Informe del Trabajo Práctico

Estructura de Datos y Algoritmos

Elección de la estructura de datos

Para elegir la estructura de datos a utilizar para el trabajo, pensé en 3 posibilidades, de las cuales una fue eliminada por no ser eficiente temporalmente mientras que se distinguió a la ganadora por su facilidad de implementación aunque es menos eficiente espacialmente con respecto a la restante.

- La primer idea que se me ocurrió para representar el tablero fue utilizar una estructura de grafos. Siguiendo esta idea, es muy fácil representar el tablero ya que podemos distinguir los objetos del tablero como nodos y aristas.
- La segunda idea fue utilizar una matriz en donde cada casillero iba a representar a cada cuadrado, por lo que cada casillero contiene: un valor entero para guardar su color (que también abre la posibilidad a más de 2 jugadores) y 4 datos de objeto Boolean para indicar si los bordes de este cuadrado fueron o no marcados (que van a ser compartidos por sus vecinos).
- La última idea fue en tener 2 matrices para representar, en una las aristas horizontales, en otra las verticales y en la última, los cuadrados. Las primeras dos son de tipo boolean mientras que la última, de tipo entero, lo que también permite la posibilidad de que hayan más de 2 jugadores.

De las 3 ideas, la primera fue descartada. La razón para esto fue porque el algoritmo para saber si se llenó un cuadrado era mucho menos eficiente que las otras 2 propuestas. En esta implementación, cuando se crea una nueva arista, se debería realizar un recorrido del estilo DFS para ver si hay uno o más ciclos, que representarían los cuadrados completos[1] (Similarmente ocurriría si el recorrido fuera BFS).

De las dos ideas restantes, decidí utilizar la segunda opción ya que es más fácil para manejarse y para verificar si hay o no *cadenas*, elementos que sirven para distintas estrategias en le juego.

Luego de varios intentos, pude notar que el lenguaje de Java no soporta que se compartan los objetos Boolean (ya que siempre crea nuevas instancias), por lo que hay que descartar la idea de tener los lados compartidos y hacer que cada cuadrado tenga 4 Booleans que indiquen si el lado esta o no marcado, teniendo en cuenta que si marcamos un lado, tenemos que marcar el mismo de su cuadrado adyacente.

Además, cada cuadrado contiene un entero que indica su color y la cantidad de aristas activas que tiene, para que sea más fácil luego hacer cálculos con estos.

Estrategias y Heurísticas de la IA

La heurística a utilizar es simplemente la resta de: las cajas que he completado – las cajas que mi contrincante ha completado. En un principio analicé la posibilidad de utilizar otras heurísticas que evalúen la existencia de cadenas o de “cuadrados dobles”, elementos que son estratégicos para el juego. Pero luego, por una falta de tiempo, decidí mantenerlo simple.

1 Si bien se puede optimizar el recorrido DFS poniendo como límite la altura de la profundidad igual a 4, de todas formas se revisarían al menos todos los vértices que son adyacentes a los vértices del cuadrado en cuestión que se está revisando (Ver imagen 1 en Anexo).

Lautaro Pinilla (57504)

Decisiones de diseño para la implementación del minimax

Para la implementación del algoritmo utilicé un modelo “estándar” en donde en cada paso el algoritmo devuelve un paquete del mejor tablero y su puntaje.

De esta manera, el algoritmo puede trabajar perfectamente utilizando los puntajes y así devolver al mismo tiempo la mejor jugada.

Problemas encontrados en la implementación del algoritmo

El principal problema al desarrollar el algoritmo fue el manejo de la recursividad y de la variable de retorno, ya que de esta dependía la jugada a elegir por el algoritmo.

Mi problema fue que, por un descuido en el manejo de variables, devolvía el último nodo del árbol, lo que hacía que el algoritmo devuelva un tablero en el que haya jugado n jugadas, siendo n la profundidad del árbol, por más que éste no haya completado ninguna caja.

La solución por suerte fue sencilla y fue resuelta devolviendo el tablero adecuado.

Posibles extensiones

Como posible extensión primero revisaría el Hashcode por una variación del Zobrist Hashing para utilizar tablas de trasposición durante el recorrido del minimax para recortar aún más la cantidad de nodos si ya hemos analizado esa configuración previamente.

Además, se podría revisar un tipo de simetría que es el siguiente: supongamos que tenemos la opción de colocar una arista lateral en una caja y que esta no la completa, se podría decir en algunos casos reducidos que colocarla en un lateral o en otro son jugadas simétricas.

Lautaro Pinilla (57504)

Anexo

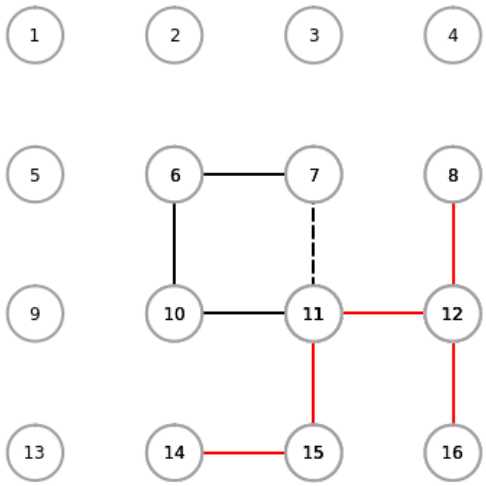

Imagen 1: si la última arista que pusimos fue la arista 7-11, completando el cuadrado, entonces desde 11 deberíamos hacer un recorrido DFS para encontrar un ciclo hasta 7. En rojo podemos ver algunos de los nodos que revisa de más en dicho recorrido solo partiendo de 11.

Lautaro Pinilla (57504)