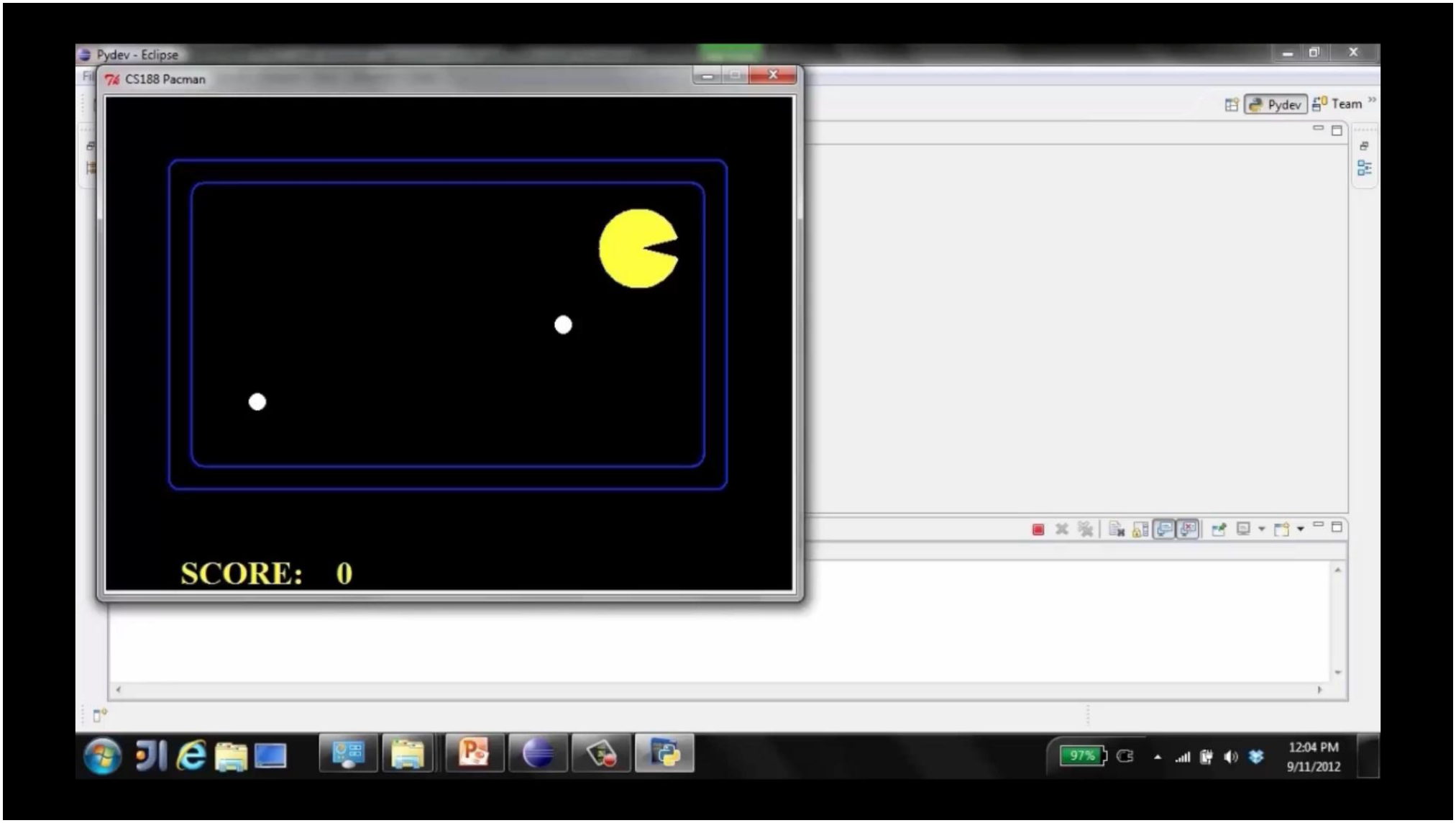
- ¡Peligro de agentes que replanifican!
  - Función de evaluación: **10 por cada punto comido**
  - Pacman sabe que su puntuación aumentará yendo en las 2 direcciones comiendo un punto (west, east).
    - Pero Pacman sabe que su puntuación también subirá si hace (east, east)
    - Después de comer el punto, no hay oportunidad de anotar más puntos (en el horizonte, en este caso, 2 movimientos (profundidad))

¡Pacman desfallece! ¿Mejora?

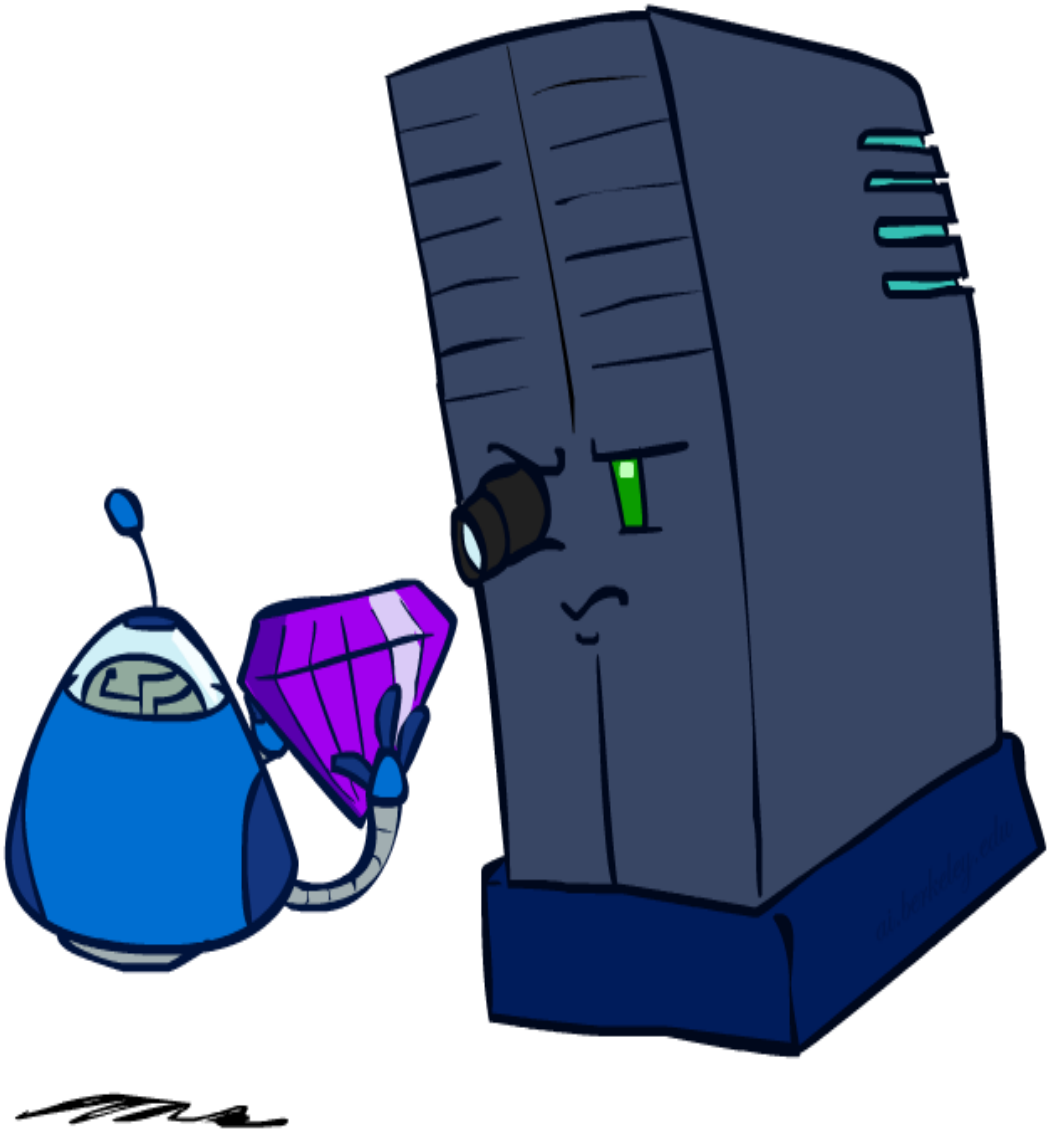


- ¡Peligro de agentes que replanifican!
  - El problema **no era el algoritmo!** sino la **función de evaluación!**
  - **Nueva función de evaluación:** 10 por cada punto comido **+ 1 por cercanía** al siguiente punto

D = 10 (función de evaluación: movimientos + cercanía)

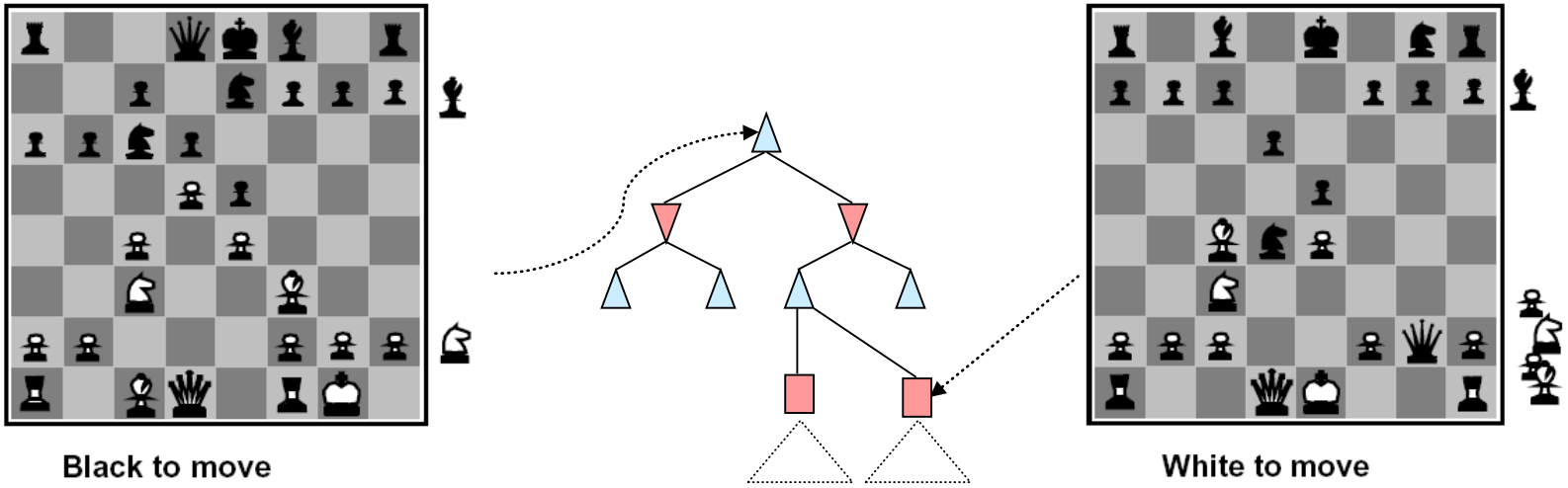


Funciones de evaluación



Funciones de evaluación

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



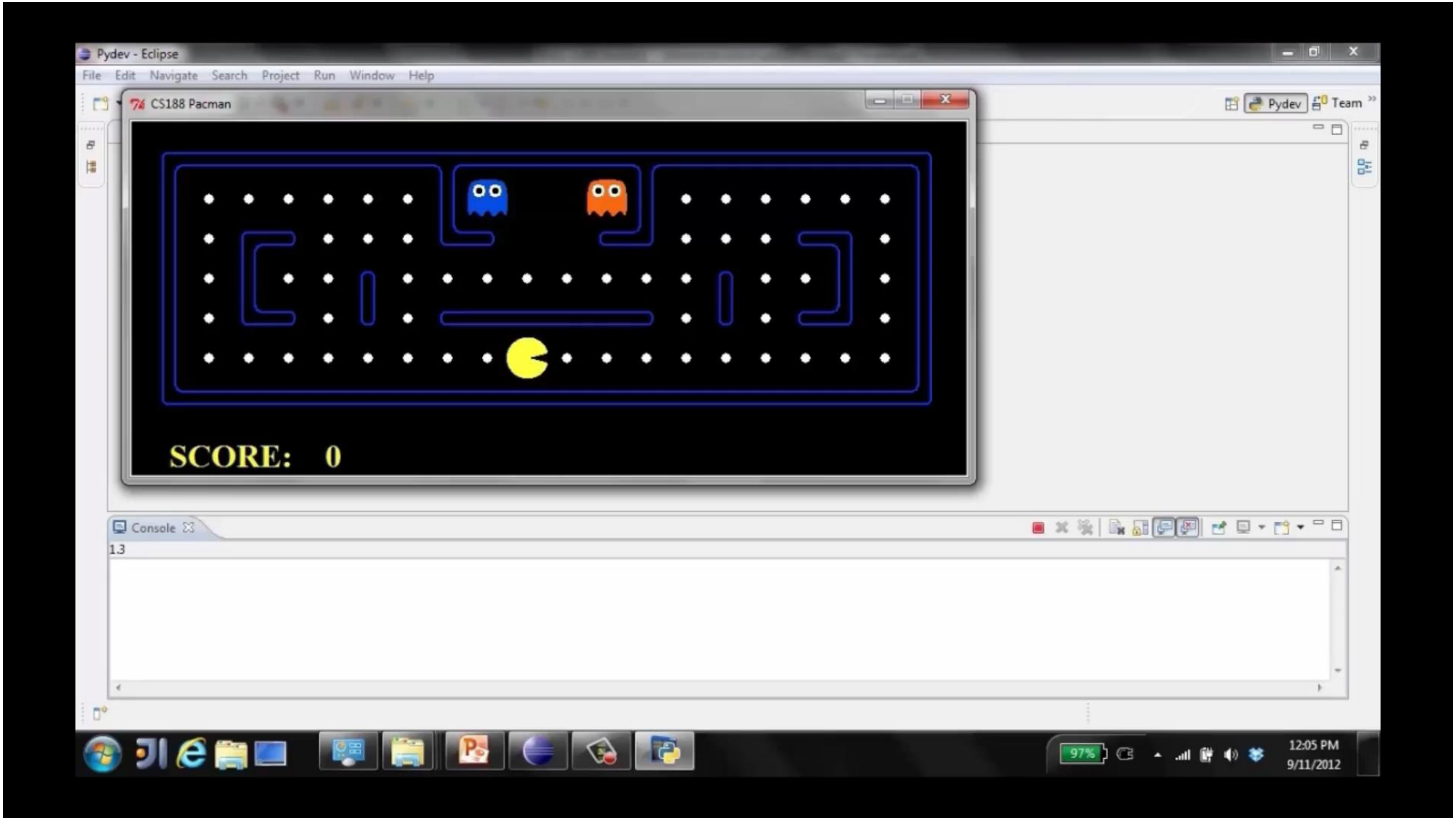
- **Función ideal**: devuelve el valor real de minimax en esa posición hasta estado terminal
- **En la práctica**: normalmente suma lineal ponderada de características (features)

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

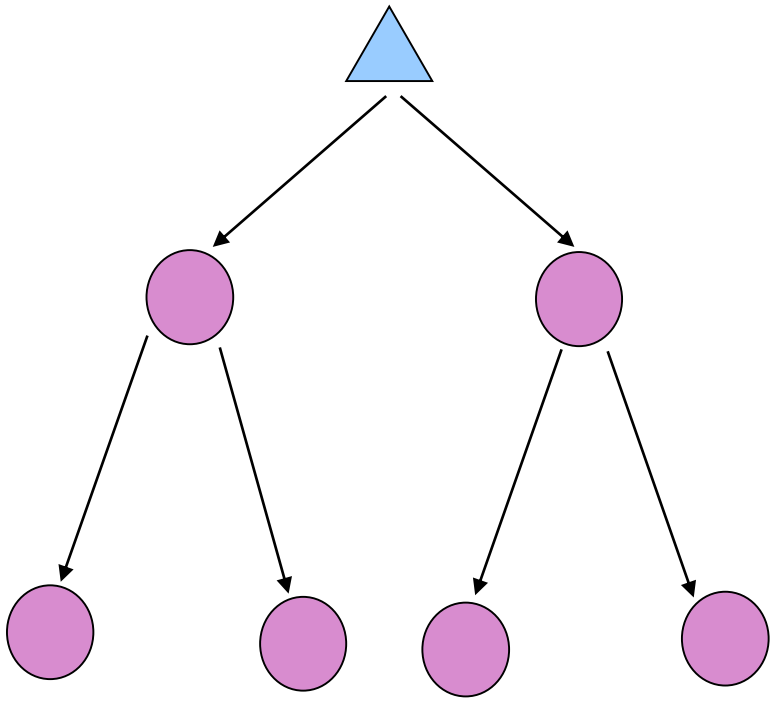
- Por ejemplo:

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

Vídeo Demo Fantasmas inteligentes (Coordinación)



MiniMax (coordinado)



MAX

MIN

MIN

MAX

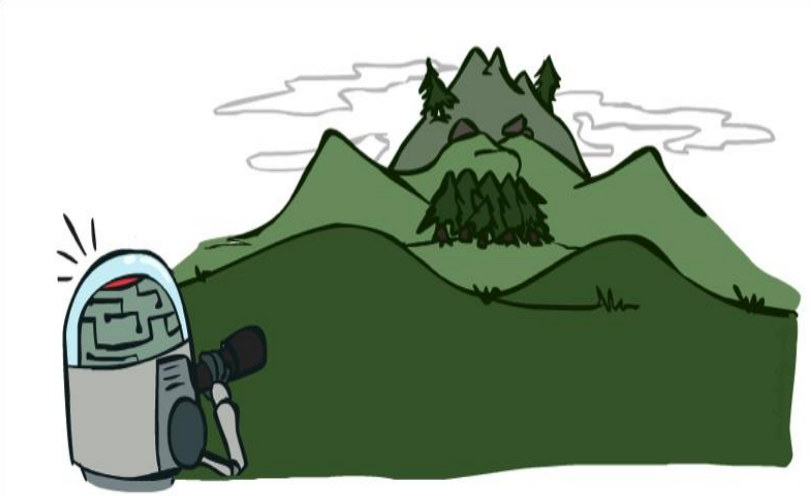
MIN

MIN

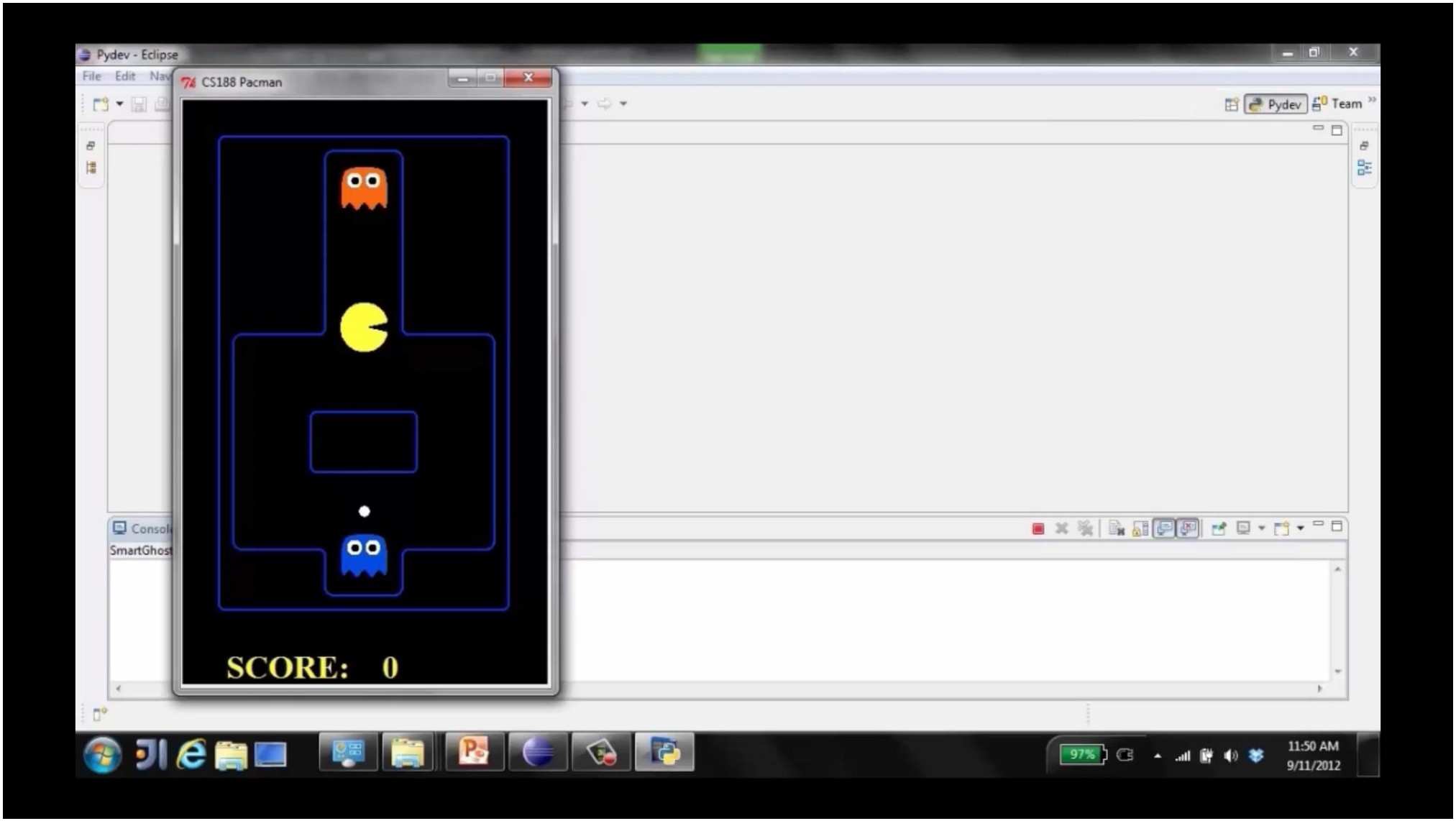
La profundidad es importante

- Las **funciones de evaluación son** siempre **imperfectas**

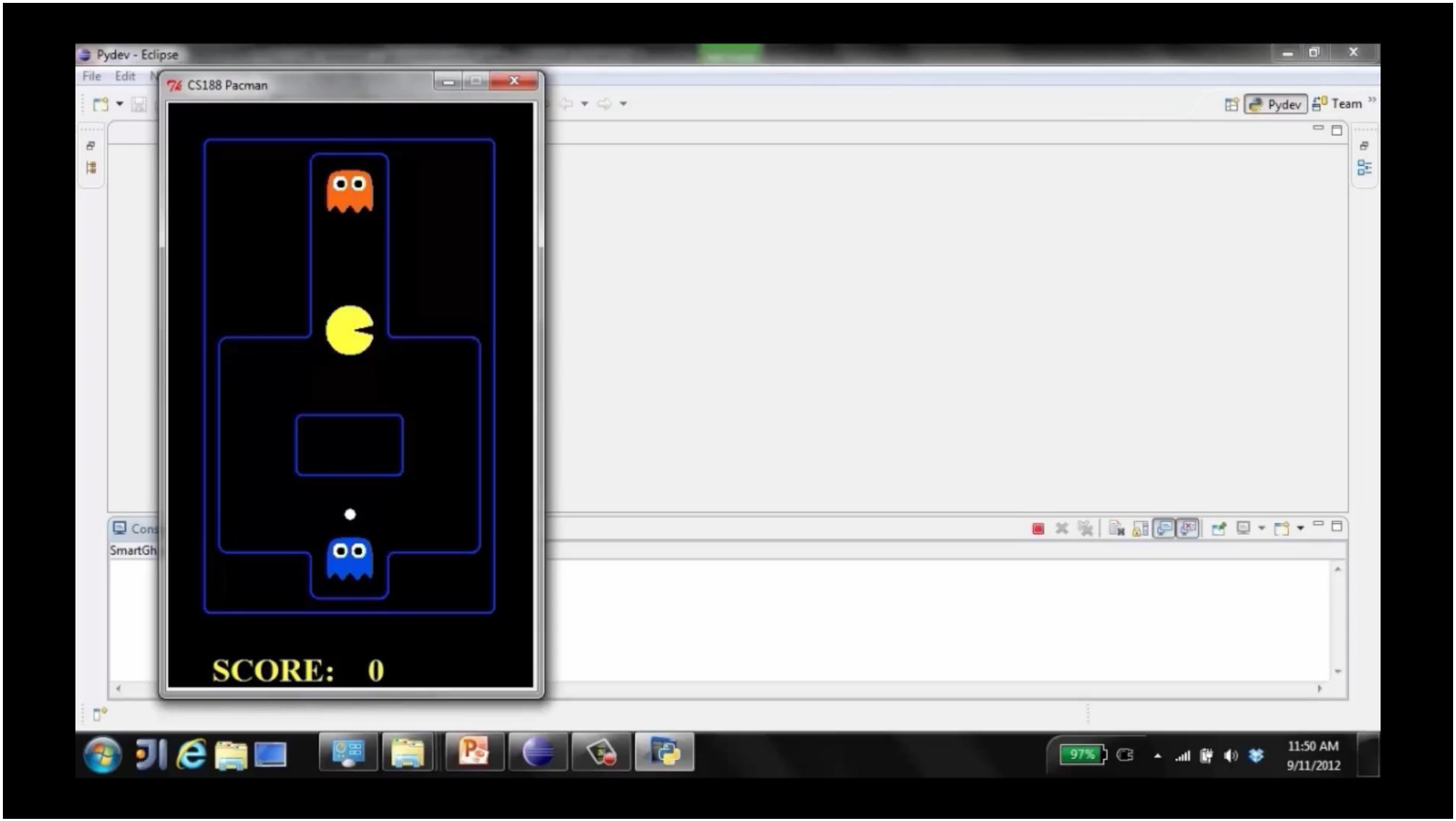
- Cuanto **más profundamente en el árbol probemos la función de evaluación**, es menos importante la calidad de la función de evaluación



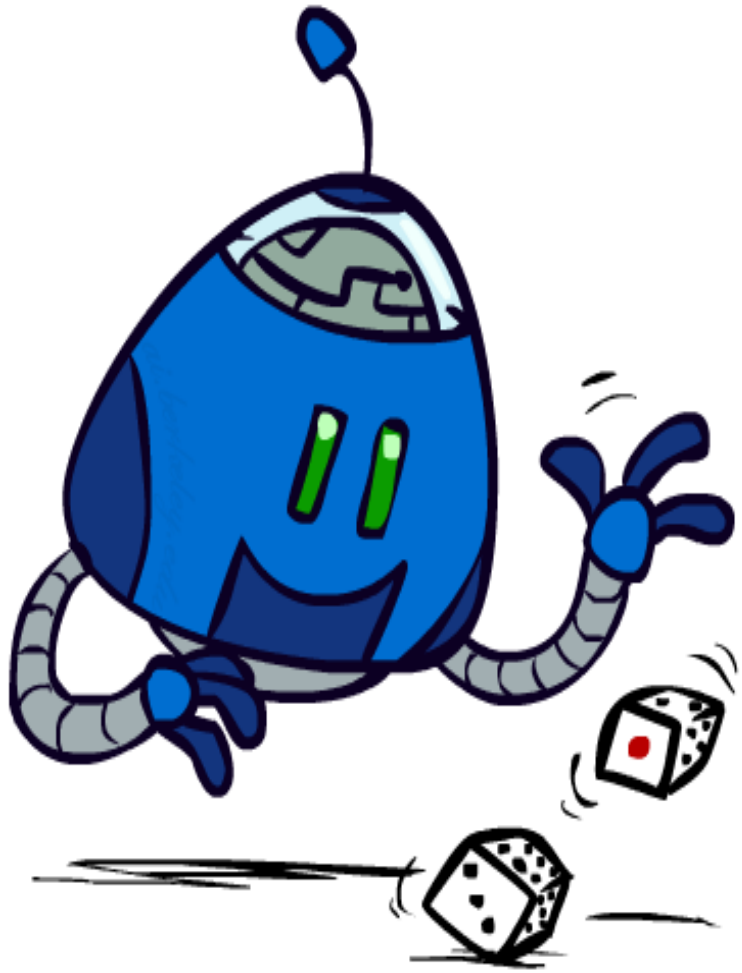
Vídeo Demo Profundidad limitada (d=2)



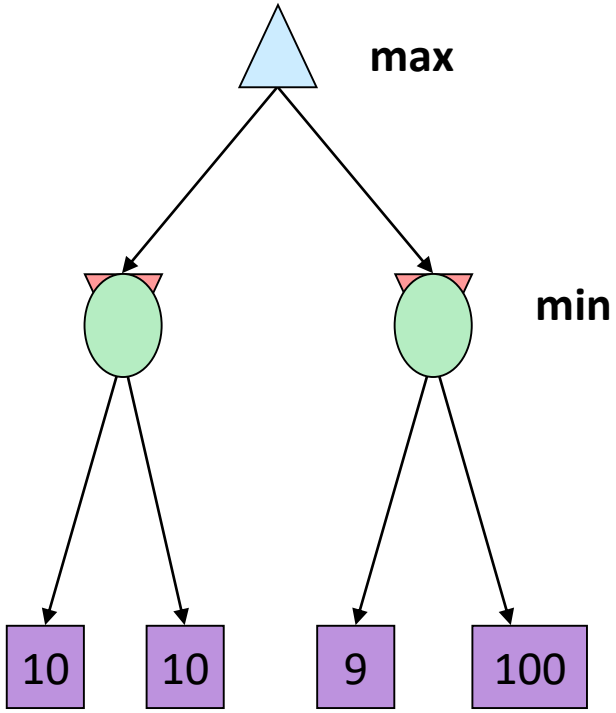
Vídeo Demo Profundidad limitada (10)

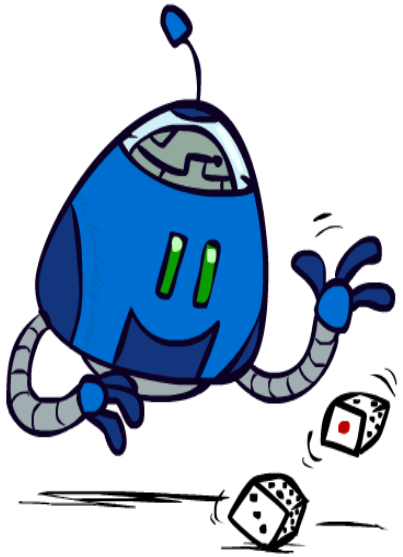


Resultados inciertos: juegos estocásticos



Caso peor vs. caso medio

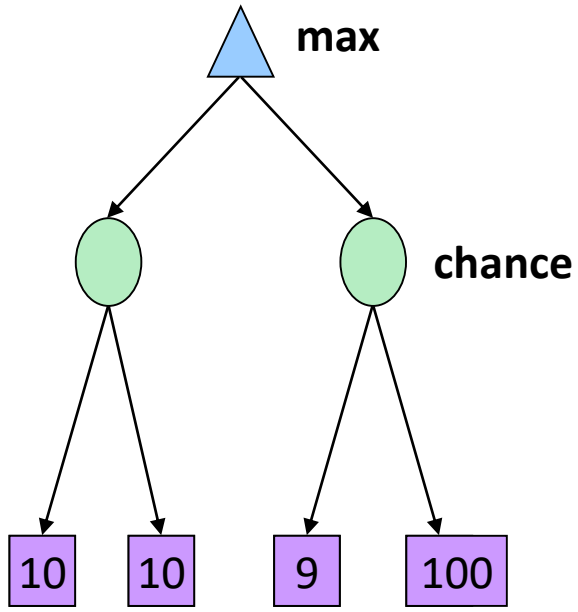




Idea: los resultados inciertos están controlados por el azar, no por el adversario!

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



- **Búsqueda expectimax**: calcular la puntuación media con un juego óptimo
  - Estados MAX como en búsqueda minimax
  - Los estados aleatorios son como los estados MIN pero el resultado es incierto
  - Se calcularán las utilidades esperadas
    - Por ej: Tomar la media ponderada de los hijos (expectativa)

Pseudocódigo de Expectimax

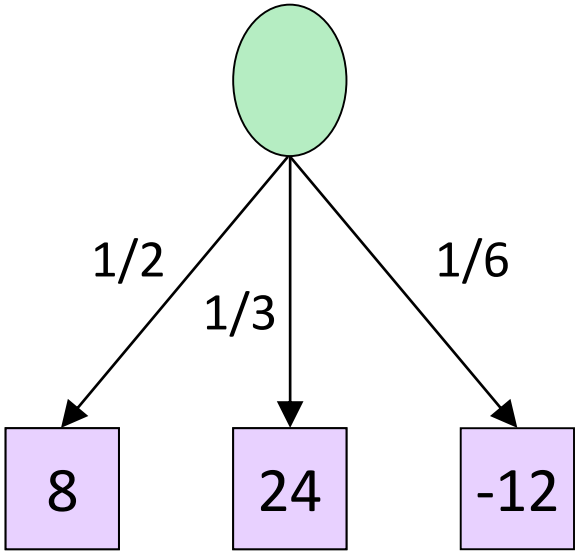
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

```
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
```

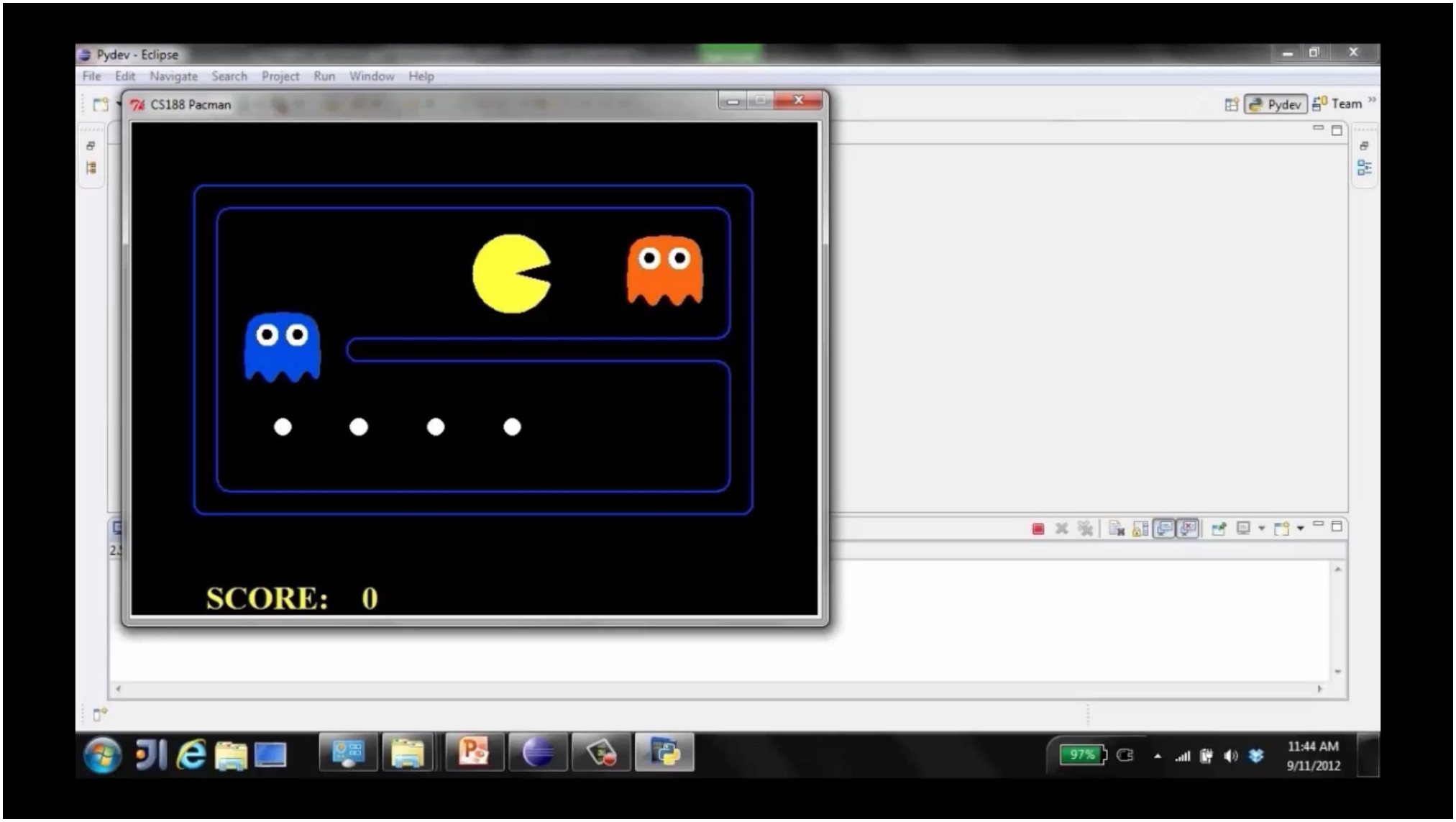
Pseudocódigo de Expectimax

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

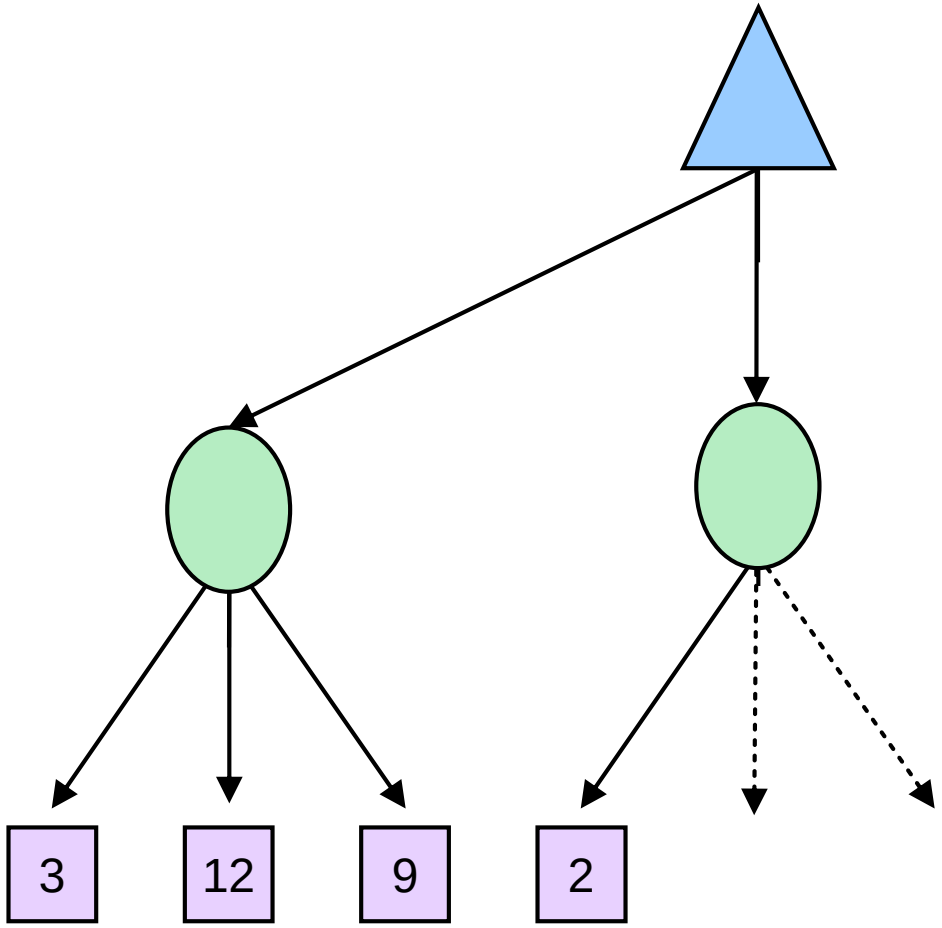


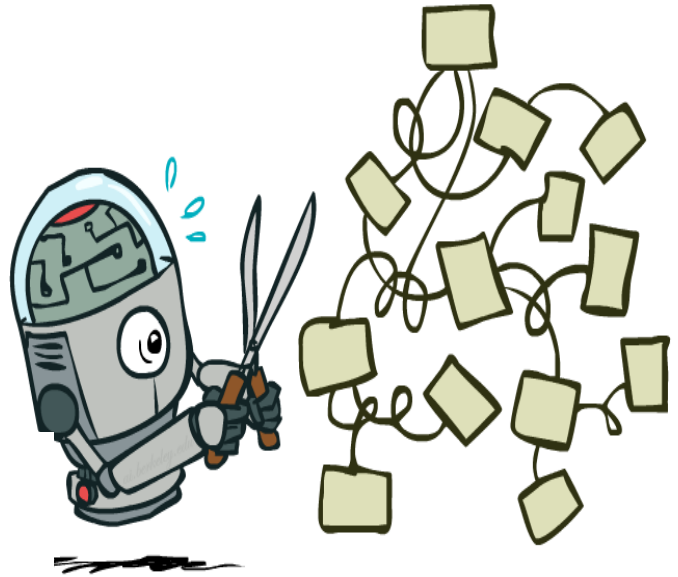
$$v = (1/2)*(8) + (1/3)*(24) + (1/6)*(-12) = 10$$

Vídeo Demo Expectimax (fantasmas aleatorios)



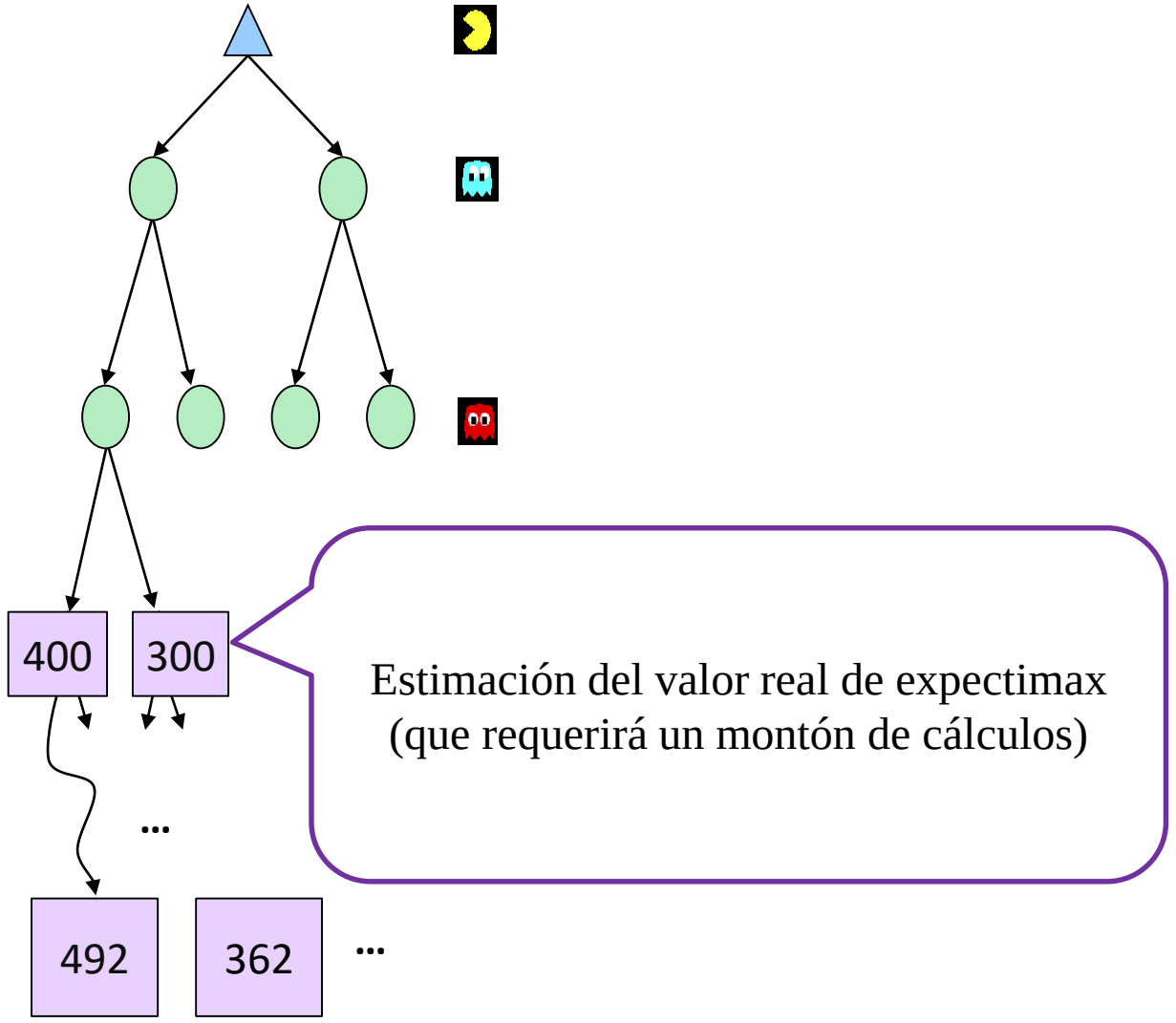
¿Poda de Expectimax?



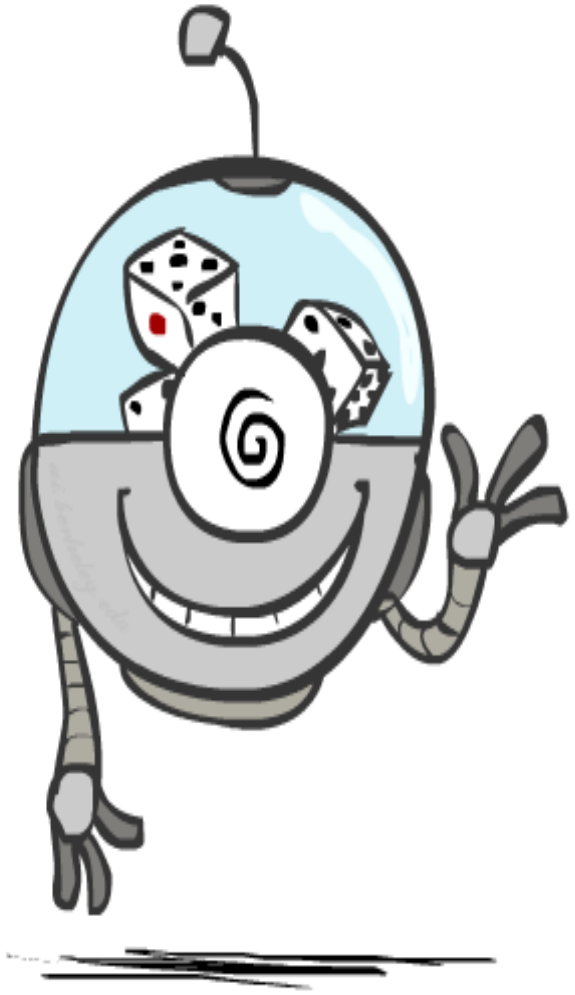


No se puede podar! Ya que se debe analizar todos los hijos!

Expectimax con profundidad limitada

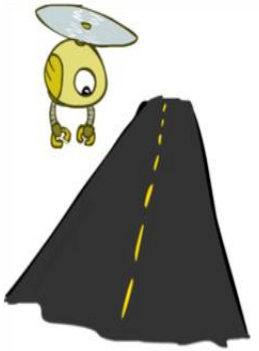


Probabilidades



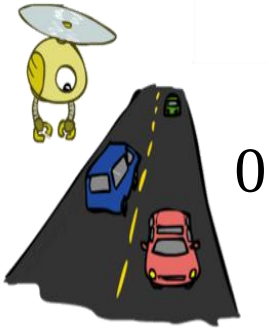
Recuerdo: Probabilidades

- 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



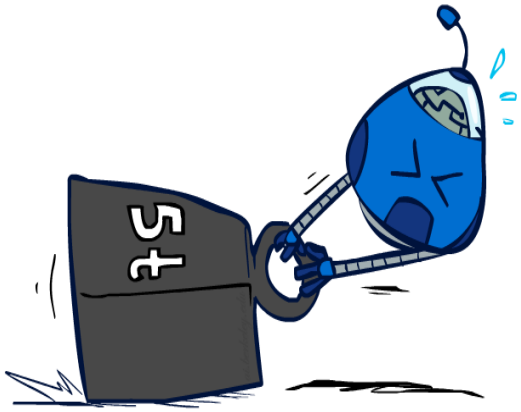
0.50

- 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

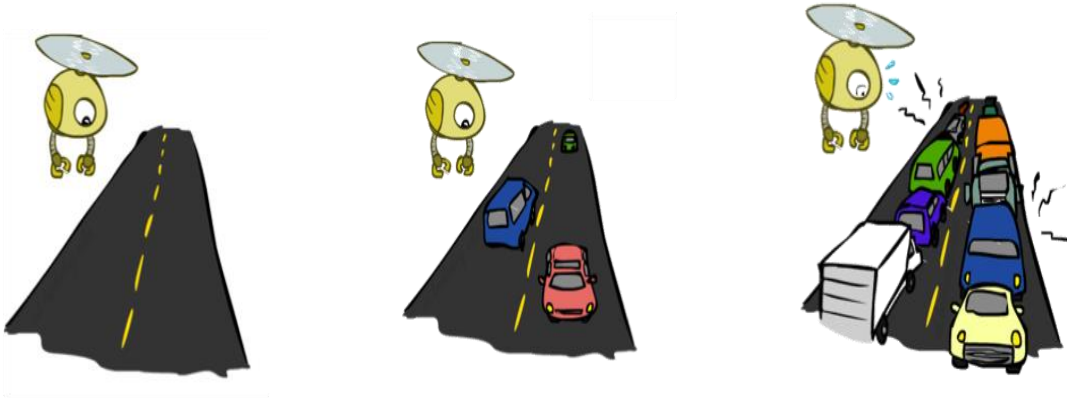
- El valor esperado de una función de una variable aleatoria es la media ponderada por la distribución de probabilidad de los resultados



- Ejemplo: **¿Cuánto tardaré en llegar al aeropuerto?**

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

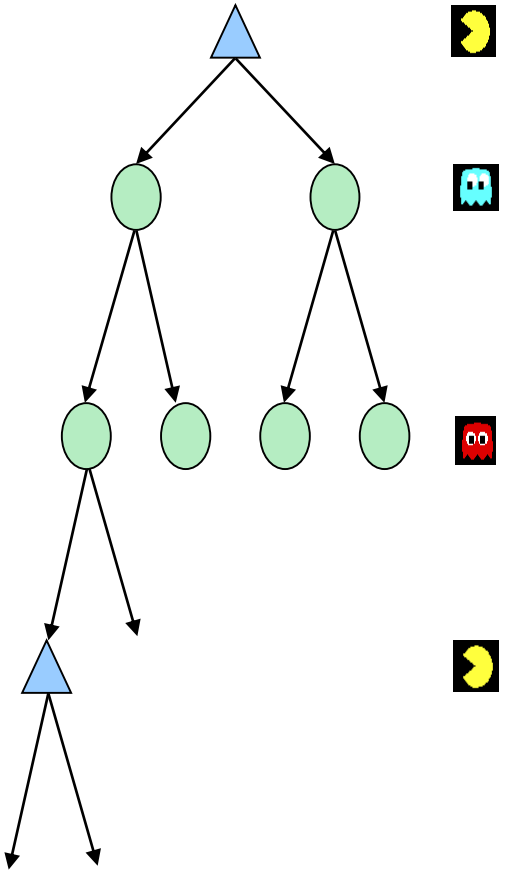
35 min



¿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!

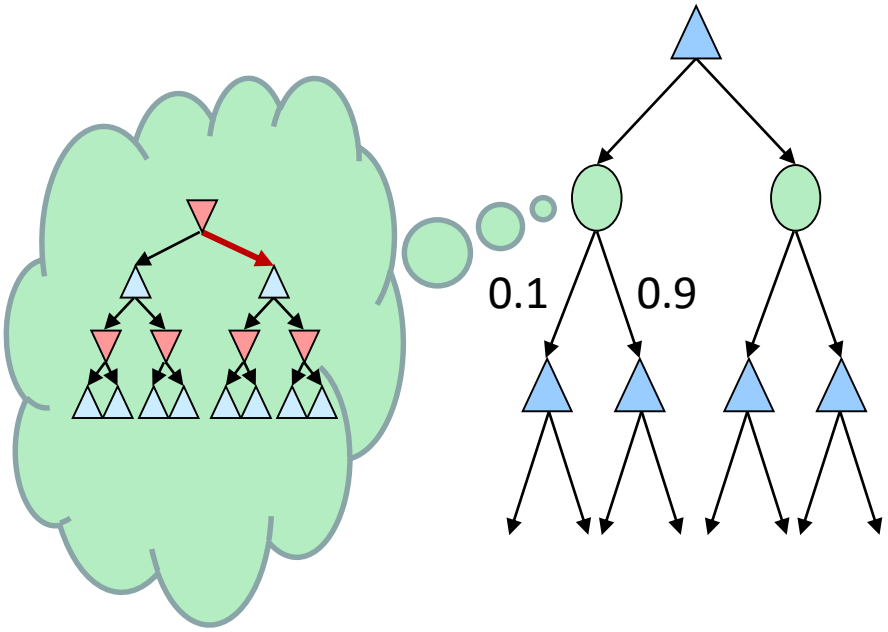
- Por ahora, asumiremos que cada nodo **viene mágicamente con probabilidades** que especifican la distribución respecto a sus valores



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

Quiz: Probabilidades informadas

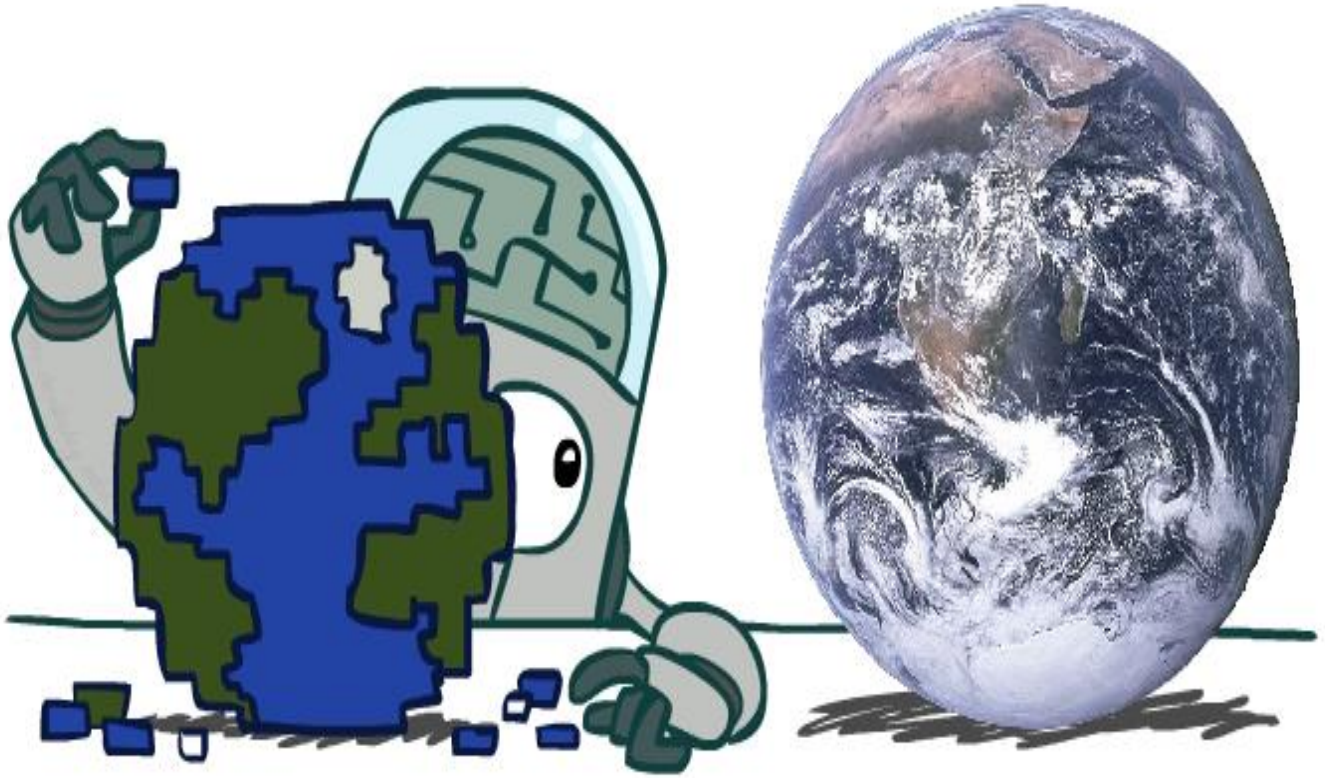
- Supongamos que nuestro oponente está ejecutando un minimax de profundidad 2, usando ese resultado un 80% de las veces, y moviéndose aleatoriamente en otro caso
- Pregunta: ¿Qué tipo de búsqueda en árbol usaríamos?



- 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 ...

Modelando asunciones



Los peligros del optimismo y el pesimismo

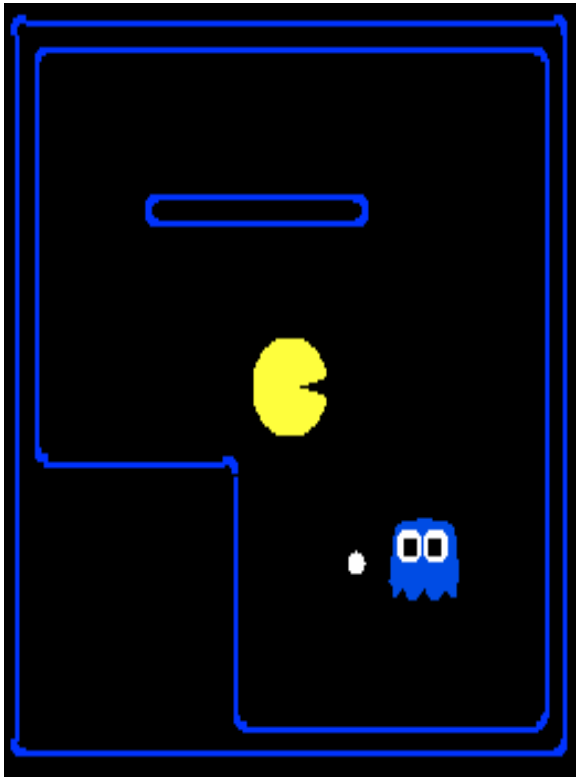
Optimismo peligroso

Asumiendo azar cuando el mundo es adversarial

Pesimismo peligroso

Asumiendo el caso peor cuando es poco probable

Asunciones vs Realidad

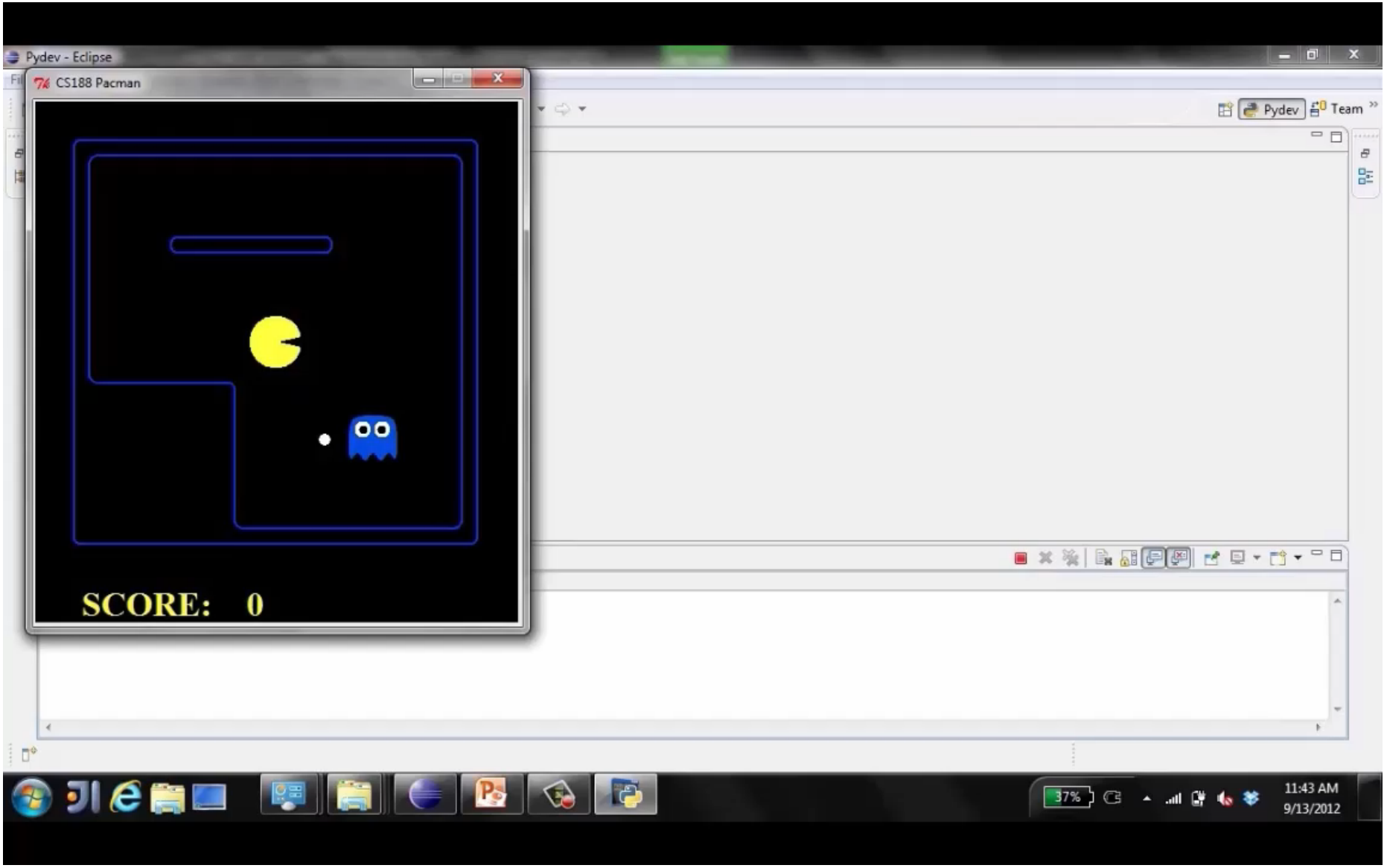


	Fantasma Adversarial	Fantasma Aleatorio
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

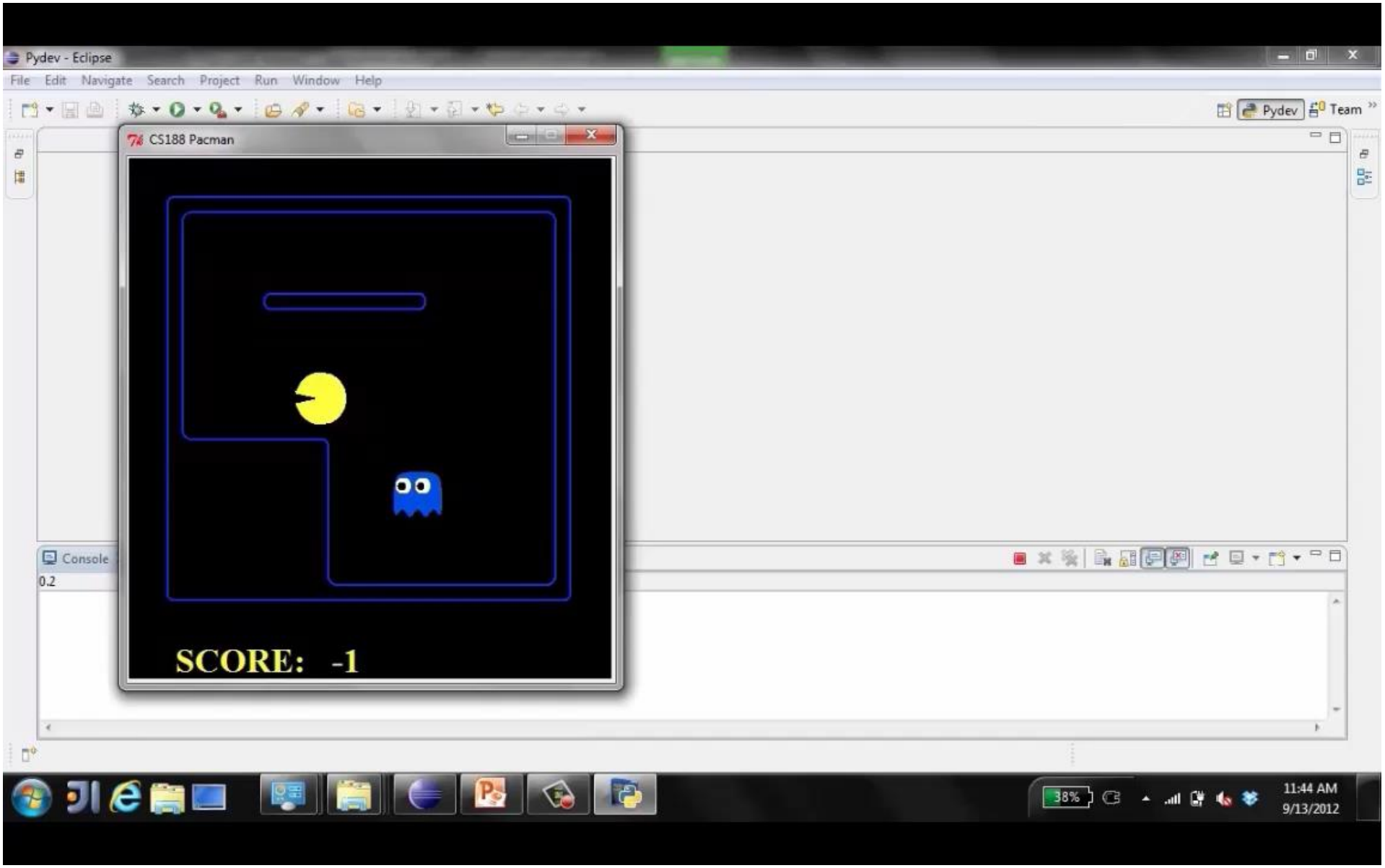
Resultados jugando 5 partidas

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

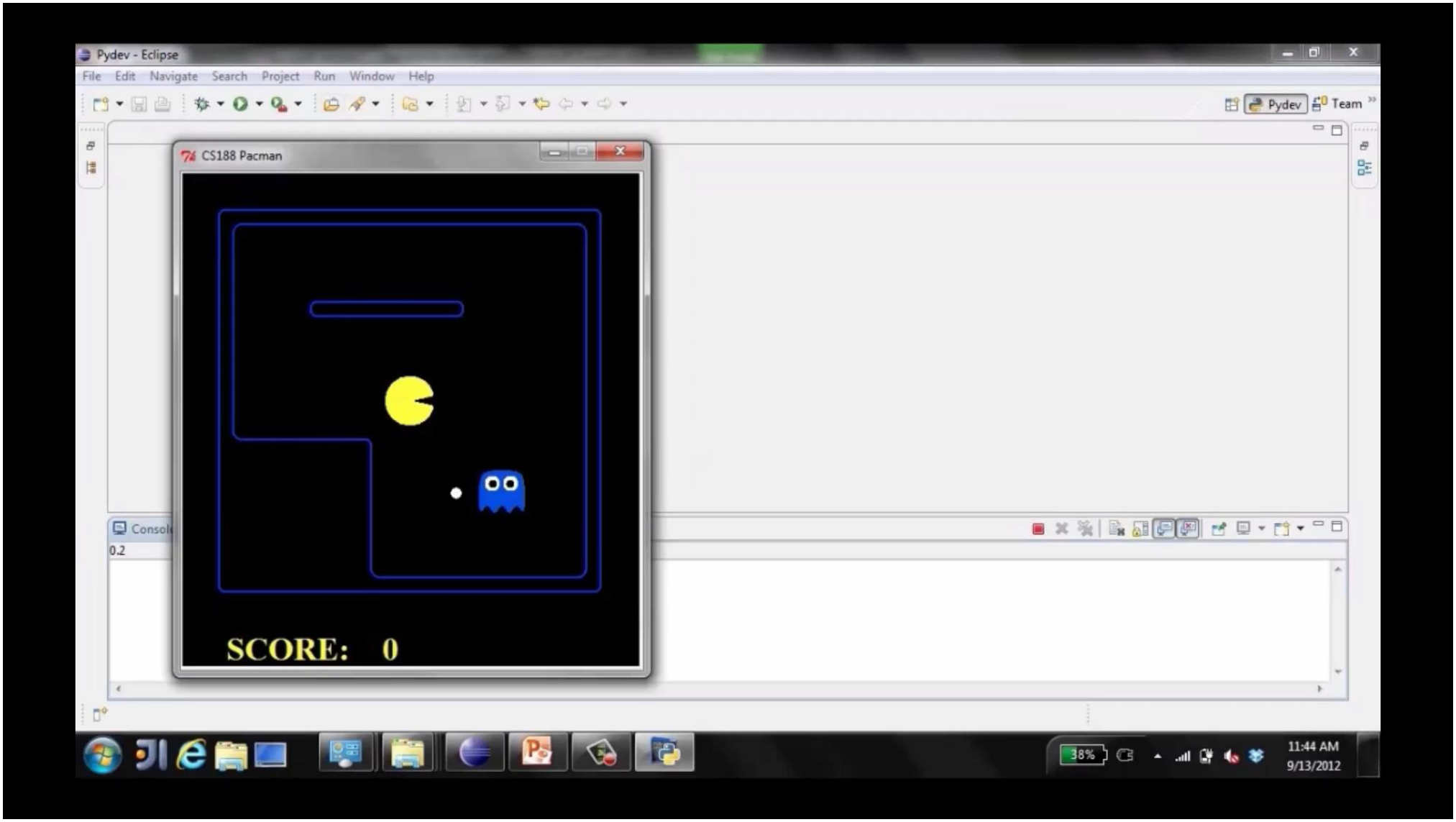
Vídeo demo asunciones sobre el mundo Fantasma aleatorio – Pacman Expectimax



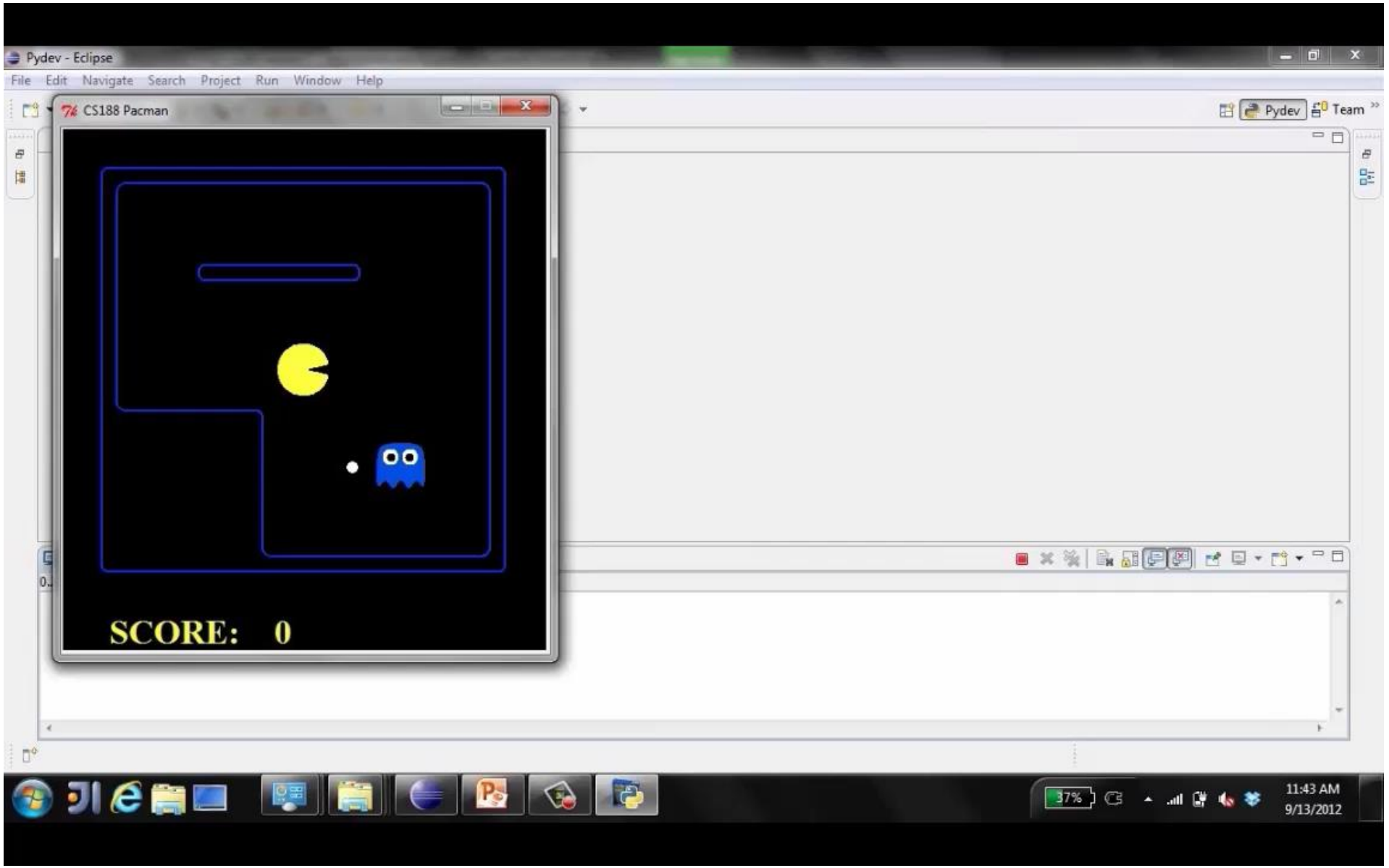
Vídeo demo asunciones sobre el mundo Fantasma aleatorio – Pacman Minimax



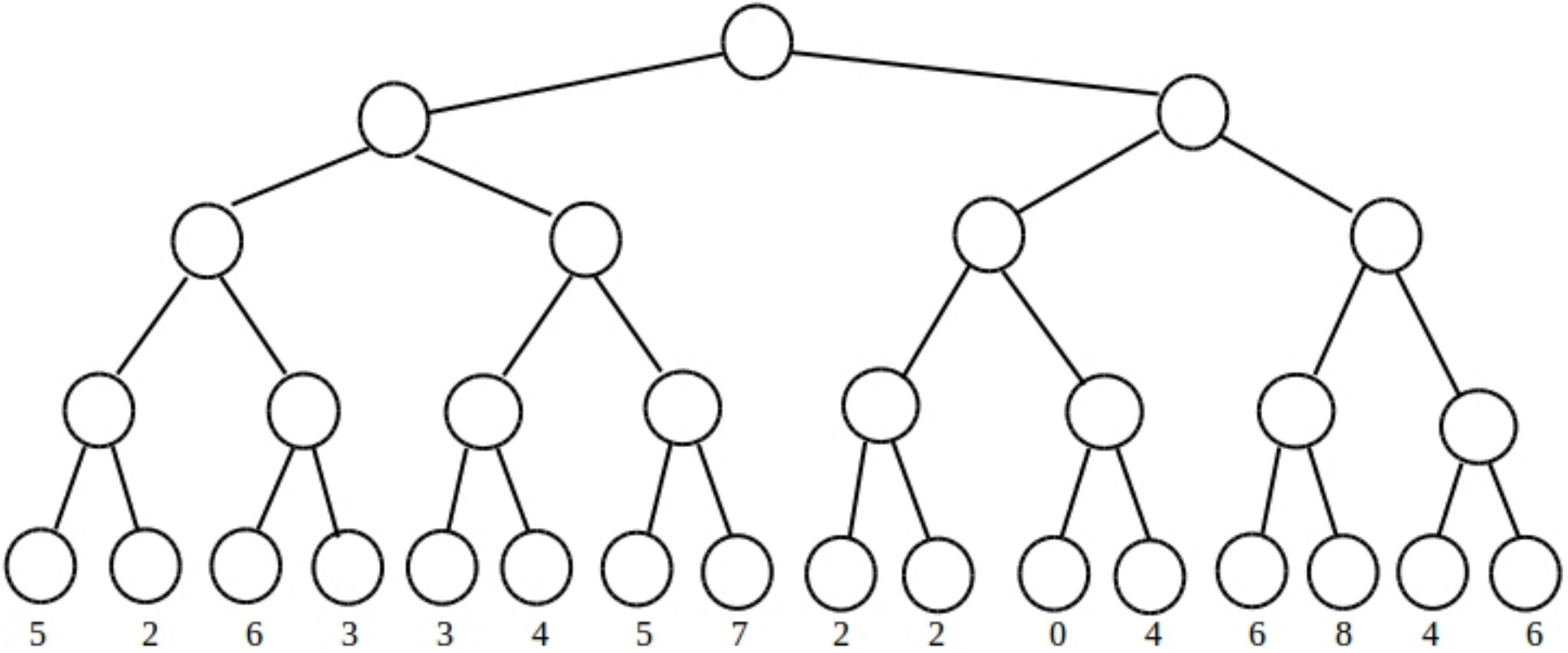
Vídeo demo asunciones sobre el mundo Fantasma adversarial – Pacman Expectimax



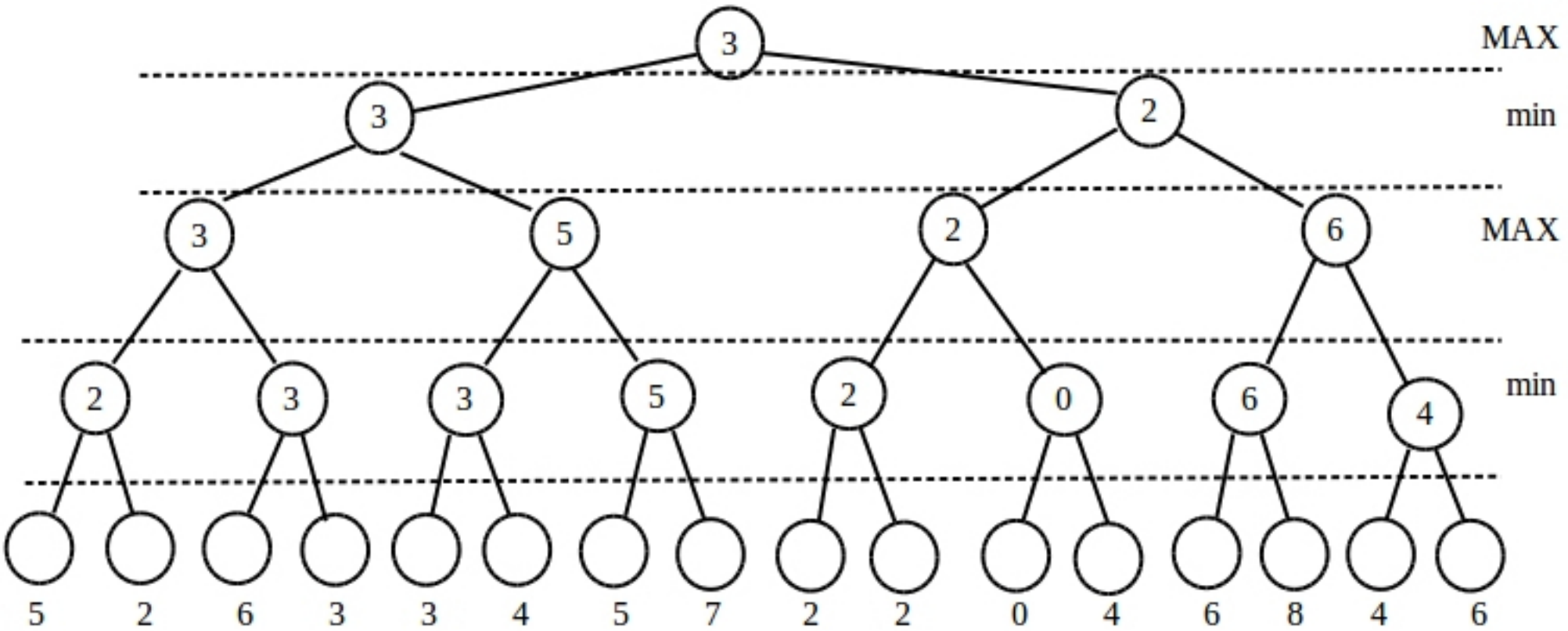
Vídeo demo asunciones sobre el mundo Fantasma adversarial – Pacman Minimax



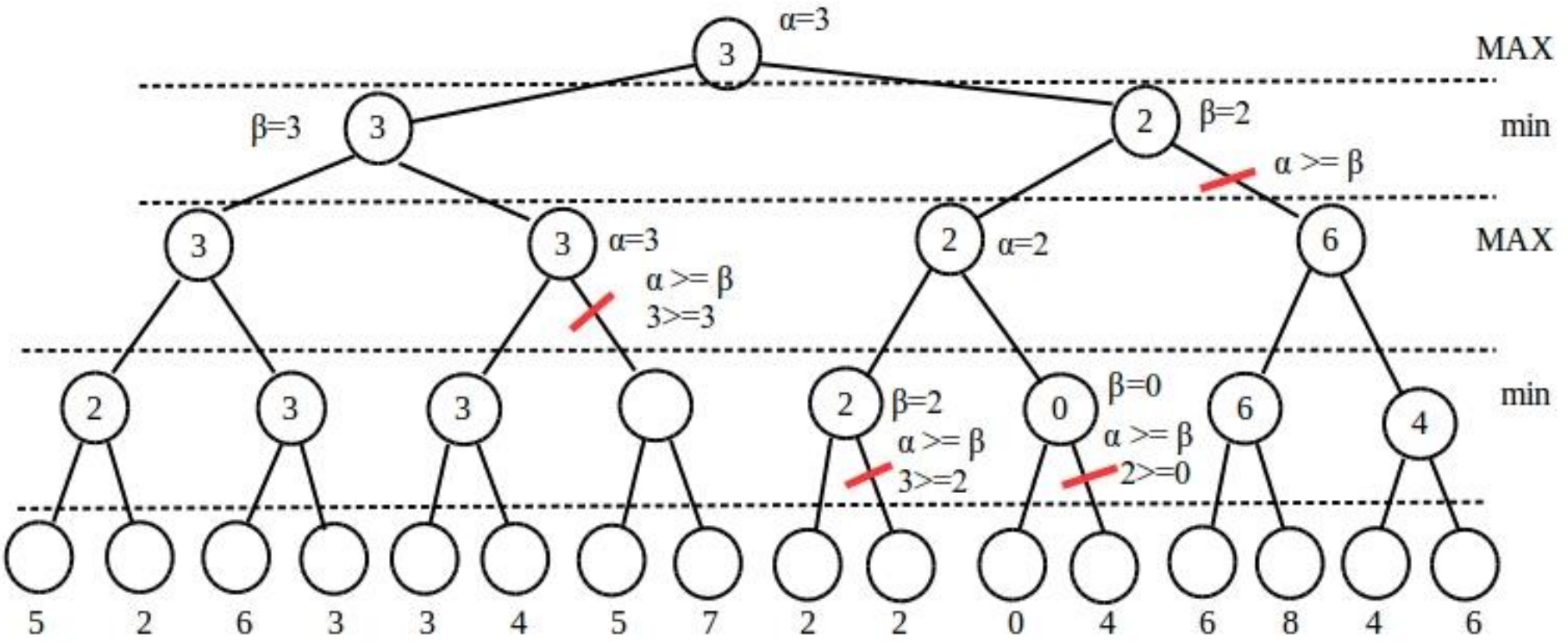
Ejercicios: MiniMax y α – β



Solución: MiniMax

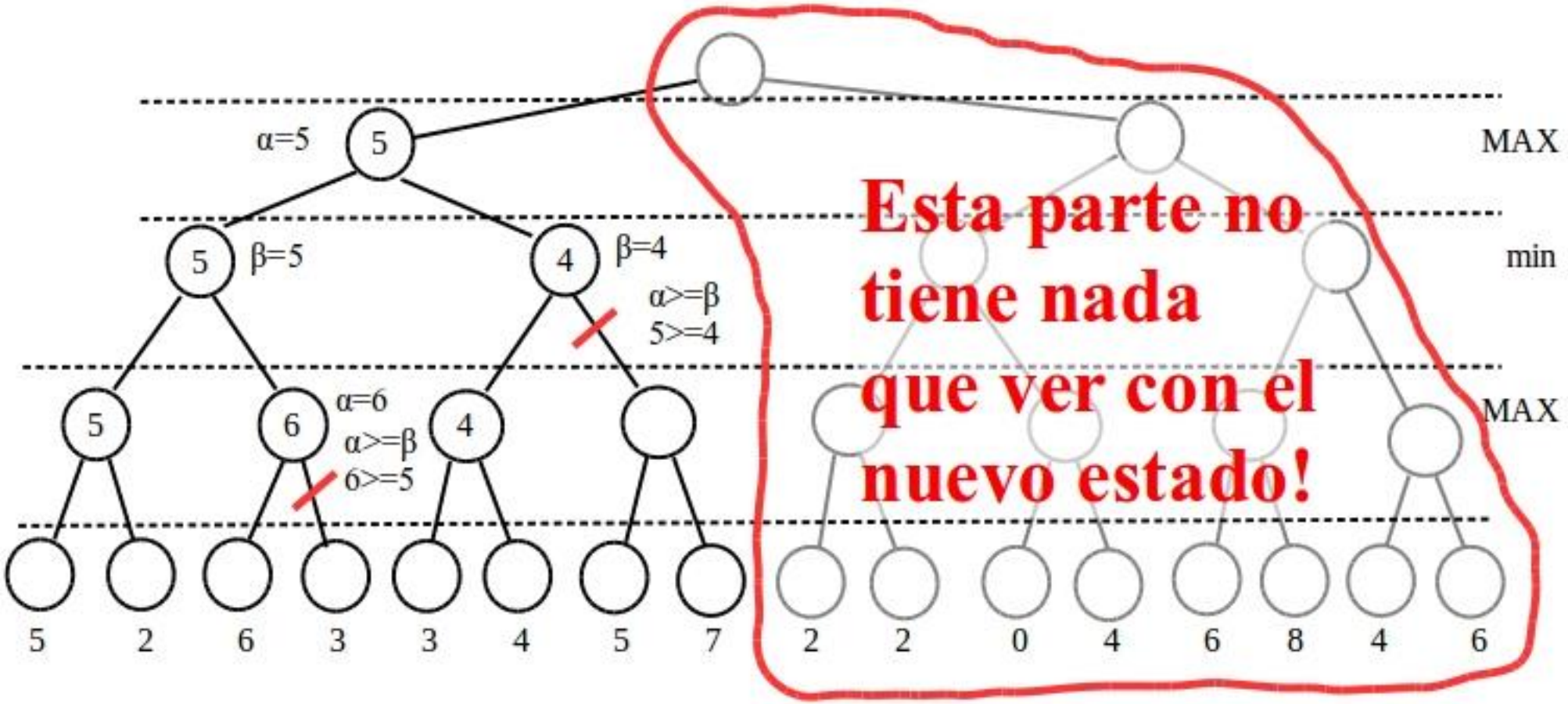


Solución: α – β

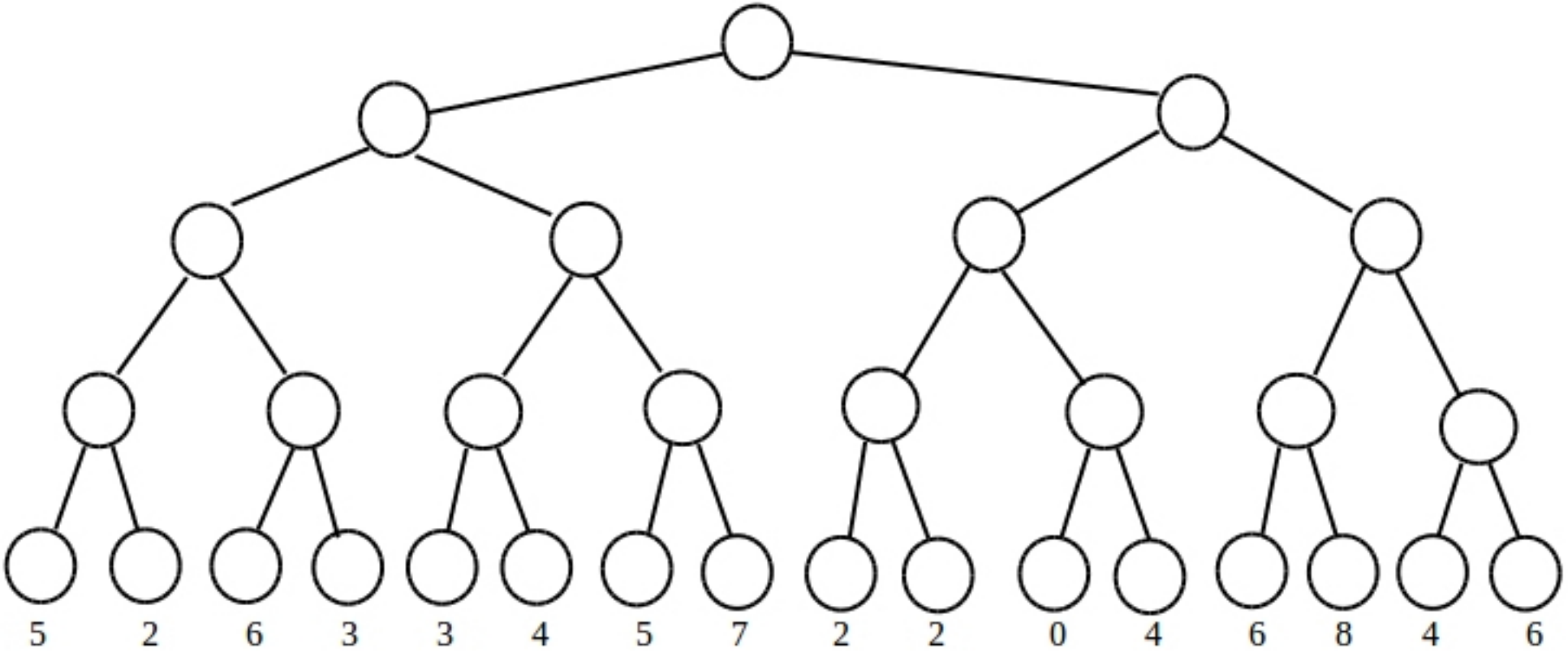


- Teniendo en cuenta tu elección … y que el oponente utiliza la misma función de evaluación (heurístico), cual sería el movimiento que haría el oponente?

Solución: α – β


α=5
5
β=5
5
4
β=4
α>=β
5>=4
5
6
α=6
α>=β
6>=5
4
5 2 6 3 3 4 5 7
2 2 0 4 6 8 4 6
MAX
min
MAX
Esta parte no tiene nada que ver con el nuevo estado!

Ejercicios: MiniMax con 3 jugadores?



Solución: MiniMax con 3 jugadores