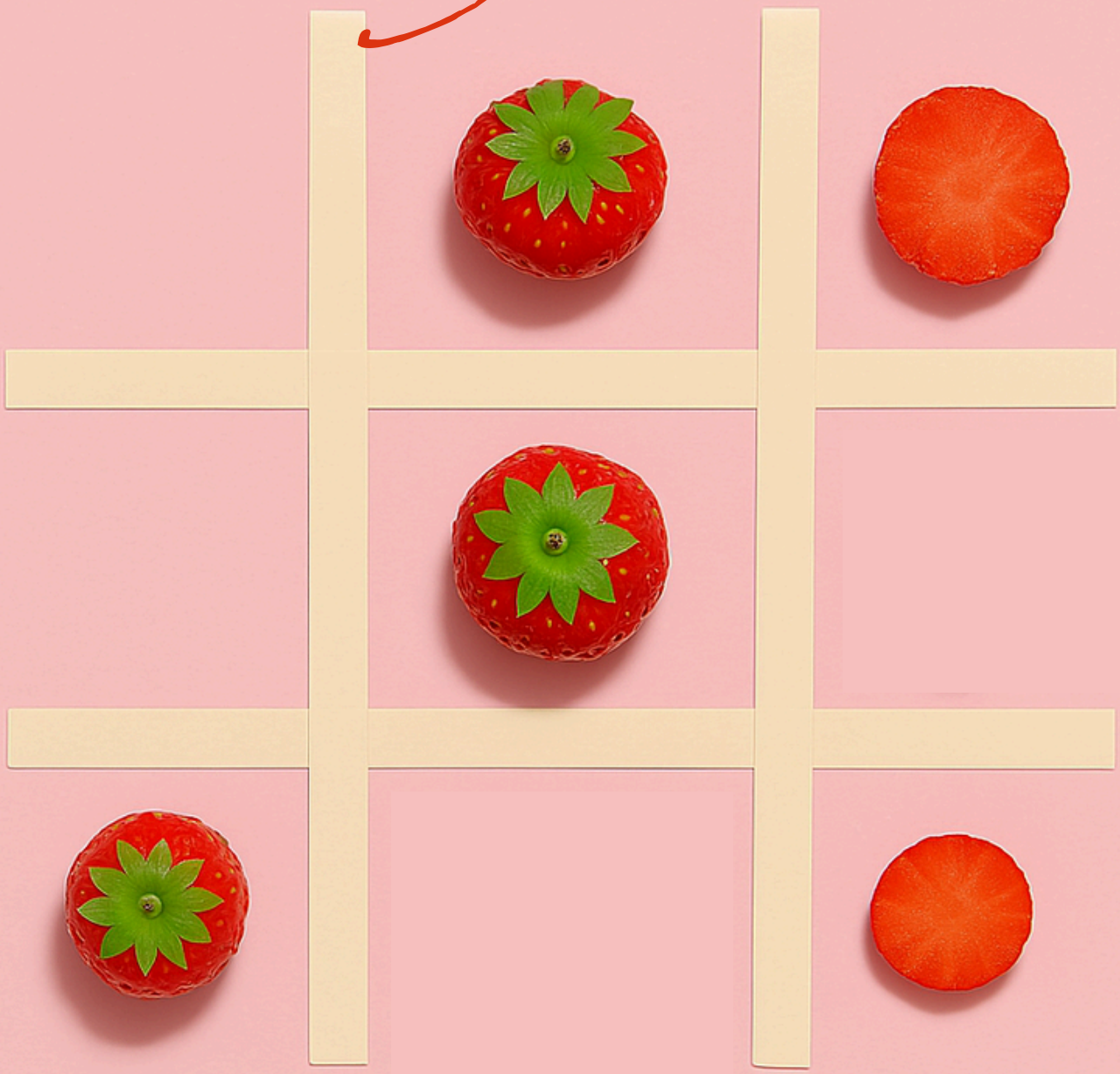
BÚSQUEDAS

de Adversario



Lucía Zamudio Cecilia
Práctica 3. IA 2025

Índice

1. **Introducción**

Los entornos competitivos dan ocasión a problemas de búsqueda entre adversarios, lo que se conoce como juegos. El estado de un juego es fácil de representar. En el caso de el tres en raya tiene un pequeño número de acciones cuyos resultados están definidos por reglas precisas.
Tic tac toe es un juego de suma cera, debido a que la ganancia de un jugador, en este caso el PC(+1) equivale exactamente a la pérdida de otro, el humano (-1), si sumamos las recompensas el resultado es 0.

OBJETIVO

Como objetivo hay que implementar dos tipos de algoritmos de búsqueda con adversarios, comparando su desempeño y resolución en los juegos. Además evaluar su efectividad.

2. **Descripción de los algoritmos**

ALGORITMO MINIMAX

Se pretende maximizar la ganancia del PC y minimizar la del jugador. Max mueve primero, y luego mueven por turno hasta que el juego termina. El estado inicial: incluye la posición del tablero e identifica al jugador que mueve, una función sucesor: devuelve una lista de pares (movimiento, estado), indicando un movimiento legal y el estado que resulta.; y un test terminal: también conocido como función objetivo o función de rentabilidad, da un valor numérico a los estados terminales.
Siendo la utilidad, 1 gana el PC, -1 gana el jugador y 0 empate:
En la función **public static int decisionMinimax(Estado estado)**, primero obtenemos el valor máximo del estado actual, y devolvemos su movimiento.

En la función **private static Par_movimientoValor maxValor(Estado estado),** primero se observa si es el el estado terminal, es decir un movimiento final de partida, si es así devuelve la utilidad. Se inicializa el valor con lo mínimo posible para encontrar el máximo. Y de entre todos los sucesores según lo que podría realizar el jugador, se escoge el valor máximo.

En la función **private static int minValor(Estado estado),** primero se comprueba si el estado es terminal y devuelve la utilidad, inicializa el valor con el valor máximo posible. Se llama a maxValor para ver que movimientos haría para cada uno de los posibles sucesores de min, y de todos esos valores se elige el menor posible. Finalmente devuelve ese valor.

ALGORITMO PODA ALFA-BETA

Cuando lo aplicamos a un árbol minimax estándar, devuelve el mismo movimiento que devolvería minimax, ya que podar las ramas no puede influir, posiblemente, en la decisión final. La poda alfa-beta puede aplicarse a árboles de cualquier profundidad, y, algunas veces, es posible podar subárboles enteros.

$\alpha \rightarrow$ el valor de la mejor opción (es decir, el valor más alto) que hemos encontrado hasta ahora en cualquier punto elegido a lo largo del camino para MAX.

• $\beta \rightarrow$ el valor de la mejor opción (es decir, el valor más bajo) que hemos encontrado hasta ahora en cualquier punto elegido a lo largo del camino para MIN.

En la función **public static int decisionPoda(Estado estado)** se inicializa los valores de alfa y beta con menos infinito y más infinito llama a maxValor() para determinar el mejor valor en el estado actual, y retorna el mejor movimiento.

La función **private static Par_movimientoValor maxValor(Estado estado, int alfa, int beta)** si el estado es terminal, se devuelve la utilidad y si no -1. inicializa el valor a menos infinito, recorre los sucesores, llamando a minValor() sobre cada sucesor, y si encuentra un valor mayor que el actual lo guarda junto al movimiento, si ese valor es >= beta, se poda, y se actualiza alfa con el mejor valor encontrado hasta el momento, finalmente se devuelve el mejor valor y el movimiento.

La función **private static int minValor(Estado estado, int alfa, int beta)** si el estado es terminal se devuelve -1, se inicializa v como menos infinito y se llama a maxValor() para cada sucesor, se actualiza v con el valor más pequeño y si el valor es más pequeño o igual que alfa se poda, y se actualiza beta con el mejor valo además de devolver el valor mínimo encontrado.

3. Resultados experimentales

Algoritmo MinMax(empate)

```
Jugador Humano: (X)
Jugador MinMax: (0)
- - -
- - -
- - -
Introduce movimiento
4
- - -
- X -
- - -
El tiempo de ejecucion es:40155
Nodos: 55505
Movimiento de la IA:
0 - -
- X -
- - -
Introduce movimiento
2
0 - X
- X -
- - -
El tiempo de ejecucion es:363
Nodos: 933
Movimiento de la IA:
0 - X
- X -
0 - -
```

```
Introduce movimiento
3
0 - X
X X -
0 - -
El tiempo de ejecucion es:20
Nodos: 51
Movimiento de la IA:
0 - X
X X 0
0 - -
Introduce movimiento
7
0 - X
X X 0
0 X -
El tiempo de ejecucion es:4
Nodos: 5
Movimiento de la IA:
0 0 X
X X 0
0 X -
Introduce movimiento
8
0 0 X
X X 0
0 X X
Partida finalizada.
La partida ha terminado en empate.
```

Algoritmo MinMax(gana)

```
1. Minimax
2. Poda Alfa-Beta
Opción (1 o 2): 1
Jugador Humano: (X)
Jugador MinMax: (0)
- - -
- - -
- - -
Introduce movimiento
4
- - -
- X -
- - -
El tiempo de ejecucion es:20129
Nodos: 55505
Movimiento de la IA:
0 - -
- X -
- - -
Introduce movimiento
2
0 - X
- X -
- - -
El tiempo de ejecucion es:254
Nodos: 933
Movimiento de la IA:
0 - X
- X -
0 - -
```

```
Introduce movimiento
7
0 - X
- X -
0 X -
El tiempo de ejecucion es:8
Nodos: 38
Movimiento de la IA:
0 - X
0 X -
0 X -
Partida finalizada.
Jugador MinMax ha resultado ganador.
```

Algoritmo Poda Alfa-Beta (empate)

```
1. Minimax
2. Poda Alfa-Beta
Opción (1 o 2): 2
Jugador Humano: (X)
Jugador MinMax: (0)
- - -
- - -
- - -
Introduce movimiento
4
- - -
- X -
- - -
El tiempo de ejecucion es:5891
Nodos: 2316
Movimiento de la IA:
0 - -
- X -
- - -
Introduce movimiento
3
0 - -
X X -
- - -
El tiempo de ejecucion es:340
Nodos: 179
Movimiento de la IA:
0 - -
X X 0
- - -
```

```
Introduce movimiento
1
0 X -
X X 0
- - -
El tiempo de ejecucion es:45
Nodos: 35
Movimiento de la IA:
0 X -
X X 0
- 0 -
Introduce movimiento
2
0 X X
X X 0
- 0 -
El tiempo de ejecucion es:140
Nodos: 5
Movimiento de la IA:
0 X X
X X 0
0 0 -
Introduce movimiento
8
0 X X
X X 0
0 0 X
Partida finalizada.
La partida ha terminado en empate.
```

4. **Preguntas**

Estudiar la diferencia entre los algoritmos, tanto teóricamente como experimentalmente.

• ¿Qué algoritmo es más eficiente? ¿Por qué?.

Poda Alfa-Beta es más eficiente que MinMax, ya que se reduce significativamente el número de los nodos que explora, porque poda aquellos que no influyen en la decisión final. Alfa-beta tendría que examinar sólo O(b ^d/2) nodos para escoger el mejor movimiento, en vez de O (b^d) para minimax.

Además, experimentalmente, como podemos ver en el primer movimiento realizado para ambos algoritmos es el 4, y para el algoritmo MinMax se han visto un total de 55505, mientras que para el poda alfa-beta 2316, por lo que podemos confirmar la afirmación que poda-alfa beta visita menos nodos.

• (Opcional) Investiga sobre el algoritmo MinMax aplicado en el ajedrez. ¿Es una solución viable? ¿Y el algoritmo poda Alfa-Beta?.

Un grave problema del algoritmo Minimax es que, dependiendo del juego como lo es en el caso del ajedrez, no es factible realizar el árbol completo de jugadas posibles, ya que el número de estados a explorar puede llegar a ser exponencial respecto al número de estas jugadas, llegando a tener un coste temporal exponencial.

El algoritmo alfa-beta llega a más profundidad que el algoritmo minimax, reduce mucho el número de nodos evaluados por lo que podría ser una solución viable.

• (Opcional) ¿Que otros algoritmos de búsquedas por adversario existen? Compara sus ventajas y desventajas con los estudiados en el curso

La búsqueda Negamax es una variante de la búsqueda minimax que se basa en la propiedad de suma cero de un juego de dos jugadores. Este algoritmo se basa en el hecho de que simplifica la implementación del algoritmo minimax. Se basa en una propiedad matemática max(a,b)=-min(-a,-b), pero acaba teniendo la misma eficiencia que minimax.

5. **Conclusión**

Si comparamos ambos algoritmos, Minimax y Poda Alfa-Beta, en el caso de el tres en raya, ambos garantizan la misma calidad de decisión, ya que ambos algoritmos son lo suficientemente óptimos. Sin embargo poda alfa-beta evita explorar ramas del árbol innecesarias para la decisión final, disminuyendo el número de nodos visitados.