TRABAJO PRACTICO

ROBOT

UNIVERSIDAD

- Universidad Nacional General Sarmiento
- Materia: Programación 3
- Comisión: 01

ALUMNOS

- Mayra Rossi (35.366.464) Maya.rossi03@gmail.com
- Melina Scabini (44.756.058) Melina.Scabini@gmail.com

PROFESORES

- Patricia Bagnes
- Ignacio Sotelo

Introducción

El objetivo de este trabajo práctico fue desarrollar una aplicación visual que permita simular el recorrido de un robot a través de una planta energética representada como una grilla rectangular de n x m. Cada celda de la grilla contiene una carga eléctrica que puede ser positiva (+1) o negativa (−1).

El robot debe partir desde la celda superior izquierda (0,0) y llegar a la celda inferior derecha (n−1, m−1), moviéndose únicamente hacia la derecha o hacia abajo. Como restricción principal, el recorrido debe tener una suma total de cargas igual a cero, es decir, debe tener la misma cantidad de celdas con carga positiva que negativa. Dado que solo se permiten caminos mínimos (de n + m − 1 pasos), esta condición sólo puede cumplirse si el total de pasos es par.

La implementación se enfocó en una búsqueda por fuerza bruta de todos los caminos posibles, incorporando estrategias de poda inteligentes para reducir la complejidad y mejorar los tiempos de ejecución.

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 trabajar con distintas grillas, que pueden provenir de archivos json o ser generadas aleatoriamente.

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.

Buenas prácticas

- **Funciones auxiliares legibles:** En nuestro código aplicamos funciones auxiliares para hacer más legibles varias condiciones que antes eran difíciles de entender a simple vista y a causa de esto colocábamos comentarios. Por ejemplo, la condición `Math.abs(sum) > stepsLeft`, que verifica si es imposible alcanzar un balance de cero aun eligiendo los mejores valores posibles, fue reemplazada por `isBalanceImpossible(sum, stepsLeft)`. También hicimos algo similar con la validación de condiciones como `x == lastRow && y == lastColumn` fueron reemplazadas por la función `isAtDestination(x, y)`, que describe explícitamente su propósito: comprobar si se llegó al destino. Lo mismo hicimos con los límites de movimiento: en lugar de `y < lastColumn y x < lastRow`, usamos `canMoveRight(y)` y `canMoveDown(x)`.
  Esto está en línea con los principios de Clean Code, que promueven escribir condicionales que se lean como frases naturales y autoexplicativas.

- **Pruebas Unitarias:** Se implementaron pruebas con JUnit para las clases críticas de negocio (JsonReader, Grid, Robot) Siendo útil para detectar puntos de quiebre cubriendo casos como archivo no existente, grilla vacía, grilla con cantidad impar de pasos, si se ejecutaron bien los algoritmos de poda y fuerza bruta, entre otros ejemplos. Esto nos ayudó a asegurar la robustez, dando lugar a la detección y manejo temprano de errores.

- **Modularidad y Organización del Código:** El proyecto está estructurado en paquetes bien definidos como model, view, controller, observer, lo cual refleja un enfoque modular y respetuoso con el principio de separación de responsabilidades. Cada paquete tiene una tarea clara y se encarga de una parte del sistema, lo que facilita la mantenibilidad y la escabilidad del proyecto a medida que crece.

- **Nombres claros:** Se emplean nombres descriptivos para clases y métodos, haciendo que el código sea autoexplicativo y fácil de mantener. Esta práctica, alineada con los principios 3 de Clean Code, reduce la necesidad de comentarios innecesarios y mejora la legibilidad general.

- **Encapsulado de Componentes para Reutilización:** Los elementos visuales, como botones, etiquetas, paneles, se encapsularon en funciones, lo que permitió su reutilización evitando repetir código y mantenimiento de forma independiente. Este enfoque también nos ayuda a mantener los archivos más ordenados, pequeños y comprensibles.

- **Métricas desacopladas del Robot**
  Tratando de seguir con el principio de responsabilidad única de *Clean Code*, movimos las métricas del algoritmo a una clase aparte llamada Metric. Así evitamos mezclar la lógica del recorrido con el control de estadísticas como tiempo.
  Esto hizo que la clase Robot quedara más limpia y enfocada solo en buscar caminos, mientras que Metric se encarga exclusivamente de medir el rendimiento. Además, usar

dos objetos (nonPruningMetric y pruningMetric) nos permitió comparar fácilmente el impacto de aplicar poda o no.

- **Desacoplamiento de Estilos con ColorPalette:** El uso del componente ColorPalette permite separar los aspectos visuales del comportamiento de la aplicación. Al centralizar los colores en un único lugar, se logra una mayor coherencia, se facilita la modificación de estilos y se refuerza un diseño desacoplado y escalable.

Decisiones de Desarrollo

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

- **Estrategias de poda:** Para evitar que el programa explore caminos innecesarios y así mejorar su rendimiento con menos llamados recursivos, pensamos en dos estrategias de poda. La más útil fue la **poda por balance inalcanzable**. Esta detiene el camino si la suma actual es tan alta (en positivo o negativo) que, aunque las celdas que quedan sean del signo contrario, no se puede llegar a una suma final de cero. Por ejemplo, si al robot le quedan 4 pasos y la suma actual es +6, aunque en cada paso reste −1, solo puede llegar a +2. Como no es cero, ese camino se descarta. Esto ayudó mucho a reducir las llamadas recursivas.
  También analizamos en aplicar una **poda por paridad**, que verifica si la suma actual y los pasos restantes tienen la misma paridad (par o impar). Si no la tienen, es imposible llegar a cero. Por ejemplo, si la suma parcial es +2 (par) y quedan 5 pasos (impar), nunca se puede llegar a 0. Sin embargo, como esta verificación ya se hace al principio cuando se carga la grilla (se descartan las que no tienen un camino de longitud par), vimos que no hacía falta repetir esta poda dentro de cada paso de la recursión.

- **Implementación del patrón Observer:** Se desarrolló un sistema de notificación basado en el patrón Observer que permite actualizar automáticamente la vista cuando cambia el estado del modelo. Esto desacopla la lógica de negocio de la presentación, permitiendo que los algoritmos se ejecuten independientemente de cómo se visualicen sus resultados, facilitando futuras extensiones y cambios en la interfaz sin afectar al modelo.

- **Cálculo del tiempo de ejecución:** Se empleó System.nanoTime() para medir con precisión los tiempos de ejecución de los algoritmos, permitiendo comparaciones exactas incluso en grafos pequeños donde las diferencias pueden ser del orden de microsegundos. Esta 4 implementación permite evaluar objetivamente la eficiencia de cada algoritmo en diferentes escenarios.

Funcionalidades opcionales

- **Generación de una grilla dinámica: I**mplementamos la sobrecarga del constructor de Grid para ofrecer dos formas de inicialización: una que permite recibir una grilla externa (pasamos una grilla que fue leída desde un archivo JSON), y otra en donde generamos dinámicamente una grilla aleatoria válida si no se pasan parámetros. Esto brinda mayor flexibilidad tanto para pruebas como para distintos escenarios de uso.

Video demo

Grabación de pantalla 2025-06-15 a la(s) 9.12.54 p. m..mov

Ubicación del programa

https://github.com/ScabiniMelina/RutaRobot