Programación III - Primer Semestre 2024-com01

1er Trabajo Práctico: Juego 2048

16 ABRIL

Creado por:

- Gabriel Acuña
- Dario Espínola
- Ana Lucia Marzocca
- Brian Sosa

Profesores:

Patricia Bagnes, Ignacio Sotelo.

UNGS

Introducción

Este trabajo consiste en realizar el juego on-line: “2048 Game” (juego 2048) utilizando la creación de clases, paquetes, y en especial la herramienta windowBuilder, entre otros conceptos visto en clase, implementando el lenguaje de programación JAVA.
El juego “2048”, consta de una grilla de 4×4, que al comienzo del juego aparecerán en forma aleatoria dos números los cuales pueden ser 2 o 4. En cada turno, el jugador presiona una de las flechas del teclado y todos los números se mueven en la dirección especificada por el usuario. En cada turno, aparece un nuevo número en el tablero de juego, que puede ser 2 o 4. Los números se mueven hasta el tope (hasta que se encuentran con otro número o con el borde del tablero) en la dirección especificada por el usuario. 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. El juego se da por perdido cuando no quedan celdas vacías y no se llegó a la suma de 2048 en uno de los cuadrantes.

Figura 1. Pantalla del juego realizado por los autores de este informe.

Descripción

Para la realización del proyecto, se utilizó la lógica de Forms and Controls, se separó el mismo en dos paquetes, una dedicada exclusivamente a la interfaz de usuario (GUI) y otra encargada de la lógica del juego (negocio). Con esto se logró garantizar la estabilidad ante cambios en la capa de alguna de ellas, ya que se evita un alto acoplamiento. Se decidió crear dos paquetes distintos con funciones bien diferenciadas.

a) Paquete llamado **interfaz**. Este contiene:
- La clase **PantallaInicial.java**, es una pantalla de inicio al juego realizada con el plugin WindowBuilder dentro de la aplicación Eclipse para diseñar las interfaces gráficas en donde se colocó un botón de “Play!” para que el jugador de comienzo al juego.

Esta clase tiene los siguientes métodos y componentes más importantes:

- Constructor PantallaInicial() *//Constructor que inicializa la pantalla inicial llamando al método initialize().*
- Método initialize() *//se inicializa el contenido del frame. Establece el tamaño, título y otros aspectos visuales del frame.*
- Se agregan componentes como JLabel (para mostrar el título del juego) y JButton (para iniciar el juego).
- mostrarPantallaJuego() *//Este método se llama cuando se hace clic en el botón "Play!".*
- *Se crea una instancia de PantallaJuego // La clase donde se desarrolla el juego*
- getFrame() *//método que devuelve el frame actual de la pantalla inicial. Permite volver a esta pantalla desde otras partes, como al dar "aceptar" al cartel que surge al jugar cuando de pierde.*

- También este paquete tiene la clase **PantallaJuego.java** que trata sobre la representación visual del juego en sí, para su construcción se utilizó nuevamente WindowsBuilder.

Esta clase tiene los siguientes métodos y componentes más importantes:

- *Se define la clase PantallaJuego que extiende JFrame e implementa la interfaz KeyListener.*
- *Se declaran las siguientes variables de instancia: juego, botones, panelTablero, y textPuntajeActual.*
- Método main(String[] args) *//Crea una instancia de PantallaJuego y la hace visible.*
- Constructor PantallaJuego() *//Inicializa la pantalla del juego llamando al método initialize(). En él se configura la ventana para que no sea redimensionable y un glassPane para prevenir que los eventos de clic interfieran con el enfoque de la ventana.*
- initialize() *//Configura la apariencia de la ventana del juego, se crea un panel de juego con un diseño de cuadrícula 4x4 y lo agrega a la ventana. Además, crea y posiciona los botones del tablero en el panel de juego, agrega el título, un campo de texto para mostrar el puntaje y una imagen de fondo. Por último, llama al método actualizarTablero().*
- limpiarCoordenadas() *//limpia las etiquetas y los colores de fondo de los botones.*
- actualizarTablero() *//Actualiza la interfaz de usuario del tablero utilizando los valores de la matriz del juego. También en él se recorre la matriz, establece el texto y el color de fondo de cada botón según el valor de la celda. Por último, llama al método actualizarPuntaje().*
- cambiaColorCelda(int valor) *//Devuelve un color específico según el valor de la celda*.
- obtenerColorTexto(int valor) *//Devuelve el color del texto según el valor de la celda*.
- verificarEstadoJuego() *//Verifica si el juego está ganado o perdido, por lo que muestra el mensaje correspondiente, luego limpia la matriz del juego y muestra la pantalla inicial.*
- mostrarPantallaInicial() *//Crea una instancia de PantallaInicial, la hace visible y cierra la ventana actual.*
- Implementación de KeyListener *//Define los métodos keyTyped, keyPressed y keyReleased. keyPressed, maneja los eventos de teclado para mover el juego en la dirección correspondiente, luego actualiza el tablero y verifica el estado del juego.*

Implementación de los objetivos opcionales no obligatorios:

- actualizarPuntaje() *//Actualiza el campo de texto del puntaje con el valor actual del puntaje del juego.*
- En el constructor PantallaJuego se coloca un temporizador (Timer) *// ejecutará una acción cada cierto tiempo definido por la constante DELAY. La acción que se ejecuta cada vez que el temporizador dispara es proporcionada por una clase anónima que implementa la interfaz*

ActionListener. En este caso, la acción consiste en llamar al método actualizarPosicionesRecomendadas(). Finalmente, se inicia el temporizador con el método start().

- actualizarPosicionesRecomendadas() *//Este método se llama cada vez que el temporizador llega a 7seg aproximadamente. Obtiene las posiciones recomendadas para una jugada utilizando el método obtenerPosicionesJugadaRecomendada() del objeto juego. Si se encuentran dos posiciones recomendadas (dos números adyacentes): Cambia el color de fondo de los botones en la matriz de juego (botones) en esas posiciones a amarillo (YELLOW) para resaltarlas, emite un sonido y coloca un cartel en la parte inferior, el cual dice: "JUGADA RECOMENDADA". Como se puede observar en la Fig. 1.*

- Por último, contiene la clase **Sonido.java**, este contiene el método:
  - reproducirSonido(String rutaArchivo), el cual, como su nombre lo indica, tiene como objetivo reproducir los sonidos de: movimiento con el teclado, las sumas logradas, pierde o gana y la jugada recomendada.

b) Paquete llamado **negocio**. Este contiene:

✓ Una clase llamada **Juego.java**, la cual se utiliza para manejar toda la representación lógica del juego, su estructura de datos.

Esta clase tiene los siguientes métodos y componentes más importantes:

- Constructor Juego() *//Inicializa una instancia de juego llamando al método iniciarMatriz() y estableciendo el puntaje inicial en 0.*
- iniciarMatriz() *//Rellena la matriz con dos valores aleatorios de 2 o 4 en posiciones aleatorias. Utiliza el método obtenerPosicionRandomDisponible() para obtener posiciones aleatorias dentro de la matriz y el método obtenerValorRandom() para obtener valores aleatorios de 2 o 4.*
- obtenerMatriz() *//Devuelve la matriz actual del juego.*
- definirMatriz(int[][] matrizParam) *//Permite definir una matriz específica para propósitos de pruebas.*
- obtenerPosicionRandomDisponible() *//Devuelve una posición aleatoria disponible en la matriz.*
- obtenerValorRandom() *//Devuelve un valor aleatorio de 2 o 4.*
- Métodos de Movimiento *//tiene los métodos: moverArriba(), moverAbajo(), moverIzquierda(), moverDerecha(): controlan el movimiento de los elementos en la matriz hacia arriba, abajo, izquierda y derecha respectivamente. Cada uno de estos métodos mueve los elementos y combina celdas si es posible, luego llama al método agregarNuevaFicha() si hubo algún movimiento. También están los métodos moverElementosArriba(), moverElementosAbajo(), moverElementosIzquierda(), moverElementosDerecha() que realizan el movimiento y la combinación de celdas.*
- incrementarPuntaje(int puntaje) *//Incrementa el puntaje actual sumando el puntaje pasado como argumento.*
- agregarNuevaFicha() *//Agrega una nueva ficha de valor 2 en una posición aleatoria disponible.*
- obtenerRandom(ArrayList<Integer> list) y obtenerPosicionRandom(ArrayList<Posicion> list) *//Devuelven un valor o una posición aleatoria de una lista dada.*
- juegoGanado() *//busca si hay una celda con el valor 2048.*
- juegoPerdido() *//busca si hay posibles combinaciones de celdas en todas las direcciones.*
- limpiarMatriz() *//Limpia la matriz estableciendo todas las celdas en 0.*

Implementación de los objetivos opcionales no obligatorio:

- obtenerPuntaje() *// Devuelve el puntaje actual del juego.*

- obtenerPosicionesJugadaRecomendada() *//Retorna un array con las posiciones de una jugada recomendada, es decir, si hay celdas adyacentes que puedan combinarse y el jugador no se da cuenta de realizar la jugada.*

✓ Clase **Posicion.java** para ser utilizada en la clase Juego.java, se utiliza en Juego.java para dar la posición del tablero de juego y en **Celda.java**.

✓ Clase **Celda.java** para ser utilizada en la clase Juego.java, se utiliza en Juego.java para obtener la Posicion y el valor del cuadrante.

✓ Clase **Recomendacion.java** se utilizan diversos métodos para obtener la jugada recomendada

También se agregó un paquete con los Test realizados a la sección de negocio (Juego.java)

- @BeforeEach setUp(): se ejecutará antes de cada prueba (@Test). Su objetivo es inicializar la instancia de Juego (juego) antes de ejecutar cada prueba, asegurando que cada prueba se ejecute en un estado limpio y consistente.
- @Test obtenerValorRandom(): verifica si devuelve uno de dos valores específicos: 2 o 4. Utiliza la aserción assertThat de JUnit combinada con anyOf y equalTo de Hamcrest para realizar esta verificación.
- @Test obtenerPosicionDisponible(): Define una matriz con celdas ocupadas y vacías, y comprueba si el método devuelve una posición disponible correctamente.
- @Test moverArriba(), moverAbajo(), moverIzquierda(), moverDerecha(): Estas pruebas verifican los métodos mensionados. Se establece una matriz inicial, se realiza un movimiento y se comprueba si la matriz resultante es la esperada después del movimiento.
- @Test incrementarPuntaje(): verifica si incrementarPuntaje() incrementa el puntaje correctamente después de realizar un movimiento.
- @Test obtenerRecomendacion(): verifica si devuelve la recomendación de jugada correctamente en una matriz dada.
- @Test juegoGanado(): verifica si identifica correctamente si el juego ha sido ganado. Se prueba con matrices que contienen o no el valor 2048 en alguna celda.
- @Test juegoPerdido(): verifica si identifica correctamente si el juego ha sido perdido. Se prueba con matrices que no pueden realizar ningún movimiento válido.

Dificultades en la implementación del proyecto

Para la realización de este proyecto en grupo, se tomó la decisión de trabajar con un sistema de control de versiones (VCS), en este caso Github.

- Al comienzo del proyecto se encontraron problemas respecto a módulo no encontrado: game_2048 (el nombre de nuestro proyecto). Se creó un nuevo repositorio y un nuevo proyecto, pero sin seleccionar la opción de “Create module-info.java” la cual podía ser la causa del problema, sin embargo, el problema persiste por lo que al investigar sobre esto en más profundidad, se llega a la conclusión de que es un problema de dependencias de bibliotecas. Al eliminar las dependencias, se elimina el problema y el proyecto se puede ejecutar. El problema residía en el archivo .classpath, el cual hace referencia a una serie de bibliotecas que existen en la notebook del creador del proyecto en Github el cual utiliza Ubuntu como Sistema Operativo por lo cual las rutas de las bibliotecas no funcionan para quienes utilicen Windows.
- Surgieron problemas al colocar el ícono del juego en el extremo izquierdo de la pantalla, se intentó solucionar cambiando el código y la ubicación de la carpeta, pero no se pudo solucionar.

- Fue necesario modificar la modalidad de trabajo con la lógica del juego de estática a clase instancias. Eso requirió modificar los test suite y en la interfaz de usuario, instanciar un objeto de juego, quitar la responsabilidad de inicializar la matriz al juego, entre otros.
- Ocurrieron problemas al adaptar código entre compañeros de proyecto, los mismos fueron solucionados y se reutilizaron los métodos obtenerPosicionRandomDisponible en agregarNuevaFicha. Se adapta la variable “size” por matriz.length definida al inicio de la función.
- Se agregó como *objetivo opcional no obligatorio*, la funcionalidad de jugada recomendada, en la cual se creó una estructura de Celda.java incluyendo las propiedades posición (con objeto posicion) y un valor (integer).
- Se tuvieron problemas al hacer los casos de test, sirvieron para detectar errores de runtime, por ejemplo el out of index y falsos positivos: el juego recomendaba dos celdas con valor 0 porque eran iguales.
- Otras de las situaciones surgidas durante la implementación del juego fueron los errores provocados al hacer clic con el mouse en la ventana del juego mientras se realizaban los movimientos con el teclado. La solución fue implementar un panel “protector” colocado en el constructor para evitar que los eventos producidos por el clic del mouse perjudiquen al correcto desarrollo del juego. Además, se agregó un método de frame setResizable(false), para evitar que se redimensione el alto y ancho de la ventana del juego.
- Surgieron problemas al lograr el puntaje acumulado, por el jugador, ya que el mismo no se actualizaba, ya que se llamaba incorrectamente al método en Juego.java.
- Para realizar el objetivo opcional: “Hacer una sugerencia de la próxima jugada”, al realizar el Timer de la Jugada Recomendada, se agrega el timer para jugada recomendada ya funcionando. Se activa aproximadamente en 7 segundos si no se realiza ninguna acción. Otro de los problemas que surgieron, fue la demora en limpiarse las coordenadas que iban a ser utilizadas para guiar al jugador. Se soluciona este problema, sacando el código de limpiar coordenadas que estaba dentro del método actualizarPosicionesRecomendadas el cual se activaba cada 10 segundos, por lo que también tardaba 10 segundos en desaparecer. Por lo tanto, se lo extrae como método limpiarCoordenadas y se lo llama inmediatamente después de apretar una tecla, solucionando así el problema. Sin embargo, se toma la decisión de no colocar coordenadas en la pantalla de juego para avisar al jugador la jugada recomendada y se opta por un cartel que de aviso de la “marcación” de esta jugada en color amarillo, como muestra la Fig.1. Por último, se extrae el método reproducirSonido a una clase "Sonido" para no cargar la clase pantallaJuego con métodos no correspondientes a su objetivo principal.

Conclusiones

En conclusión, el desarrollo del juego utilizando el lenguaje de programación JAVA ha sido una experiencia interesante para realizar como trabajo en equipo. A lo largo de este proyecto, hemos podido explorar y aplicar diversos conceptos de programación vistos en clase como el uso de WindowBuilder, el uso en mayor profundidad del plugin Junit para las pruebas mediante Test unitarios, la utilización correcta de los nombres a los métodos, el uso Forms and Controls, para separar la interfaz gráfica del negocio, el uso y aprendizaje de VCS (Github) por parte de algunos miembros del grupo.

Durante el proceso de desarrollo, nos enfrentamos a diferentes obstáculos y desafíos, ya mencionados, de los cuales pudimos resolver por nuestra cuenta o al realizar consultas a los profesores. A medida que avanzábamos, nos dimos cuenta de la importancia de la planificación y el diseño adecuado del juego, así como de la organización eficiente del código.

Aprendimos a dividir tareas, asignar responsabilidades y compartir conocimientos para aprovechar al máximo nuestras habilidades individuales.