TRABAJO PRACTICO

HISTERIA

UNIVERSIDAD

- Universidad Nacional General Sarmiento
- Materia: Organización del computador
- Comisión: 01

ALUMNOS

- Mayra Rossi ()
- Melina Scabini (44.756.058) Melina.Scabini@gmail.com

PROFESORES

- Patricia Bagnes
- Ignacio Sotelo

Introducción

El objetivo de este trabajo práctico es implementar una aplicación visual para el juego "Histeria", donde el jugador debe colorear completamente una grilla sin que celdas vecinas tengan el mismo color. Inicialmente, todas las celdas están sin colorear (gris), y cada clic asigna un color aleatorio de entre 6 posibles a una celda seleccionada. Sin embargo, si el nuevo color coincide con el de una celda vecina, tanto la celda clicada como sus vecinas se "apagan" (vuelven a gris).

La aplicación fue diseñada en Java, siguiendo un enfoque modular que separa la lógica del juego (paquete Model) de la interfaz gráfica (paquete View), con una capa de control (paquete Controller) que coordina ambas partes mediante el uso del patrón Observer para la comunicación de eventos. La integración de Observers nos permitio que las vistas reaccionen a cambios en el modelo sin acoplamiento directo, mejorando la modularidad y facilitando la extensión del sistema.

Además de cumplir con los requisitos mínimos, se implementaron características opcionales como el conteo de clics y la posibilidad de ajustar el tamaño de la grilla para diferentes niveles de dificultad. Este documento describe la estructura del proyecto, las decisiones tomadas durante el desarrollo y los desafíos enfrentados, con el propósito de ofrecer una visión clara del proceso llevado a cabo por el equipo.

Decisiones de Desarrollo

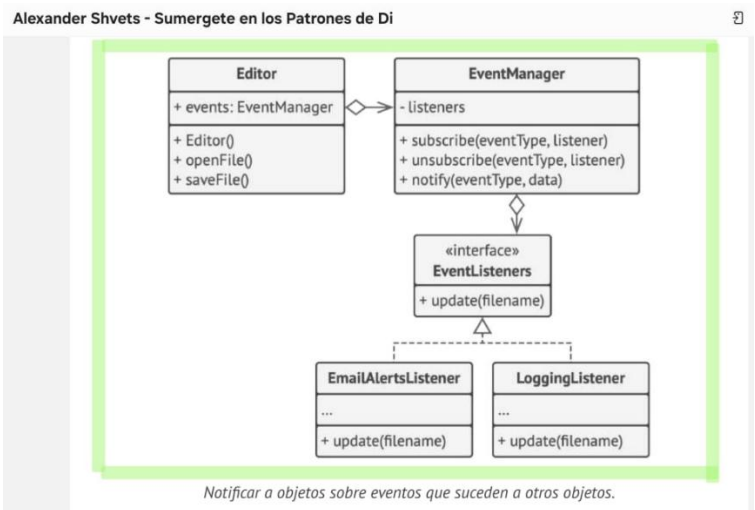
Durante el desarrollo del juego "Histeria", se tomaron varias decisiones clave para optimizar la implementación, garantizar la modularidad y cumplir con los requisitos del trabajo práctico. A continuación, se detallan las principales:

- **Detección de victoria con una variable de conteo**: Para determinar si el jugador ha ganado (es decir, si todas las celdas están coloreadas sin conflictos), se optó por usar una variable totalGrayCells en BoardModel en lugar de recorrer la matriz en cada turno. Esta variable se inicializa con el número total de celdas y se actualiza dinámicamente en updateTotalGrays cada vez que una celda cambia de color: se decrementa cuando una celda gris se colorea y se incrementa cuando una celda vuelve a gris por un conflicto. La victoria se detecta en checkWin cuando totalGrayCells llega a cero, logrando una complejidad constante O(1) en lugar de $O(n^2)$ que implicaría recorrer la matriz completa cada vez que se hace un clic.

Diseño de la vista del tablero: En *BoardView*, dimos uso de GridLayout que nos elimina la necesidad de manejar manualmente la posición de cada botón, reduciendo la complejidad del código.
Cada botón se almacena en la matriz gridButtons, facilitando el acceso y modificación de las celdas sin necesidad de recorrer la interfaz gráfica.

Cada celda se representa con un JButton almacenado en una matriz gridButtons, lo que permite asignar colores mediante setBackground y capturar clics con ActionListener.

- **Manejo de eventos con EventManager:** La comunicación entre el modelo y las vistas se implementó mediante un sistema de eventos gestionado por EventManager en el paquete Observer.
  En BoardModel, se instancia un EventManager que mantiene una lista de observadores (vistas) suscritos a eventos específicos definidos en el enum EventType (START, UPDATE_BOARD, WIN, REPLAY). Cuando ocurre un cambio relevante, como iniciar el juego, actualizar una celda o detectar una victoria, el BoardModel invoca notify con el tipo de evento correspondiente avisándole a las vistas suscriptas que tienen que proceder con la acción que definieron en el método update.
  Con este enfoque, inspirado en el libro 'Sumérgete en los Patrones de Diseño' de Alexander Shvet nos aseguramos de que cada vista reciba solo la información que es importante para su funcionalidad, mejorando la escalabilidad del proyecto de forma sencilla y eficiente.



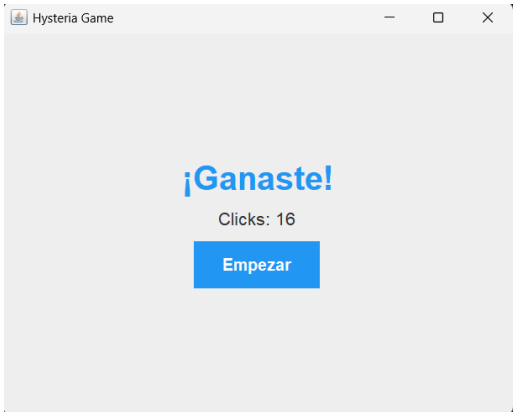
Funcionalidades opcionales

- **Conteo de clics:** Se incorporó un sistema para registrar la cantidad de turnos utilizados por el jugador. En la clase 'BoardModel', el método click() que incrementa un contador

cada vez que se realiza una acción sobre una celda. Este valor se recupera mediante getClicks() y se muestra al usuario en la vista 'WinView' al completar el juego.

- **Niveles de dificultad:** Se añadió soporte para diferentes tamaños de grilla, permitiendo ajustar la dificultad del juego. La clase GameConfig define tres tamaños preconfigurados en un enum llamado Level declarado en otra clase para mejorar e desacopamiento del programa: EASY: 3x3, MEDIUM: 5x5, HARD: 7x7
  El método initBoard de 'BoardModel' acepta un parámetro gridCells que ajusta dinámicamente el tamaño de la grilla según el nivel seleccionado por el usuario en 'LevelView'

Vistas



Ubicación del programa

- https://github.com/AgusNico-java/histeria