MEMORIA-CUADERNO DE BITÁCORA

CONECTA 4
BOOM

Practica 4 Inteligencia Artificial

Juan Carlos Ruiz Fernández 2ºB3

Análisis del problema

Principalmente me enfrentaba a dos problemas gruesos:

- La implementación del algoritmo de juego (minimax/podaAlfaBeta)
- Valoración Heurística

Además de implementar acciones deliberativas, como tener en cuenta la bomba o ciertas partidas clave.

Algo evidente es el desconocimiento de como funcionaba todo el programa Conecta4Boom. Que se solvent

Descripción y evolución de las soluciones

Al principio de esta práctica pensé en generar el árbol de juego antes de llamar al algoritmo, para que lo recorriera directamente. Cree la clase *Nodo* para crear este árbol, el cual llevaría un entorno raíz, y un array de nodos para los hijos. Añadiendo la cantidad de hijos que hay y una valoración de ese nodo.

A la vez cree el algoritmo MiniMax y la heurística de valoración primitiva que no evaluaba los nodos no-terminales.

Mi primer problema, de muchos, fue que MiniMax devolvía un *double* pero no sabía cuál sería la acción ultima realizada. Para ello recorrí de nuevo los hijos después de la llamada al algoritmo y busqué el que tenía esa misma valoración, calculando después su LastAction. Ya que en MiniMax modificaba esos valores. Esto puede verse en mi código comentado, la primera funcion MiniMax Antigua.

Mi primer error fue continuar sin haber probado el algoritmo con ValoracionTest al menos.

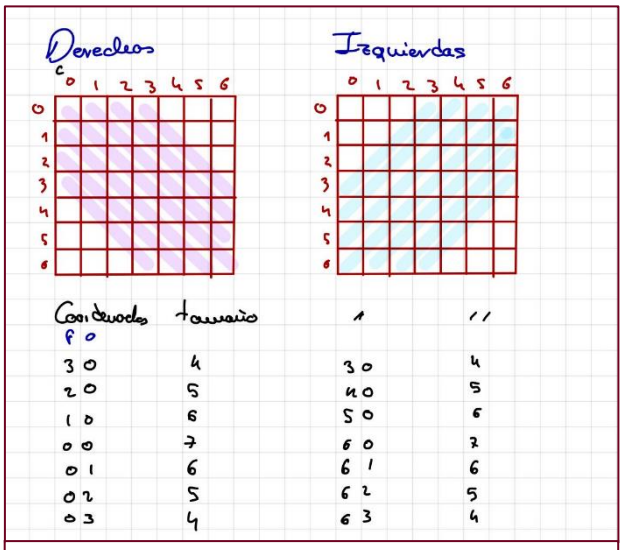
Continúe ideando la heurística. La cual la subdividí en problemas tales que ver las posibilidades verticales, horizontales y diagonales. De nuevo, vi un factor común y es que todas se calculaban sobre una línea.

Pues mi siguiente paso fue crear la heurística base llamada *evalualinea* que evalúa una ristra de casillas con tamaño predefinido en función de un jugador. Esta devuelve un valor positivo cuando las posibilidades son favorables para el jugador pasado por argumento y uno negativo para el contrincante, que lo calculo. Estas posibilidades de ganar aumentan cuando tengo varias fichas consecutivas, jugador o contrincante. Además, si la siguiente ficha es del jugador a maximizar y la valoración previa es negativa (el contrincante tenía varias fichas consecutivas) la evaluación cambiará de signo y aumentará más. Esto se traduce a que valora muy positivamente que se rompa la secuencia del otro jugador. Respectivamente pasa con el contrincante.

La función que evalúa las columnas (posibilidadesVerticales), es la más simple. Porque solo tiene que evaluar la ristra de casillas en un sentido. Pues esta la guarda en una variable y con ella ejecuto *evalualinea*. Así con todas las columnas.

La función que evalúa las filas (posibilidadesHorizontales) si que debe tener en cuenta el sentido en el que se recorre. Pues se puede ganar de izquierda a derecha y viceversa. Pues la única diferencia es que capta la misma ristra en ambos sentidos a la vez y pasa cada una a *evalualínea.*

La función que evalúa las diagonales es algo más compleja ya que hay dos tipos de diagonales, las que ascienden a derechas y a izquierdas. A su vez cada diagonal debe ser evaluada en ambos sentidos, igual que las filas. Pues previamente calculé el inicio de las diagonales posibles tanto a derechas y a izquierdas y lo almacené en unas variables globales, junto a su tamaño. Como se ve en mi croquis de la derecha:



Croquis diagonales calculadas

El programa, como siempre, a la primera no funcinó. Me daba error de segmento y *free(): invalid next size (fast)* porque creaba las variables de filas y columnas en las evaluaciones verticales y horizontales con el heap y a veces las que tenían tamaño 0 las eliminaba sin tener nada. Al final no usé heap y creé las variables dentro del bucle.

Aun así, me daba error de segmentación. Bajando el numero de profundidad no pasaba nada, al menos el juego funcionaba durante una o dos jugadas. El problema estaba en el nodo almacenaba los hijos, creados en el heap, y nunca los eliminaba.

Para ello cambie de estrategia, y en vez de almacenar los hijos los generaba en cada llamada al algoritmo antes de evaluar, así se irían eliminando con forme termina la recurrencia. Esto se puede ver en mi segundo minimax comentado. Este utilizaba un struct *ValorLastAction* que contiene un atributo de valor y otro de ultima acción.

A partir de aquí el juego funcionaba, pero no daba unos resultados esperados. Las acciones eran extrañas y aparentemente nada premeditadas. En algunos casos ni si quiera juagaba, obtenía valores indefinidos para valor y para lastAction.

Después de varios días probando, avancé sin arreglar ese error e implementé el algoritmo podaAlfaBeta con el struct anterior, es el último algoritmo que tengo comentado antes de llegar al definitivo. Funcionó y no volvía a darme valores extraños. Pero tampoco funcionaba bien.

Ahora el juego funcionaba hasta casi el final sin errores. Pero no ganaba. Estuve varios días dándole vueltas porque aparentemente no evaluaba bien las fichas consecutivas del contrincante, pues esta tenía dos juntas y en la siguiente partida me haría perder. Esto no lo consideraba y ponía las fichas en cualquier otro lado.

Por si no manejaba bien el struct, rehíce de nuevo el algoritmo. Adaptándolo con pair<double, ActionType>. Como el error no residía ahí, seguía sin funcionar. Pero esta implementación es la que he mantenido.

Aquí pasé varios días repasando las valoraciones, cambiando las puntuaciones y la forma de recorrer las líneas y no conseguía nada. Intenté pedir tutoría y preguntarlo en clase, pero no pude ni tuve tiempo.

Tras revisar de nuevo el algoritmo razoné que en este juego se hace cada dos turnos. Pues yo en cada llamada recursiva cambiaba el jugador actual a evaluar. Este error lo arrastré desde el principio de la idea de minimax, que el algoritmo general funcionaba así.

La solución fue que para saber el jugador a maximizar solo bastaba con llamar a JugadorActivo y en las llamadas recursivas mantener ese jugador constante sin alterarlo.

Después de este arreglo no iba mucho mejor. Siempre me devolvía el algoritmo una valoración final de 0 y un PUT1. Cosa extraña. Revisé las valoraciones verticales a mas no poder y bloqueando las valoraciones que se hacían en una columna llena y que la anterior acción fuese poner la ficha en esa misma columna.

Pero el error no residía ahí. Pues yo me creaba una variable pair auxiliar que representaba el mejor valor hasta el momento obtenido en los bucles. Revisando videos del funcionamiento del alfa beta me percaté que no tenía en cuenta que cuando se evaluaban nodos peores no actualizaba esa variable mejorValor. Pues ésta por defecto inicializaba sus atributos a 0. Ahí me di cuenta que debía inicializarlo a los valores previos a la llamada de la función, que para no cambiar de nuevo la implementación seteé el valor en función del jugador a evaluar y calculaba la última acción de nodo actual antes de evaluar los hijos. De esa manera si solo encontraba hijos peores mantendría las valoraciones pasadas y no las cambiaría a 0.

Ahora el juego va sin problemas ni errores y vence al Ninja 1, un poco torpe pero lo gana.

Yo no consigo ganar a mi heurística, seré bastante torpe.

Ahora me faltaba mejorar la heurística, pero no tengo mas tiempo. Solo he podido hacer una acción deliberativa y es que cuando sea la primera partida, empiece por la acción PUT4 ya que las casillas centrales son cruciales.

Me hubiese gustado implementar una valoración para las casillas bomba en función del numero de fichas que eliminaría del contrincante.