PROGRAMACIÓN III

UNIVERSIDAD NACIONAL GENERAL SARMIENTO

TRABAJO PRÁCTICO 1: 2048

Federico Atanasoff

Marcos Vera

Introducción

Este trabajo práctico consiste en crear una grilla de 4 × 4, y en cada posición se tiene un número natural. En cada turno, el jugador presiona una de las teclas de dirección y los números se mueven en la dirección asociada a la tecla en cuestión. En cada turno, aparece un nuevo número en la grilla, que puede ser 2 o 4. Los números se mueven lo más posible en la dirección especificada por el usuario, hasta que se encuentran con otro número o con el borde de la grilla. Si dos números iguales colisionan, entonces se combinan en una única celda con la suma de los dos, y esta nueva celda no se combina con otras en el mismo turno. El objetivo es obtener una celda con el valor 2048 para ganar el juego.

Nuestro proyecto consta de dos paquetes “juego” y “gui”, donde el primero almacena la lógica del juego, y el segundo la interfaz gráfica, procederemos a explicar cada clase.

Lógica del juego, clase Juego2048

Creación del tablero:

En esta parte, creamos una matriz de enteros y la inicializamos con ceros y dos números aleatorios (estos números son generados por la función generarNumero();)

Movimientos:

Una vez con la matriz inicializada, para realizar algún movimiento direccionado, usamos el uso de las siguientes funciones para controlar el estado de la matriz:

moverIzquierda()

moverIzquierda()

moverAbajo()

moverArriba()

Por ejemplo a la hora de realizar un movimiento a la izquierda, lo primeros que debemos corroborar es si tenemos algún movimiento valido, esto lo hacemos con la función movimientoValidoIzquierda() que recorre la matriz de izquierda a izquierda buscando un número, para verificar si en la posiciones siguiente hay un espacio que puede ocupado o un número igual para ser sumado. Ya encontrado un movimiento valido, desplazamos todos los números hacia la izquierda, esto lo realizamos buscando en la fila el primer lugar vacío. Una vez terminado este procedimiento, invocamos a la función SumaDer(); que se encarga de recorrer la matriz para verificar si dos números contiguos son iguales para así sumarlos y en este caso mover la suma a la posición contigua y en la posición previa poner un número cero. En algunos casos donde generamos sumas queda algún espacio vacío donde debemos nuevamente correr a la izquierda para corregir este tipo de errores, esto lo solucionamos creando la función moverNumerosIzquierda(); mueve a la izquierda si queda un espacio vacío entre sumas que hicimos previamente.

Los restantes métodos para realizar otro movimiento, utilizamos la misma lógica pero recorriendo de diferentes formas la matriz.

Generar un numero aleatorio en el tablero:

Para poder generar un numero en el tablero utilizamos la función generarNumero(), lo que hacemos es recorrer dicho tablero y guardar las coordenadas en dos ArrayList, uno que hace referencia a las filas y otro a las columnas. Lugo con un random, seleccionamos dos números de la misma posición en los dos ArrayList y seteamos el valor en el tablero.

Verificar si el usuario pudo ganar:

Para ver si el participante gano, creamos un método llamado ganador(), que recorre la matriz en busca en busca de un valor 2048, si lo encuentra retorna true si no, false.

Reiniciar la partida:

Para poder reiniciar la partida, requerimos de un método que simplemente recorra e todos los valores de la matriz poniéndolos en cero y generando dos números en posiciones aleatorias.

Dificultades que nos encontramos en la lógica del juego:

- Para realizar un movimiento teníamos que tener en cuenta que haya un lugar vacío hacia la dirección a donde movemos.
- Tuvimos que recorrer la matriz de diferentes formas a la hora de realizar un movimiento, suma, o verificar si un movimiento es válido.
- Nos encontramos que en algunos casos cuando sumábamos nos quedaba un lugar vacío en el medio, lo solucionamos creando otra función que vuelva a mover hacia la dirección correspondiente.
- Tuvimos problemas a la hora de realizar los puntajes, lo solucionamos sumándolo a una variable, y luego a un ArrayList.

Interfaz del juego, clase InterfazGrafica:

En esta parte, creamos una matriz de JButton para poder crear el tablero. Luego lo que hicimos fue extraer todos los valores de la matriz que se encuentra en la clase Juego2048 con el método tablero(); que simplemente recorre la matriz de enteros y va seteando todos los valores en la matriz de JButton.

Movimientos en el tablero:

Para ver los movimientos en el tablero, hicimos un método jugar() que captura los eventos del teclado. Ya con el evento del teclado capturado, le pasamos el código de la tecla presionada al método movimientos(), que realiza un swich para ver que método ejecutar de los movimientos de la clase Juego2048.

Ventanas del ganador y del perdedor:

Cuando se llega a formar un 2048 en el tablero, esto lo detectamos con el método ganador() de la clase Juego2048, lanzamos un **`JOptionPane`** para avisar que gano, y preguntar si quiere seguir jugando o no. Lo mismo hicimos para ver si el usuario perdió, pero esta vez lanzamos un **`JOptionPane`** que avisa que perdió y pregunta si quiere reiniciar la partida, si oprime que si llamamos al método reiniciar(), reiniciarScore() y tablero() para actualizar.

Botón new game:

Creamos un JButton para iniciar un nuevo juego, primero capturamos el evento cuando el usuario clickea sobre el botón con el método mouseClicked(MouseEvent) , luego hacemos uso nuevamente de JOptionPane para preguntar si realmente desea reiniciar el juego, si oprime que si llamamos al método reiniciar, reiniciarScore, actualizamos el tablero con el método tablero() y al método actualizarScore() que setea en el JLabel que se muestra en la ventana.

Manejo del Score y HighScore:

Para el manejo de estos datos, lo que hicimos fue a medida que se encontraba una suma en el tablero, la íbamos acumulando en una variable para luego ser mostrada en la ventana. Decidimos guardar el score en el caso de que el jugador gana y no decida seguir jugando y en el caso de que pierda y salga del juego. Para que este dato sea consistente decidimos guardarlo en un archivo txt donde accedemos a él para leer todos los puntajes y para agregar el score de otra partida. Una vez con estos datos, para un manejo un poco más óptimo manejamos todos estos puntajes en un array de enteros que luego ordenamos de forma descendente para así mostrar los primeros 5 mejores puntajes desde el botón mostrar resultado que nos devuelve un panel con los mejores resultados.

Dificultades que nos encontramos en la interfaz gráfica:

- A la hora de mostrar los valores en el tablero, nos tuvimos que manejar con coordenadas ya que no encontramos la manera de hacerlo de manera responsive, lo mismo a la hora de dibujar el resto de los paneles.
- Cuando implementamos los botones, perdíamos el foco del panel del contexto en el que se estaba ejecutando.
- Hay veces que hay que hacer más de un click en un botón y no tenemos idea del por qué.

Conclusión

El trabajo costo más de lo que imaginábamos, ya que no contemplamos todas las condiciones que requerían los movimientos, sumas y validaciones. En cuanto a la interfaz si bien, swing nos proporciona bastante ayuda a la hora de trabajar con componentes puede llegar a ser muy dificultoso a la hora de querer realizar algo más escalable ya que tiene sus particularidades.