Razonamiento y Planificación Automática

César Augusto Guzmán Álvarez

Doctor en Inteligencia Artificial

Tema 7 : Búsqueda multiagente

Sesión 1/2

unir LA UNIVERSIDAD EN INTERNET

Universidad Internacional de La Rioja

Resumen – Tema anterior

Tema 6 : Búsqueda heurística

Sesión 1 :

- Que es una heurística ?
- Búsqueda A*
- Practica del Puzzle-11

Sesión 2 :

- Búsqueda por subobjetivos
- Búsqueda online
- Practica del Laberinto

1		3
7	2	4
6	8	5

Problema del Puzzle-8

Índice

Sesión 1 :

- Asunción de los problemas a resolver
- Búsqueda minimax

Sesión 2 :

- La poda alfa-beta
- Búsqueda expect minimax

Asunción de los problemas a resolver

- Entorno ***único***
- Cada agente ***controla*** la ejecución de sus acciones.
- Las acciones ***influyen*** en la decisión de los otros agentes
- Un agente "puede" ***predecir*** acciones de los demás

Damas

Go Game

Parchis o parques

Monopoly Zelda.

Blackjack

Ajedrez

juegos ***bipersonales***
con *información* ***perfecta***
con y sin *elementos de azar*

Fuente: https://fortunerico.com/2016/01/27/google-deepmind-chinese-go/
https://es.wikipedia.org/wiki/Damas#/media/Archivo:International_draughts.jpg

Fuente: https://www.amazon.es/Monopoly-22927-Legend-of-Zelda/dp/B00OOMCDU0

Búsqueda minimax – historia y definición

minimizar la pérdida *máxima* esperada en juegos con adversario y con información perfecta

John von Neumann demostró que en juegos de **suma cero** con **información perfecta** entre **dos** competidores existe una **única solución óptima**.

"Un juego es una situación conflictiva en la que uno debe tomar una decisión sabiendo que los demás también toman decisiones, y que el resultado del conflicto se determina, de algún modo, a partir de todas las decisiones realizadas."

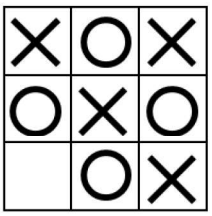
También afirmó que

"Siempre existe una forma racional de actuar en juegos de dos participantes, si los intereses que los gobiernan son completamente opuestos."

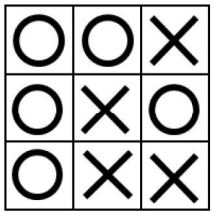
Búsqueda minimax – Ejemplo

Tres en Raya:

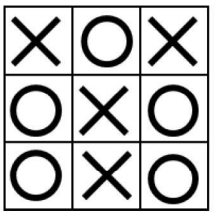
- dos jugadores (*min* y *max*)
- los jugadores van poniendo fichas en las casillas de un tablero 3x3
  - *max* usa las fichas **X** / *min* usa las fichas **O**
  - una casilla puede contener como mucho una ficha
- Reglas:
  - Inicialmente el tablero está vacío
  - *max* empieza y los jugadores se van alternando en poner sus fichas
  - *max* gana si obtiene una raya de tres fichas **X**
  - *min* gana si obtiene una raya de tres fichas **O**
  - si todas las casillas están ocupadas sin que haya una raya de 3 fichas del mismo tipo, hay empate



gana *max*
10

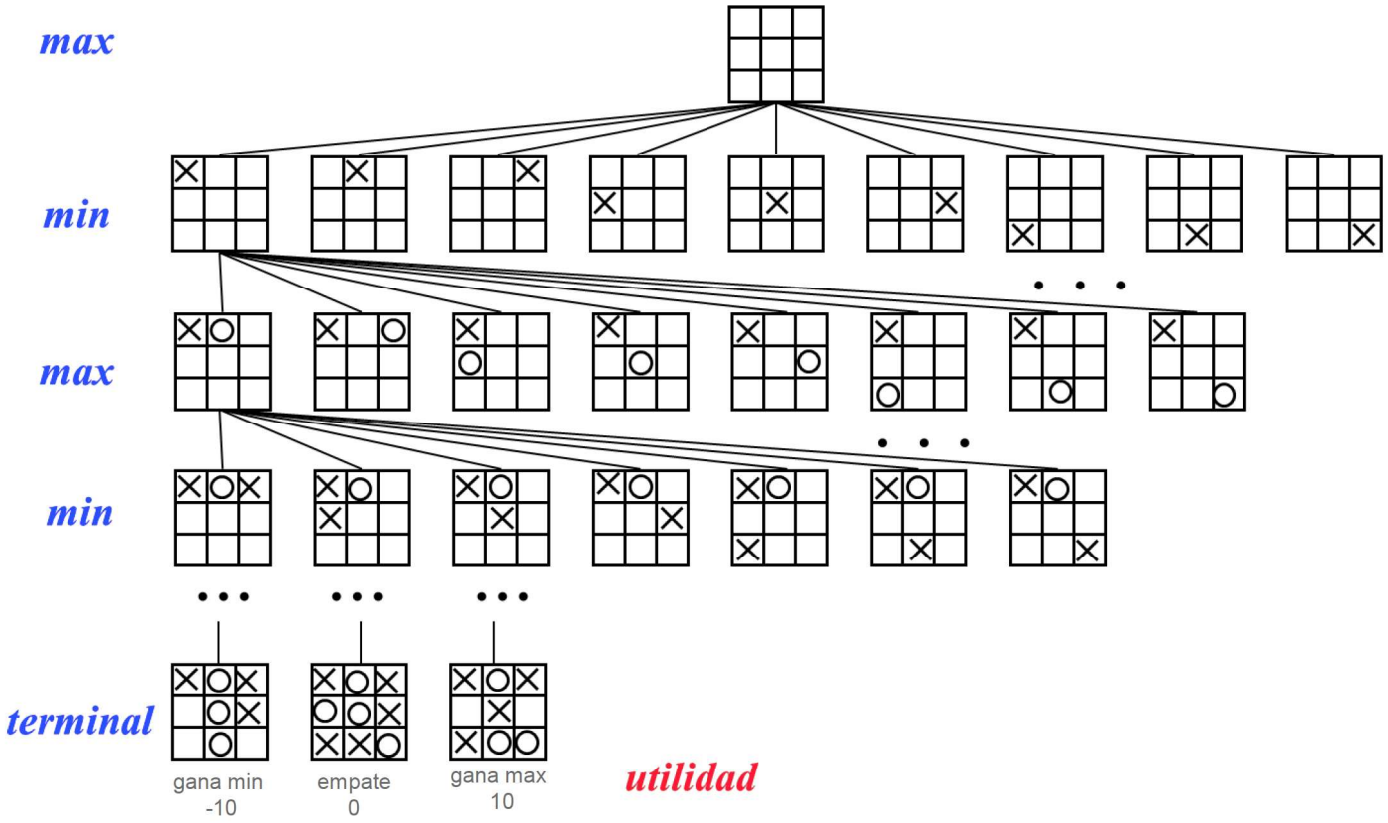


gana *min*
-10



empate
0

Búsqueda minimax – Árbol tres en raya



Búsqueda minimax

Conocimientos mínimos (min y max):

➢ s_0	Estado inicial
➢ *expandir(s)* : $\{s_1, ..., s_n\}$	Conjunto finito de estados sucesores
➢ *esTerminal(s)* : true | false	Si es un nodo terminal
➢ *utilidad(s)*: k, $k \in \mathbb{R}$	función parcial de utilidad del juego

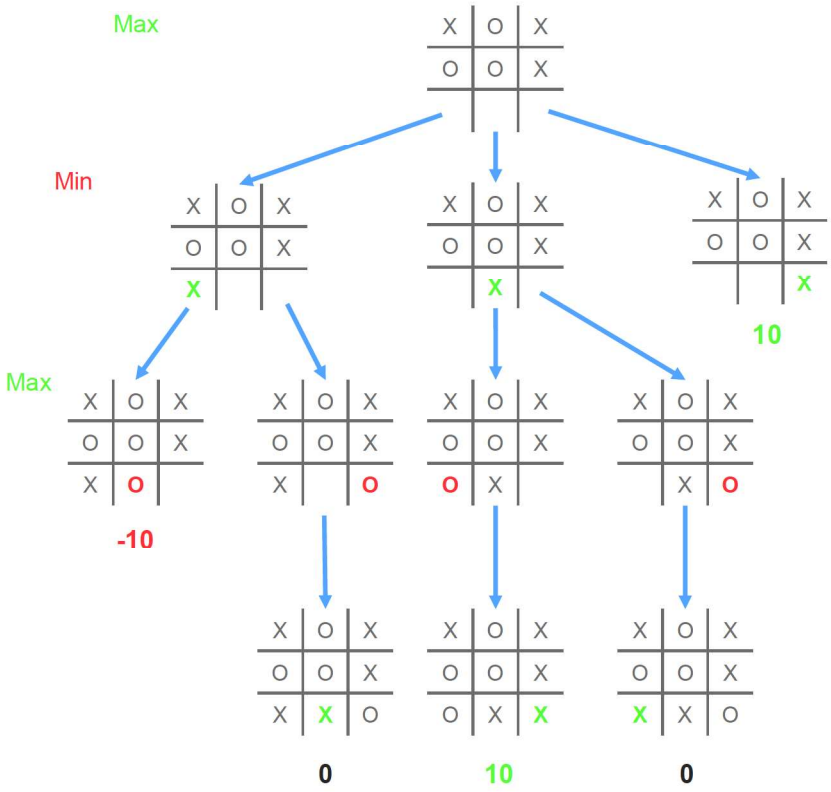
- *expandir(s):*
  - codifica las jugadas (acciones) permitidas en un estado *s*
  - supone que los jugadores se alternan en realizar las jugadas

- *utilidad(s)* : está definida sólo en los estados terminales
  - gana *max* : *utilidad*(*s*) = 10 : número real positivo
  - gana *min* : *utilidad*(*s*) = -10 : número real negativo
  - empate : *utilidad*(*s*) = 0 : número real neutro

Búsqueda minimax – Algoritmo general

1. Generar el árbol de juego completo
2. Aplicar la función de utilidad en cada nodo terminal
3. Propagar las utilidades hacia arriba
   - ❑ nodos *max*, se elige la utilidad máxima
   - ❑ nodos *min*, se elige la utilidad minima
4. Elegir la jugada óptima de *max* en el nodo raíz.
   - ❑ El sucesor del raíz con utilidad máxima

Búsqueda minimax – Tres en raya



Algoritmo general

1. Generar el árbol de juego completo
2. Aplicar la función de utilidad en cada nodo terminal
3. Propagar las utilidades hacia arriba
   - ❑ nodos *max*, se elige la utilidad máxima
   - ❑ nodos *min*, se elige la utilidad mínima
4. Elegir la jugada óptima de *max* en el nodo raíz.
   - ❑ El sucesor del raíz con utilidad máxima

Búsqueda minimax – Algoritmo Implementación

```
minimax(estado, esMax, [profundidad]):

1:  si esTerminal(estado) entonces
2:     retornar utilidad(estado)
3:  sucesores = expandir(estado)

4:  si esMax entonces
5:     mejorUtilidad = -INFINITO
6:     para cada s en sucesores :
7:           utilidad = minimax(s, false, [profundidad+1])
8:           mejorUtilidad = max( mejorUtilidad, utilidad)
9:     retornar mejorUtilidad

10:  sino
11:    mejorUtilidad = +INFINITO
12:    para cada s en sucesores :
13:          utilidad = minimax(s, true, [profundidad+1])
14:          mejorUtilidad = min(mejorUtilidad, utilidad)
15:    retornar mejorUtilidad
```

Búsqueda minimax – Optimización

- En la práctica minimax es impracticable
- Genera el árbol completo. Primero en profundidad.
- Complejidad $O(b^d)$ (factor de ramificación b; profundidad d)
- Realizar la búsqueda completa requiere cantidades excesivas de tiempo y memoria.
- Posibles soluciones:
  - Limitar la generación del árbol con heurísticas.
  - Limitar por nivel de profundidad del árbol
  - Limitar por tiempo de ejecución
  - Poda alfa-beta

Práctica - Búsqueda minimax

Fuente: https://www.codingame.com/ide/puzzle/tic-tac-toe

Practica - Búsqueda Minimax. Tic Tac Toe

índice fila y columna	0	1	2
0			
1			
2			

fila = 2
columna = 0
Índice lista = (3*fila) + columna
= (3*2) + 0
= 6

índice fila y columna	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

0	1	2	3	4	5	6	7	8
X		O						

Índice lista = 6
fila = 6 / 3 = 2
columna = 6 % 3 = 0

Gracias!