

Conecta4Boom : Memoria

Álvaro Vega Romero
Inteligencia Artificial

1. Análisis del problema

El problema que nos encontramos, es un juego basado en el conocido como “4 en raya”, las variaciones que tenemos en este son:

- Los turnos tienen dos movimientos (menos el primero).
- Cada cuatro turnos aparece una bomba cuyas propiedades aparece en la guía de la práctica.

Con estos conceptos claros, el objetivo de la práctica se basa en buscar una heurística (función valoración) para valorar los nodos terminales (terminales ya que no lo desarrollaremos totalmente el árbol de juego) y desarrollar con las estrategias minimax o poda alfa-beta el árbol de juego. En nuestro caso, haremos ambas, pero usaremos solo la de poda.

Una vez hecho lo anterior, debemos desarrollar la función think() para ver el movimiento que debe hacer nuestro jugador (con la estrategia minimax) en el estado actual que le garantice el mayor beneficio posible.

Como la valoración que desarrollemos es muy probable que no sea muy buena, lo más complicado y lo que tendremos que desarrollar en el paso 2. es el cómo plantearla, hacerla y cómo la mejoramos.

Lo siguiente a hacer es entender el código que se nos plantea, sobre todo los métodos de la clase `Enviroment` que se plantean en la guía de la práctica ya que necesitaremos usarlos para implementar las funciones anteriormente mencionadas y luego ver el código de `jugador.cpp` dado ya que puede que nos sirva para aclararnos en la implementación del código.

Y habiendo hecho esto correctamente, ya podemos pensar en la implementación en código de lo planteado anteriormente.

2. Descripción de la solución planteada

Nos pusimos manos a la obra y lo primero que hicimos fue implementar la función `think()` (que es simplemente comentar una parte del código y descomentar otra, además de añadirle una llamada a MiniMax y elegir entre poda alfa-beta y MiniMax comentando una opción y descomentando la otra).

Lo siguiente que hacemos, son las funciones correspondientes al algoritmo minimax y a poda alfa-beta, cuya explicación la especifico en el fichero `jugador.cpp` con comentarios

Luego, lo que hicimos fue hacer la función `valoración`. Lo primero que hicimos en esta, fue plantear el caso de que en el estado que se pasa como parámetro de la función, el juego se ha terminado y, por tanto, tenemos un ganador. Esto lo podemos comprobar con la función `"int RevisarTablero() const;"` donde según el resultado que nos devuelva, devolveremos directamente el mayor valor (en caso de victoria del jugador que llama a `valoración`), el menor (en caso de derrota) o 0 si es un empate.

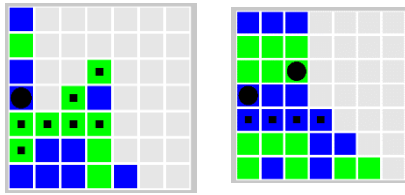
Luego, en caso de que no hayamos llegado a un estado que cumpla las condiciones previas, es donde tenemos que analizar el estado. Para esto, lo que hacemos es procesar todas las casillas del tablero y en caso de que estas casillas sean del jugador que llama la función `"Valoración(estado, jugador, fila, columna)"`, se suma el valor que se obtenga al valorar esa casilla (pasándole como parámetro "jugador" el jugador que llama a `"Valoración(estado, jugador)"`) y en caso de que sean del jugador rival, se restan (se le pasa como parámetro. La suma global de todas las valoraciones de las casillas, será la valoración del estado.

Como hemos dicho, tenemos que valorar las casillas, y para esto, crearemos una función que valore las casillas horizontal, vertical y diagonalmente. (Sobre esto, cabe decir que en la heurística final, no valoramos diagonalmente).

Como primera implementación, hemos tenido en cuenta solo horizontal y verticalmente. Lo que hacemos es:

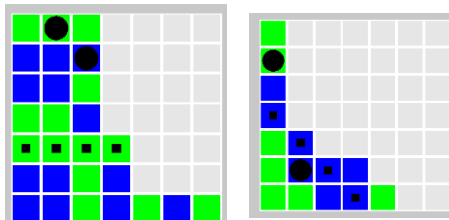
- Contar el número de casillas contiguas a la casilla que pasamos como parámetro.
- En función del número de casillas que haya juntas, asignamos una valoración numérica. Cuando tengamos tres casillas juntas, nuestra valoración será mejor que cuando tengamos dos juntas, y esta, a su vez será mejor que cuando tengamos una casilla únicamente.
- Además, en el caso de la valoración horizontal, en caso de ser una casilla, le daremos mayor valoración cuanto más centrado esté la casilla, ya que es más probable ganar si tenemos el centro.

Con lo mencionado anteriormente, conseguimos ya vencer al ninja 1. (Heurística es el jugador verde en la primera imagen, y el azul en la segunda).



Luego, probamos con el ninja 2, y vemos como este nos gana debido a que no tenemos en cuenta las diagonales. Por tanto, nuestro futuro cambio será tener en cuenta las diagonales en las valoraciones. Es importante destacar que cada casilla tiene dos diagonales, una hacia la “izq” (es decir, la casilla más a la derecha de la diagonal está lo más abajo) y otra hacia la “dcha” (lo contrario).

Así, contra el ninja 1 se quedan así los resultados: (Heurística es el jugador verde en la primera imagen, y el azul en la segunda)



Y contra el ninja 2 seguimos sin conseguir la victoria.

Ya que no ganábamos al Ninja2 de ninguna de las maneras, lo que hicimos fue modificar las valoraciones, pero en vano, ya que no sirvieron para ganarle.

Tras ver que no es problema de las valoraciones, vemos que lo mejor es cambiar de estrategia.

Lo que haremos ahora, será tener en cuenta **únicamente** los estados que pueden llegar a hacer un 4 en raya:

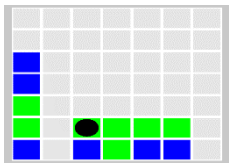
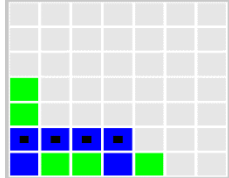
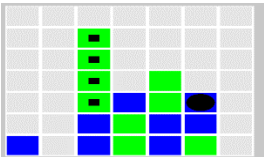
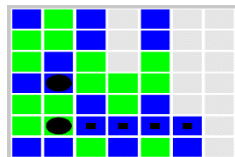
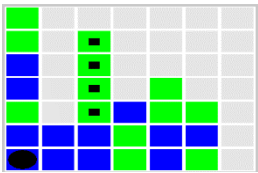
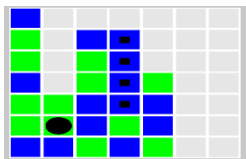
- Verticalmente, cuando calculamos el número de fichas contiguas que tenemos, tendremos en cuenta si la casilla superior a la última nuestra es un hueco. En caso de que sea un hueco, sabiendo el número de casillas contiguas que calculamos previamente y en que fila está el hueco (si hay) podremos saber si es posible llegar a un 4 en raya verticalmente. Merece la pena destacar que si el 4 en raya se saldría de los márgenes del tablero no lo contamos.
- Horizontalmente, la filosofía es similar a la anterior, pero hay una diferencia. Si antes había un hueco encima de una casilla, las casillas superiores a este hueco, también

eran huecos. Aquí, tendremos que analizar las casillas que están a los lados de los huecos.

Así, tendremos en cuenta si las casillas a la izquierda y derecha de “nuestro conjunto de fichas” son huecos y luego, en función del número de casillas contiguas más cosas:

- Si tenemos 3 casillas contiguas, con que alguno de los huecos de los lados exista, ya lo valoramos (100).
- Si tenemos 2 casillas contiguas, si los dos huecos existen, le damos valoración de dos (25 en nuestro caso). Si existe solo uno de ambos, veremos la casilla siguiente al hueco y dependiendo si es otro hueco o una casilla del jugador que estamos procesando o del rival le damos valoración de 2 (25), 3(100) o nada respectivamente.
- Si tenemos una única casilla contigua, vemos el número de huecos y casillas de nuestro jugador que hay a ambos lados antes de encontrarnos con una casilla del rival o salirnos del mapa. Si la suma de huecos y casillas nuestras es ≥ 3 , podremos llegar al 4 en raya. La valoración aquí depende la posición de la casilla, si está centrada le damos (18), a los lados del centro (16), a los lados de los bordes (14) y en los bordes (12), haciendo que busque el centro.

Volvemos a hacer las funciones de valoración de las casillas horizontal y verticalmente (diagonalmente no, ya que no hace falta) y vemos como ya si ganamos a todos los ninjas:

Ninja / Jugador Heurística	Jugador Verde	Jugador Azul
Ninja 1		
Ninja 2		
Ninja 3		

Y así, fue como conseguimos hacer la práctica, ganando a todos los rivales que se nos plantearon.