Materia: Estructura de datos y algoritmos

Docentes:

- Frias Marcelo Fabian
- Itzcovich Ivan
- Sal-lari Julieta
- Liberman Daniella

Trabajo práctico especial:.

Alumnos:

- Bossi, Agustín - legajo: 57068
- Dorado, Tomás - legajo: 56594
- Martín, Fernando - legajo: 57025

Estructuras

Tablero

El tablero está compuesto por un entero que guarda el tamaño, una matriz de enteros que guarda referencia a los cuadrados del tablero con el número de aristas libres a su alrededor, una LinkedList para los movimientos realizados y un LinkedHashSet para los movimientos aún posibles en el juego, que se eligió para que se guarde el orden con el que se ponen, porque antes de ponerlos se le hace un shuffle para que no tengan el mismo orden en todas las partidas. Dado que los movimientos iban a ser una de las colecciones más consultadas a lo largo del código, se pensó que lo mejor era utilizar una estructura que permitiera la consulta en orden constante. También se cambió que antes se tenía un HashSet de movimientos realizados y un stack para deshacer movimientos, pero esto era tener lo mismo en dos estructuras diferentes basicamente, asi que se descartó el stack y únicamente nos quedamos una lista en lugar de HashSet ya que no era necesario. Otro cambio fue que para los cuadrados antes teniamos un arreglo y era mucho más cómodo hacerlo con una matriz, así que se realizó el cambio.

Juego

El juego está conformado por dos jugadores (clase Player), un tablero, una variable entera para el manejo de los turnos, otra variable entera para el tipo de AI.

Jugador

La clase jugador consiste en un entero que guarda puntaje y una referencia al juego, a través de la cual se pasan los movimientos realizados y se accede al tablero para el manejo de los Sets de movimientos.
También se tiene la clase AIPlayer que extiende de Player, como tiene que usar los mismos métodos, para realizar los movimientos, una vez que los elige, y también tiene que tener un puntaje.

Guardado de partidas

Para el guardado de partidas se decidió realizar una serialización utilizando las clases de serialización de java, ya que esto se aprendió en POO bastante bien.

Movimiento

Move

Clase sencilla, getters, equals, toString y hashCode como métodos. Las jugadas están compuestas por cuatro enteros, rowFrom, rowTo, colFrom y colTo. En caso de ser una jugada realizada va a tener también el dato de player, que jugador la realizó, y pointsDone, la cantidad de puntos que hizo que gane ese jugador. Esto último se utiliza simplemente para el deshacer movimientos que fue requerido.
Dado que para verificar si un movimiento fue realizado lo que se hace es ver si esta o no en el LinkedHashSet de posibles movimientos del tablero, el hashCode y equals de esta clase está implementado de manera que únicamente utiliza rowFrom, rowTo, colFrom y colTo. Ya que lo otro no nos interesa para esta utilidad.
Una posible mejora para esta clase sería en lugar de guardar de qué nodo sale a qué nodo va, seria guardar que arista es, con dos int, y un boolean que indique si es vertical o horizontal.

Guardado y cargado de partidas

Está desarrollada la serialización del juego, de manera que cuando se guarde una partida se va a guardar en un archivo binario, lo mismo al cargarlo.

Parte visual

En cuanto a la parte visual se tuvieron grandes problemas debido a que ninguno de los integrantes poseían ningún tipo de conocimiento sobre algún framework visual. En cuanto a lo desarrollado, generamos una grilla con círculos referenciando a los nodos del juego, y las aristas las colocamos con rectángulos. Estas últimas se diferencian con un color de acuerdo al jugador que haya realizado el movimiento.

Hay un botón de undo move, que deshace el último movimiento, un botón make dot file, que toma el texto del TextFile que está arriba a la derecha, y crea el archivo .dot con el nombre que se le ponga en ese campo, lo mismo para el botón save game, solo que en lugar de un archivo .dot lo crea haciendo la serialización ya hablada y con ese nombre en la carpeta target.

Para los turnos, si la partida es de AI vs AI se puede hacer de dos maneras, o hacer clic en el botón next turn o hacer clic en cualquier parte de la pantalla, y se realiza el movimiento.

Si la partida es de Jugador vs Jugador, simplemente en el turno de cada jugador hacer clic en la arista que se quiere usar y el movimiento se realiza solo.

Por último si es Jugador vs Computadora, luego de que el jugador haga clic en la arista que quiere que sea su movimiento, se llama automáticamente al movimiento de la computadora luego del de jugador y se realiza.

MinMax y heurística

El problema con el minimax fue que no eran pasos de simplemente un movimiento, sino que pueden ser de varios, ya que al ganar un punto el jugador debe seguir jugando, entonces lo que hicimos fue hacer dos método ya sea para el tiempo o para profundidad, en donde uno agrega jugadas de 1 solo movimiento, y el otro agrega jugadas de más de 1 movimiento, y se decide cual usar si cuando se hace un movimiento el turno no cambio ahí se llama al método que hace más de 1 movimiento.

Por profundidad

Para cada recursión del minimax al principio lo hicimos de manera que cree una copia de la instancia del tablero, pero esto no solo consume mucha memoria, sino que tardaba mucho más tiempo ya que tenia que hacer una copia profunda. Lo elegido finalmente fue que después de cada recursión se haga la función deshacer de los movimientos realizados correctamente.

Por tiempo

Lo que se hizo fue que por cada nivel busque todos los movimientos posibles, y los guarde en una pila y de ahí en adelante tomando de la pila se va repitiendo esto, hasta llegar al tiempo límite o hasta completar todo el árbol. Cabe destacar que a diferencia del minimax por profundidad este requiere guardar una instancia de todos los valores del juego en cada nodo del árbol.
Luego de completar todos los movimientos posibles en ese tiempo, simplemente se hace una recorrida del árbol que se llegó a formar para decidir qué movimiento fue el elegido.

La función heurística fue simple retorna el puntaje del jugador que la llama menos el puntaje del jugador contrario.

Como se debe poder hacer un archivo .dot con todos los nodos en el formato pedido, lo que se hizo fue una clase que guarde cada movimiento y las siguientes instancias de ese movimiento. Esto se va guardando siempre a medida que se hace el minimax de manera que siempre quede guardado como se eligió el movimiento anterior de la computadora.