Informe TPE

Estructuras de Datos y Algoritmos

Alumno: Pedro Vedoya (57605)

Fecha: 19/06/2018

Justificación de la estructura de datos:

Tablero: para el tablero, se descompuso el juego en distintas “piezas”, imitando lo que sería el tablero en la vida real. Se comenzó por lo mas pequeño, una línea, que tiene las propiedades de las coordenadas donde se encuentra (lo que le permite además ser comparada y distinguida con las demás), y si fue pintada no. Luego, se decidió representar cada cuadrado formable como una clase independiente, que parecía ser la manera más cómoda para poder almacenar las líneas que la conforman y el jugador al que le pertenece (jugador 1, 2 o ninguno). Finalmente, el tablero se represento como una matriz de dichos cuadrados, que a la hora de comprender como esta formado resultó mas simple. Sin embargo, más adelante esta estructura resultó problemática para recorrerlo y agregar líneas, dado que había líneas repetidas entre dos cuadrados, lo que se solucionó usando un mapa que por cada línea se tenía un set con los tableros donde pertenecía, que se generaba cada vez que se creaba el tablero, recorriendo todos los cuadrados y armando el mapa, y que resulto realmente practico a la hora de agregar una línea, ya que se agrega en en los dos tableros al mismo tiempo, buscando la línea que se quiere poner, y ahorra más tiempo. El tablero, además, guarda el color de cada jugador y sus respectivos puntajes. Esta clase posee métodos para imprimir el tablero, acceder al tablero y su puntaje, y manipular los cuadrados que forman el tablero para agregar las líneas, lo que resulta especialmente útil para armar el algoritmo minimax. Esta clase resultó ser el principal problema, dado que mucho tiempo fue invertido en poder hacerla funcionar correctamente. Al principio presentó muchos problemas de acceso al tablero, especialmente porque guardaba bien las líneas en el mapa, y resultaba un gran problema a la hora de actualizarlo. Esto se soluciono haciendo siempre clonaciones del mapa, que directamente reemplazaban las viejas. Sin embargo, mucho tiempo también fue dedicado a lograr que el algoritmo minimax no tuviese problemas para crear las posibilidades de líneas, tirando siempre un “NullPointerException”, y finalmente la impresión del tablero en pantalla, dado qie acceder a cada línea en particular mediante sus coordenadas fue particularmente difícil. Una posible manera de hacer óptimo el acceso al tablero podría ser cambiando el objeto línea por un booleano que diga si hay línea pintada o no, dado que guardar las coordenadas de cada línea no es muy eficiente.

Jugador: el jugador, representado por un número que es correspondiente a un turno, está compuesto por un stack donde se guarda el estado previo la última jugada que hizo, así cuando se toca el botón undo, se popea y reemplaza en la jugada actual, y

esto está representado por pushear y una copia del estado del tablero con toda su información, y luego reemplazando a actual al popearlo. Esto pareció ser la mejor opción, dado que en cuanto espacio no consume mucho, y es muy rápido en ejecución. La clase Player posee también el puntaje al momento, y una variable booleana que indica si es humano, que es llamada cuando se corren ciertos métodos para saber si se debe llamar a la inteligencia artificial o pedirle información al jugador. La clase de Inteligencia Artificial, que hereda de esta, cambia esta variable asignándole siempre falso, y también, además de todo esto, y guardar un método que cuando se lo llama, corre el algoritmo minimax por tiempo o profundidad (dependiendo de los parámetros con los que fue creado, que son recibidos por entrada estándar), y retorna la mejor opción que tomar en el juego(por medio de una línea), que luego el controlador del juego pone en el tablero.

Controlador: La clase "GameManager", que controla como se ejecuta y desarrolla el juego, contiene el tablero, los jugadores y los puntajes, es el encargado de decidir a quien le toca jugar, crear los jugadores dependiendo de la entrada del usuario, y de llamar al algoritmo minimax. Es fundamental para conectar el FrontEnd con el BackEnd, dado que posee las llamadas a jugadas y el acceso al tablero para realizar los cambios. Esto presenta un problema a la hora de poder hacer correctamente los accesos entre el Backend y el FrontEnd. La clase manager es la encargada de recibir los parámetros del main y armar dos tipos de jugadores (AI y Humano) y decirle al AI que estrategia de búsqueda va a tener. También se encuentra el método que maneja el "undo", que reemplaza la instancia del tablero que se encuentra en la clase

Nodo: la clase Nodo es la columna vertebral del algoritmo minimax, cuando se la crea, posee una instancia del tablero, de los jugadores, y un set vacío. Su método mas importante es el que evalúa lo que pasa al agregar cada línea disponible al tablero, es decir, agrega al set el tablero que queda agregando cada arista, y los guarda en objetos del tipo Nodo dentro del set. Así, a medida que el algoritmo minimax va creando distintos niveles de nodos, se va formando un árbol con todas las posibles decisiones a tomar. Un set es la mejor manera de guardar las opciones, dado que el acceso es casi inmediato y no puede haber repetidos.

Heurística:

Para la heurística, se intentó replicar un pensamiento bastante similar al de un jugador humano, priorizando siempre intentar cerrar un cuadrado, sobre todo, y luego considera la cantidad de cuadrados que deja con 2 o 3 líneas a su alrededor. Primero considera la totalidad el puntaje total obtenido, haciendo una resta del puntaje del jugador actual sobre el del otro

(obtiene la mayor cantidad de puntos hechos) y lo multiplica por 10, para darle la mayor ponderación de todas, ya que esto es realmente importante. Luego, dependiendo si es min o max, le suma (si es max) o resta (si es min) la cantidad de cuadrados de nivel 2 y nivel 3 que hay, dado a que es más propenso a hacer más puntos de esa manera, especialmente si puede hacerlos después de hacer si próxima jugada. Se pondera multiplicando por 5 los cuadrados de 3, dado a que son más importantes que los de 2, porque le habilita su siguiente jugada.

Algoritmo Minimax:

Este algoritmo de búsqueda, utilizando el tablero actual, permuta sobre todas las posibles opciones inmediatas que tiene (utilizando la clase nodo, como ya fue explicada), y llama para todas las opciones a una clase wrapper, que es la encargada de ejecutar el algoritmo en sí y devuelve un int con que tan buena fue esta "opción", que se compara siempre con la mejor obtenida, guardando a su vez que línea fue la que se insertó (que será la que el algoritmo elija), en el caso que sea mejor. Esta simula, mediante el pasaje de un parámetro booleano que le indica si está siendo max o min, si debe ponderar positiva o negativamente. Primero se fija si se cumple la condición de finalización (tiempo o profundidad) pasado por parámetro, la profundidad se mide restando recursivamente el nivel inicial de cada rama hasta llegar a 0, donde devuelve la evaluación, el tiempo se mide pasando como parámetro inicial el tiempo máximo(compuesto por el tiempo que le da el usuario a la máquina, y sumándole la hora local), y constantemente a cada paso de la recursión se fija si el tiempo local es superior al máximo, de serlo retorna la evaluación, si ese es el caso, se ejecuta la evaluación del caso actual utilizando la heurístic. Si no es el caso base, se fija si el jugador es min o max , si es max (jugador que quiere ganar), primero llama recursivamente a la wrapper otra vez y lo guarda en una variable local, guardando el mejor valor de las opciones, y luego , en el caso de estar activada la poda, se fija si el valor de Alpha es mayor a Beta, siendo Alpha el máximo de sus local o Alpha, si ese es el caso, no requiere seguir haciendo evaluaciones. El procedimiento para min es el mismo, solo que simula la mejor opción para el otro jugador, es decir, cuál sería la mas perjudicial para "yo", por lo que decide el que resulta en menos puntos para max. Finalmente, cada nodo en el árbol de decisiones es encargado de devolver el mejor valor posible para max, quedándose así con la mejor jugada posible.

El algoritmo está hecho para poder representar el siguiente código, conseguido de StackOverflow:

```
function minimax(node, depth, maximizingPlayer)
```

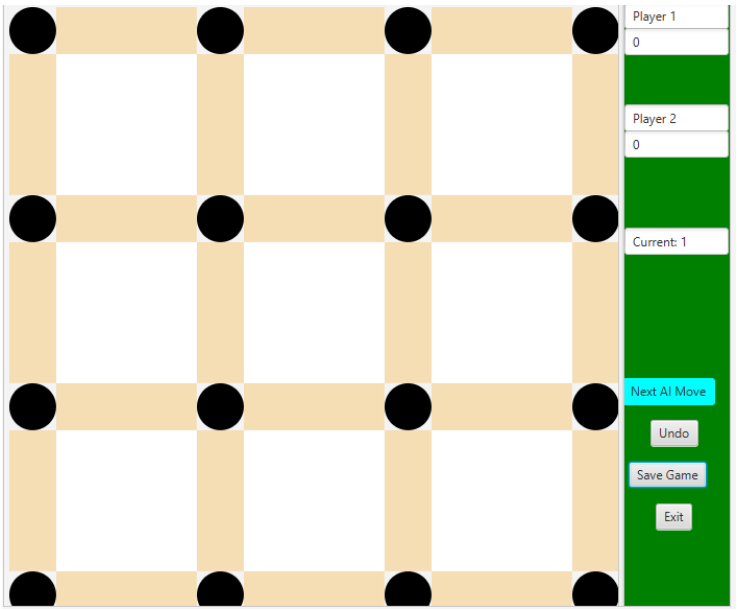
```
02     if depth = 0 or node is a terminal node
03         return the heuristic value of node

04     if maximizingPlayer
05         bestValue := −∞
06         for each child of node
07             v := minimax(child, depth − 1, FALSE)
08             bestValue := max(bestValue, v)
09         if(beta <= alpha(alpha being,max(alpha,best))) return max
10         return bestValue
11     else    (* minimizing player *)
12         bestValue := +∞
13         for each child of node
14             v := minimax(child, depth − 1, TRUE)
15             bestValue := min(bestValue, v)
16         if(beta <= alpha(alpha being max(alpha,best))) return min
17        return bestValue
```

Problemas encontrados:

El mayor problema encontrado fue el de lograr que el algoritmo reconozca la importancia de hacer dos jugadas seguidas, cosa que no puede representarse en un árbol minimax, por que alterna entre los dos, cosa que se solución ponderando los cuadrados de nivel 3 por sobre los de nivel 2, ya que inmediatamente después de hacer esta jugada podrá llenar el cuadrado que le sigue. También lograr que pueda buscar la mejor alternativa, cosa que fue solucionada de la mejor manera posible en la heurística.

Proceso del Juego:

Al inicializarse, se le aparecerá en pantalla la interfaz del juego:



En el caso donde sea turno de la máquina, el turno se controla tocando el botón "Next AI Move", que solo puede ser accedido cuando es turno de la máquina, que ejecuta el algoritmo minimax y elige la línea, reemplazándola en el tablero. Si es el turno del usuario, este se ejecuta haciendo click en la línea que quiere ser elegida que llamara al método move y luego de verificar que esté disponible, la reemplazara y verificará el cuadrado, AI Move no puede ser ejecutado en este caso. Si se quiere volver a la última jugada del jugador actual, presione "Undo", y si se quiere guardar el juego, presione "End Game". Al finalizar el juego un pop-up con el ganador del juego le aparecerá. Si se desea salir prematuramente del juego basta presionar "Exit".

Conclusión:

En conclusión, seria óptimo poder implementar un front end mas user-friendly, dado que el que pude armar ahora no tiene el funcionamiento esperado como por ejemplo poder hacer que el cuadrado tome el color del usuario. También poder definir mejor la separación del MVC, y finalmente probar haciendo el cambio a la estructura propuesta de líneas representadas como booleanos, para ver si recorre más rápidamente el tablero.