Poda alfa-beta con heurística

David Cabezas Berrido

2 de junio de 2019

Índice

1. Objetivo

El objetivo es programar una inteligencia artificial que juegue a Desconecta 4 Boom, y que use el algoritmo de poda alfa-beta con una heurística.

En la memoria explicaré como he implementado el algoritmo y la heurística, así como alguna modificación que he realizado.

2. Algoritmo de Poda alfa-beta

Es una variante del Minimax un algoritmo muy eficaz para explorar los caminos que puede seguir una partida en un juego de dos jugadores como ajedrez, damas o Desconecta 4 Boom. Se basas en explorar un árbol de posibilidades en el que en los niveles pares juega un jugador y es los impares otro, se supone que cada jugador realiza el mejor movimiento posible. Por tanto, si tenemos una valoración de un estado del juego que indica la ventaja de uno de los dos jugadores, ese jugador hará el movimiento que maximice esa valoración y su oponente el movimiento que la minimice.

El algoritmo de Poda alfa-beta incluye una modificación para evitar explorar ramas del árbol que presentan una desventaja evitable para uno de los jugadores, luego el juego jamás se desarrollará por ese camino. Para ello se almacenan:

- El la variable alpha, la puntuación mínima que tiene asegurada el jugador que maximiza. Se inicializa a $-\infty$.
- En la variable beta, la puntuación máxima que tiene asegurada el jugador que minimiza. Se inicializa a $+\infty$.

Por tanto, en el momento que alpha supere a beta se estará explorando un camino en el que un jugador tiene peor puntuación (para él) de la que tenía asegurada y evitará que el juego siga esa rama.

Nos es imposible explorar el árbol infinitamente, así que exploramos hasta profundidad 8 (4 jugadas de cada jugador por delante de al situación actual) o hasta que acabe el juego. Entonces devolvemos la valoración heurística del tablero. Un jugador intentará llegar al nodo con mayor valoración heurística y el otro al nodo con menor valoración heurística.

He implementado este algoritmo en la función `Poda_AlphaBeta`, que devuelve al jugador que la llama la puntuación óptima que puede conseguir (asegurada). El metodo `Think` toma la lista de tableros a los que se pueden llegar en la siguiente acción desde el tablero actual y evalúa la función `Poda_AlphaBeta` para cada uno de esos tableros, eligiendo la que ofrezca mejor puntuación.

En la implementación que he hecho, siempre considero que el jugador al que le toca es el que maximiza, por lo que cuando se llama a la `Valoracion`, siempre se llama con el jugador que llamó a la función en primera instancia (en el método `think`).

3. Función Valoración

La función `Valoracion` asigna un valor heurístico a un tablero para un jugador, y devuelve un valor que es mayor cuanto mayor sea la ventaja de ese jugador en ese tablero. Lo que tiene en cuenta es:

- **Si la partida ha terminado:** devuelve $-\infty$ si el jugador que solicita la valoración ha perdido, $+\infty$ si ha ganado y 0 si hay empate.
- **El número de 3 en raya:** cuenta el número de 3 en raya que tiene cada jugador. Usa la función `NenLinea3`, que se auxilia de `EnLinea3` una modificación del método `Enlinea` de la clase `Environment`, que comprueba si hay un 4 en raya. A la diferencia del número de 3 en raya le asigna un peso `PESO3L`, esto resultará en un número positivo si el enemigo tiene más 3 en raya que nosotros y en un número negativo en caso contrario.

- **El número de 2 en raya:** cuenta el número de 2 en raya que tiene cada jugador usando las funciones `NenLinea2` y `EnLinea2` de forma análoga al número de 3 en raya. Y le asigna un peso `PESO2L` a la diferencia. Hay que tener presente que un 3 en raya va a contar como dos 2 en raya.
- **El número de fichas** de cada jugador. A la diferencia le asigna un peso `PESOFICHA`.
- **La buena colocación de la bomba:** este valor ya es mayor cuanto mejor colocada esté la bomba del jugador que pide la valoración respecto a la del enemigo y le da un peso de `PESOBOMBA`. Si no hay bomba devuelve 0, en otro caso:
  - La puntuación es mayor cuanto más baja esté la bomba, le da una importancia `PESOBAJO`.
  - Es mayor cuanto más fichas del propio jugador van a ser destruidas por la bomba, multiplica por `PESOELIMINAR`.
  - Es mayor cuando más fichas tenga el adversario en la fila superior a la bomba, habrá mayor probabilidad de que sus fichas caigan cuando la bomba explote, multiplica por `PESODESPLAZAR`.

Ya sólo queda jugar con los pesos (en `jugador.h`) e ir analizando el comportamiento del jugador.

4. Problema al equilibrar el peso de la bomba

Le he dado mucha importancia a colocar la bomba bien, ya que en la práctica daba buenos resultados. El problema es que la puntuación que ofrecía a un tablero el tener la bomba bien colocada, era mayor que la puntuación que recibía por la ventaja que producía explotarla. Esto resultaba en que nunca explotaba la bomba.

En cambio, el hecho de dar poco peso a la colocación de la bomba producía en que se la tratara como una ficha normal, y se evitase colocar alineada con el resto de fichas. Esto provoca que al explotar la bomba no se gane mucha ventaja.

Esto podría solucionarse haciendo más pruebas hasta encontrar el equilibrio en los pesos deseado, pero hacer una prueba consume mucho tiempo. También podría solucionarse si tuviésemos un horizonte de búsqueda mayor, para que contase con la bomba próxima o visualizase la victoria.

Como ninguna de estas soluciones era factible, ya fuera por mi tiempo limitado o por las limitaciones de la práctica, he añadido una excepción en la regla del algoritmo de Poda alfa-beta.

Si después de calcular las valoraciones (utilizando la Poda alfa-beta) de las distintas acciones se dan una serie de características, se elige la acción BOOM, ya que la puntuación que recibe por la función valoración no hace justicia a lo adecuada que es. Las condiciones son:

- La acción BOOM está disponible (tengo una bomba colocada).
- El número de jugada es congruente con 4 módulo 5 (si no detono la bomba ahora, perderé la siguiente).
- La acción mejor valorada tiene una valoración por debajo de cierto umbral (puede haber una alternativa muy buena).
- La acción BOOM tiene una valoración por encima de cierto umbral (la acción BOOM no me lleva a una derrota casi segura).

Esto rompe con el espíritu del algoritmo de Poda alfa-beta, pero es una solución muy simple y rápida al problema y da muy buenos resultados. Con este algoritmo he conseguido vencer a los tres ninjas tanto como primer jugador, como con el segundo.