Universidad Nacional de General Sarmiento

Programación III
Trabajo Práctico N° 1

Integrantes:

- Menegol, Tomás.
- Mendoza, Leonardo.
- Cristian Brizuela.
- Sebastian Vivas.

Cursada:

- 2do Semestre 2025.

Comisión:

- 1, Turno noche.

Fecha de entrega:

- Martes 15/09/2025.

Introducción:

El presente documento tiene como objetivo plasmar en escrito los procesos, desarrollos y decisiones del primer trabajo práctico de la materia Programación 3, correspondiente al segundo cuatrimestre.

El proyecto consta de la implementación de un juego “Nonograma”. El mismo se juega sobre una cuadrícula en blanco y negro. Cada casilla de la cuadrícula puede estar pintada de negro, marcada con una X roja, o dejada en blanco (color default). Las pistas numéricas, ubicadas a la izquierda de cada fila y en la parte superior de cada columna, indican la cantidad de casillas que deben ser marcadas de negro por cada fila y columna para resolver el juego correctamente.

Herramientas:

El trabajo se desarrolló en lenguaje Java versión 17, utilizando Swing y la herramienta WindowBuilder, con el IDE Eclipse.

El codigo se organiza bajo la arquitectura “forms and controls” enseñada en clase, donde se busca principalmente respetar el principio de “separated presentation”. Parte de este principio implica que en ningún momento el código de negocio llame al código de la interfaz. Esto incluye que el código de negocio no conozca ni la identidad o las tecnologías utilizadas por la interfaz.

El proyecto consta de tres paquetes, entre ellos: un paquete para la interfaz, otro para la lógica de negocio y un tercero para el main que dispara el juego. Procedemos a listar el funcionamiento de los mismos y las clases correspondientes.

Problemáticas encontradas:

El primer desafío a enfrentar fue la división del trabajo. En este sentido, decidimos partir lo solicitado por el trabajo en problemas más pequeños y afrontarlos por partes. Nos dividimos en 2 equipos, uno encargado de la lógica, y otro de la interfaz, decidiendo que como probablemente el equipo de la interfaz finalice primero con su trabajo, todos nos reuniriamos para continuar con lo que quede de la lógica de negocio.

Una vez se decidió esto, cada equipo armó los requerimientos necesarios para representar su parte del problema (una especie de mini TAD). Dado que todavía no utilizamos TDD, previo a escribir las primeras líneas de código, decidimos probar algunos de los algoritmos a implementar (en especial los más complejos) en papel, para posterior poder directamente copiarlos sobre las clases requeridas. Esto sucedió por ejemplo con la función "acumularCasillasNegrasPorFila".

También se aprovecharon las clases de consulta para resolver dudas más propias a la implementación del juego. Por ejemplo, a fines de la lógica de negocio, para que un juego este resuelto de manera satisfactoria, lo unico que importa es que la casilla este marcada de color negro. Entonces decidimos que esto es lo único que íbamos a corregir en dicha instancia (esto NO quiere decir que la lógica de negocio no va a contemplar los otros 2 estados de la grilla: Blanco y Marcado).

Otra duda relacionada se originó en cuanto a cómo representar los 3 posibles estados de cada casilla. Primero se discute utilizar la herramienta Enum, pero posterior se decidió utilizar un sistema más simple con ints, donde:

- 0: Casilla blanca.
- 1: Casilla negra.
- 2: Casilla marcada.

Está decisión se tomó dado a que dejar las celdas negras con el valor "1", nos ayudaba para sumar rápidamente cuántas celdas negras consecutivas hay en la grilla (siendo estas una suma de 1 por cada celda consecutiva).

La problemática final y más grande ronda en cuanto a cómo corregir un nonograma resuelto por el usuario. Esto se da debido a que un nonograma podría tener más de una respuesta correcta, en especial al escalar el tamaño de la grilla. Después de unos días de deliberación, llegamos a la siguiente solución. El paso a paso del juego va a ser el siguiente:

1. Se generará una matriz cuadrada, vacía.
2. Se rellenará la grilla con celdas negras a través de algún algoritmo. Esto finalizara nuestra grilla "resuelta".
3. Otro algoritmo se encargará de recorrer la grilla resuelta por fila y por columna, contando las celdas negras, para generar las referencias que utilizara el usuario para resolver el juego.
4. Se generará otra matriz vacía, que será en la que el usuario irá ingresando el estado de cada casilla.

Dado que de todos modos tenemos que armar un algoritmo que me genera referencias para una grilla con celdas negras, podemos correr nuevamente este algoritmo sobre la grilla que completara el usuario. De está manera, si las referencias de ambas grillas coinciden, no importa cual sea el contenido en sí de cada grilla (porque pueden haber múltiples soluciones correctas), sabemos que la generada por el usuario es una solución válida.

Paquete Lógica:

Este paquete contendrá toda la lógica de negocio. Empezamos por declarar una clase abstracta "Tabla". Está tendrá los métodos comunes a todos los tableros, es decir, getters, setters y el método encargado de generar las referencias en base a una grilla. Adicional, una tabla bajo nuestra definición está conformada por una grilla y sus referencias (referencias para las columnas y para las filas) con lo cual tendrá estas 3 variables de instancia.

La clase "TableroRespuesta" será una subclase que se generará con celdas negras de manera aleatoria, para posterior generar referencias y que el usuario pueda utilizarlas para jugar.

Por otro lado la clase "TableroUsuario" será otra subclase, con una grilla vacía, la cual el usuario irá rellenando casilla por casilla en el juego. Al final del mismo, en base al estado de la grilla se generarán las referencias, las cuáles se corregirán contra las de TableroRespuesta.

Por último la clase "Juego" será la encargada de tomar las solicitudes brindadas por una interfaz y dictar el ritmo del juego, es decir, la generación de las grillas, la comparación que indica si el juego fue resuelto satisfactoriamente, etcétera.

Paquete Interfaz:

El paquete posee las clases responsables de las visualizaciones del usuario. Se comienza por la clase “mainFormFirst” la cual hace referencia al primer menú o pantalla que visualiza al usuario al ejecutar el juego, se observa la opción de brindarle al jugador la posibilidad de elegir entre tamaños de tableros, sumado a un botón “cómo jugar” el cual explica el juego en cuestión. Además, se establece el “look & feel”, en este caso “FlatDarkLaf”, a su vez, está presenta la navegación hacia la siguiente clase (pantalla) que el usuario podrá visualizar al pulsar el botón para jugar.

Luego nos encontramos con las clases de las distintas vistas, entre ellas tenemos vistaDeComoJugar, un panel de ayuda que explica al jugador en formato de texto las reglas, contiene distintos componentes como JTextArea no editable, su fin es informar y facilitar en las partidas. Posteriormente, vistaJuego es la pantalla donde el usuario hace uso del juego, muestra el tablero, las pistas ubicadas horizontalmente y verticalmente, y botones como limpiar el tablero y verificar. El usuario puede interactuar con su mouse, con el clic izquierdo pintar celda negra, clic derecho colocar una “X”, el método ajustarFuenteCeldas permite recalcular la fuente al cambiar el tamaño de la grilla, de forma que el usuario tenga legibilidad sobre los componentes al cambiar dimensiones. Siguiente a esta, tenemos vistaJugar que nos permite algunas configuraciones como por ejemplo configurar boton de limpiar tablero y verificar resultado.

De manera siguiente, la clase Result tiene la responsabilidad de mostrar al usuario un cuadro de diálogo al finalizar su partida, tiene un rol informativo, por otro lado si el usuario resuelve correctamente el Nonograma se cuenta con la clase vistaGanar, la cual en su diseño muestra un mensaje informando al jugador que ganó y se muestra al completar correctamente el tablero, sin embargo, en caso de ir por el camino incorrecto, se cuenta con la clase vistaPerder que de manera similar informa al jugador su derrota, y muestra dos tableros, a la izquierda la solución del usuario y a la derecha el tablero correcto del Nonograma. Se pensó que es importante brindarle al usuario los dos tableros para ayudarlo a comprender en qué se equivocó, y pueda comparar entre lo que eligió y lo que era la respuesta correcta.

Por último pero no menos importante, la clase Frame nos permite gestionar las pantallas que se describieron recientemente, controla las vistas a través de un registro de paneles, mediante mostrarVista() que busca el panel asociado a la clave pasada como parámetro (la cual puede ser las distintas pantallas: menu, partida, victoria y derrota) y la muestra, y le da la posibilidad de alternar entre ellas según las acciones del usuario.