Estructura de Datos y Algoritmos

2014	Trabajo Práctico Especial – Algoritmo minimax

Integrantes:

- ✓ **Nardini, Gonzalo.**
- ✓ **Pérez Biancucci, Christian.**

Indice

Introducción

El Trabajo Práctico Especial consiste en la implementación del algoritmo minimax para el juego “Azulejos”. En el presente informe, se detalla la aplicación de dicho algoritmo, los problemas encontrados durante el desarrollo y las decisiones de diseño que han sido tomadas.

Asimismo, se incluye la descripción de los algoritmos utilizados en el desarrollo, indicando su funcionamiento y sus características principales.

También se incluyen tablas de comparación de tiempo, que especifican el tablero seleccionado, las opciones de configuración del algoritmo y el tiempo

Algoritmo minimax

Descripción

Minimax es un algoritmo recursivo de decisión para *mini*mizar la pérdida *máx*ima esperada en juegos con adversario y con información exacta.

El funcionamiento de este algoritmo puede resumirse como elegir el mejor movimiento para ti mismo, suponiendo que tu contrincante escogerá el peor para ti.

Alternativamente, puede ser pensado para *maximizar* la ganancia *mínima*. En este caso, el algoritmo es conocido con el nombre **Maximin**.

Originalmente, está formulado para dos jugadores, dónde los mismos tienen turnos alternadamente, y cada uno realiza un movimiento en su correspondiente turno. Ambos jugadores conocen el entorno, sus acciones, las acciones del oponente y los efectos de las acciones.

Estrategia

La estrategia del algoritmo minimax consiste en:

1. Generar el árbol de juego. Cada nodo del árbol es un estado del juego tras haber realizado un posible movimiento, teniendo como hijos a los posibles estados, dependiendo de los movimientos subsecuentes al movimiento realizado.
2. Calcular los valores correspondientes a cada nodo.
3. Calcular el valor de los nodos superiores a partir del valor de los inferiores. Según el nivel, si es MAX o MIN, se elegirán los valores mínimos y máximos representando los movimientos del jugador y del oponente.
4. Se elige la jugada de acuerdo a los valores obtenidos en el nivel superior.

El algoritmo explorará los nodos del árbol asignándoles un valor numérico mediante una función de evaluación, comenzando por los nodos terminales y subiendo hacia la raíz.

¿Cómo elegir la mejor movida?

Se calculan los valores de cada movimiento desde los nodos terminales hacia la raíz. Según el nivel del árbol, si es MAX o MIN, se eligiran los valores mínimos y máximo. Esto se realiza en todas las ramas originadas por cada uno de los movimientos posibles.

Dependiendo de la profundidad determinada al ejecutar el algoritmo, será la cantidad de movimientos "a futuro" que se evaluarán desde el movimiento actual.

Cada nodo hijo del movimiento actual, tendrá un valor numérico dependiendo de sus respectivos hijos. De acuerdo al turno: jugador u oponente, se aplicará el criterio de selección para el mínimo o máximo valor.

Características

Complejidad temporal:

- Tamaño del árbol: $O(b^m)$
- b: posibles movimientos en cada paso.
- m: movimientos hasta el fin del juego.

Complejidad espacial:

- Generar todos los posibles movimientos: $O(b\ x\ m)$
- Generar un camino por vez: $O(m)$

Para el Juego Azulejos:

La variable depth, en la configuración inicial, determina la cantidad de movimientos a futuro que se evaluarán.

Esto implica que la complejidad temporal y la complejidad espacial se ven determinada por dicha variable de configuración.

Complejidad temporal:

- Tamaño del árbol: $O(b^{depth})$

Complejidad espacial:

- Generar todos los posibles movimientos: $O(b\ x\ depth)$

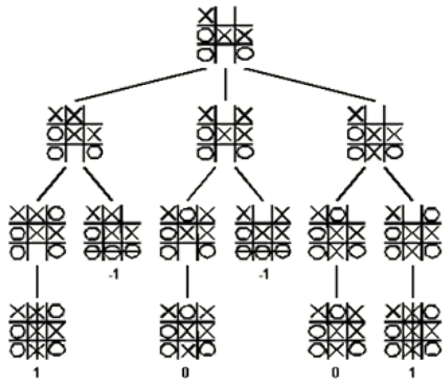
Problemas que se pueden resolver con el algoritmo minimax

Tateti

El tateti consiste en un tablero de 3x3, donde los dos jugadores, alternadamente, marcan un espacio con el objetivo de conseguir tener una línea de tres con su símbolo. La línea puede ser horizontal, vertical, o diagonal.

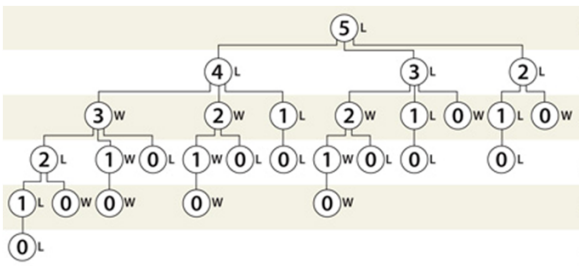
En este juego, el algoritmo de minimax es utilizado para calcular todos los posibles movimientos de los jugadores. Debido a que el tamaño del tablero y los posibles movimientos son pocos, se puede generar el árbol con todos los posibles movimientos desde el inicio.

En este caso, el algoritmo le aplica un valor entre 1, 0, -1 a cada nodo dependiendo si gana, empata, o pierde.



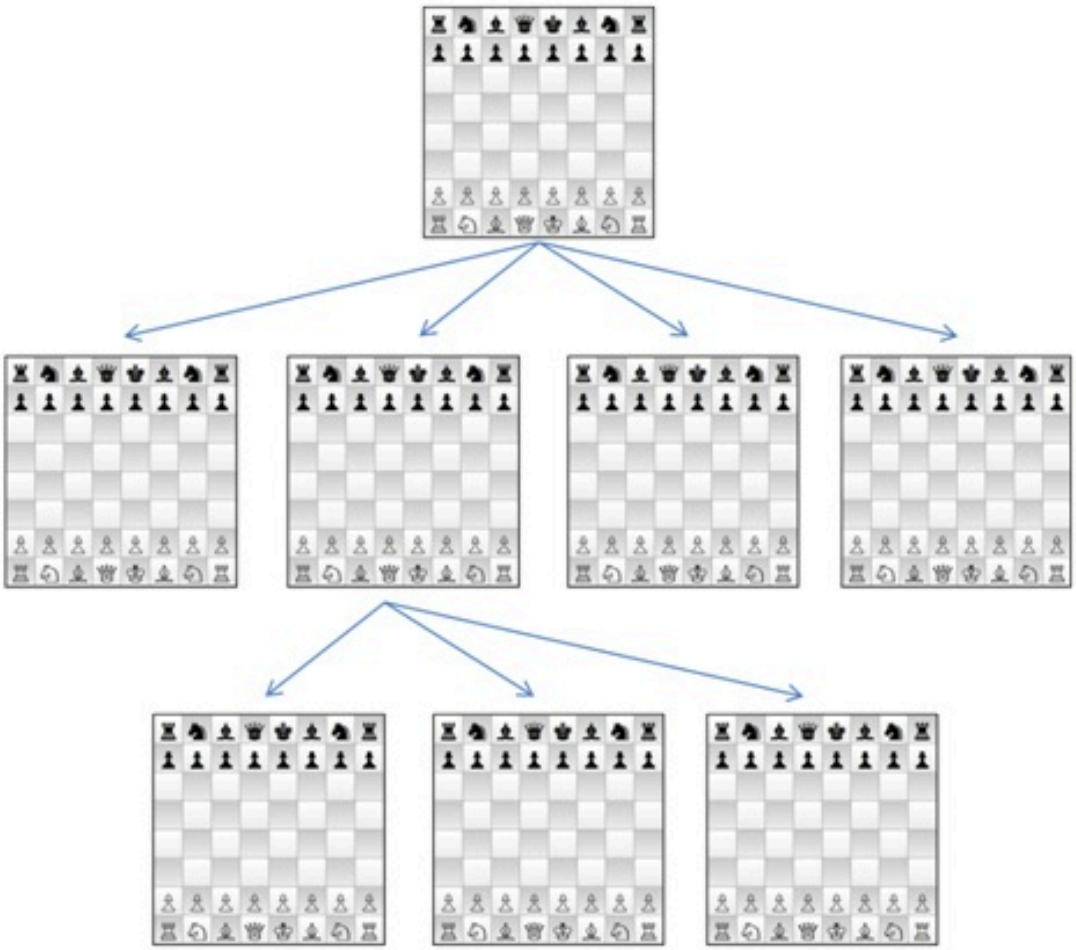
Nim

Este juego consiste en retirar una, dos o tres fichas de una pila de N fichas. Alternadamente, los jugadores, retiran las fichas y quién retire la última ficha, pierde. El algoritmo minimax determina en cada uno de sus nodos los posibles movimientos (retirar una, dos o tres fichas). En este caso, los valores que se aplican a los nodos están determinados por el resultado final: gana o pierde.



Ajedrez

El ajedrez, al igual que los juegos mencionados anteriormente, puede ser resuelto a través del algoritmo minimax. Se generará un árbol, cuyos nodos serán los estados del tablero determinado por los movimientos realizados, y cada nodo hijo será producto de los movimientos subsecuentes. La diferencia con los juegos anteriormente planteados, es al diversidad de movimientos posibles. No solo la posición de la ficha, sino los diferentes movimientos que realiza cada una de las piezas que conforman el ajedrez. En conclusión, si bien la aplicación del algoritmo es la misma, éste se verá afectado temporal y espacialmente debido a la cantidad de nodos que deberá generar en el árbol. Vale destacar que muchos nodos serán compartidos por la distintas ramas, debido a que se conseguirá llegar a un mismo estado del juego a través de movimientos diferentes.



Podas

Estrategia

El objetivo principal es eliminar aquellos nodos/ramas que no nos lleven a buenas soluciones. Para ello se aplica una heurística dónde se evalúan los nodos correspondientes a un nivel, y se determina entre ellos cúales son innecesarios para podarlos y no examinarlos.

La poda permite reducir los tiempos de exploración y evitar generar estados demás.

También se utiliza para detectar simetrías en el espacio de búsqueda. En los ejemplos planteados anteriormente, se puede aplicar la poda al algoritmo minimax para reducir su exploración y generación de nodos.

En el tateti, por ejemplo, al ubicar una ficha en el centro y una ficha en alguna esquina, se encuentra una simetría, ya que la ficha en la esquina podría ser cualquiera de las cuatro (basta con rotar el tablero).

En el juego de nim, la simetría se encuentra al llegar a una cantidad X de fichas restantes en la pila. Esta cantidad puede ser obtenida habiendo sacado diferentes números de fichas y aún así obtener la misma cantidad de fichas restantes.

Por ejemplo, el jugador 1 saca una ficha y el jugador 2 saca tres fichas. El jugador 1 saca dos fichas y el jugador 2 saca dos fichas. De cualquier forma, al finalizar ambos turnos, se han retirado cuatro fichas de la pila. Esto implica que hay una simetría al finalizar ambos movimientos.

Poda alfa-beta

El valor de un nodo es relevante sólo si existe la posibilidad de llegar a él. Si un jugador racional nunca dejaría que lleguemos a ese nodo, entonces no hace falta examinarlo.

Alfa es el valor de la mejor opción hasta el momento a lo largo del camino para MAX Esto implicará por lo anto la elección del valor más alto.

Beta es el valor de la mejor opción hasta el momento a lo largo del camino para MIN. Esto implicará por lo tanto la elección del valor más bajo.

Estructuras

Se utilizó la estructura de árbol para representar el estado de juego, y los estados subsecuentes dependiendo de los movimientos realizados.

Su implementación se encuentra en la clase *BoardState*. La clase posee el estado del tablero, el puntaje y el movimiento. A su vez posee una lista encadenada de BoardState, la cuál representa los posibles movimientos a partir del estado actual.

El algoritmo *minimax* recorre dicha estructura de árbol evaluando cada uno de los estados y sus posibles movimientos subsecuentes. Al ser una estructura recursiva, el algoritmo minimax llama recursivamente a los posibles estados de cada BoardState indicando depth[1] – 1, a través del método *runDeepAlgorithm*.

[1] Depth: Variable de configuración al ejecutar el programa. Determina el nivel de profundidad con el cual corre el algoritmo.

Problemas y resoluciones

Una de las primeras problemáticas de la implementación de dicho algoritmo se ve reflejada en el tamaño del tablero y la cantidad de colores utilizados. Si el tamaño del tablero es muy grande, y se utilizan muchos colores, el árbol generado al calcular los posibles de movimientos, aumenta notoriamente la complejidad espacial.

Uno de los problemas encontrados a lo largo del desarrollo fue la profundidad del algoritmo para calcular el movimiento óptimo. Si establecíamos una profundidad mayora a la cantidad de movimientos posibles, se encontraba con un análisis exhaustivo de movimientos que retornaban puntaje 0, modificando la elección del mejor movimiento debido a un mal cálculo. Por este motivo, agregamos una validación, para chequear que si se han agotado los movimientos posibles, regrese a su nodo padre sucesivamente hasta llegar a la raíz, y tener el movimiento siguiente.

Al desarrollo del algoritmo minimax le agregamos una heurística para elegir siempre el mejor movimiento de la estructura, por más que encontrase diferentes ramas ganadoras. En caso de tener dos ramas con igual calificación, se queda con la primera explorada.

Conclusiones

Consideramos

Referencias

Consideramos

Anexo

Consideramos