MEMORYMAZE APPLET

Kristel
Hernandez Reyes
3er Semestre

Proyecto final de la materia Estructura de Datos. Juego MemoryMazeApplet.java
Profesor: Adolfo Centeno Tellez

Introducción

El proyecto Memory Maze Applet es un juego interactivo desarrollado en Java que combina algoritmos de generación de laberintos, estructuras de datos y control de eventos dentro de un entorno visual basado en Applets.
El objetivo del jugador es mover una bolita azul a través de un laberinto hasta alcanzar la meta verde, utilizando las flechas del teclado.

Cada vez que se gana un nivel, el usuario puede presionar **"R"** para generar un nuevo mapa más grande y más difícil, logrando un avance progresivo.
Esta versión del proyecto genera laberintos reales mediante un algoritmo DFS, lo que crea caminos más complejos, naturales y siempre ganables.

1. Tecnología usada

Lenguaje y entorno

- **Lenguaje:** Java
- **Tecnología principal:** Applet
- **Framework gráfico:** AWT (Abstract Window Toolkit)

Librerías utilizadas

- **java.applet.Applet**
  Permite crear y ejecutar la aplicación dentro de un navegador como un Applet.
- **java.awt.***
  Utilizada para dibujar el laberinto, al jugador y la meta mediante figuras gráficas.
- **java.awt.event.***
  Maneja los eventos del teclado (KeyListener).
- **java.util.Random**
  Controla elementos aleatorios en la generación de laberintos.
- **java.util.Queue / LinkedList (opcional)**
  Pensadas para pruebas de validación de caminos; no se usan directamente en la versión final pero están incluidas para extensiones.

2. Descripción del funcionamiento

El juego se basa en un laberinto representado mediante una matriz bidimensional (int[][] maze):

- **0 → camino libre**
- **1 → pared**

El jugador inicia siempre en la esquina superior izquierda y la meta se coloca en la esquina opuesta.
Para moverse se usan las teclas:

- ↑ **Arriba**
- ↓ **Abajo**
- ← **Izquierda**
- → **Derecha**

Cada vez que el jugador alcanza la meta:

- Se muestra "**¡Ganaste!**"
- Al presionar **R**, se carga un nivel nuevo:
    - Aumenta el tamaño del laberinto
    - Se hace más complejo
    - Se reduce ligeramente el tamaño de cada celda para que quepa
- El jugador y la meta regresan a sus posiciones iniciales

Nuevas funciones agregadas

- Contador de movimientos realizados
- Laberintos garantizados como ganables
- Complejidad progresiva por niveles

3. Estructuras de datos utilizadas

- Matriz bidimensional (int[][] maze)

Es la estructura principal del proyecto.
Representa el laberinto en forma de grilla cuadrada.
Cada celda contiene:

- **0** = Se puede caminar
- **1** = Es una pared

-Arrays unidimensionales

- Para manejar movimientos y direcciones dentro del algoritmo DFS:

  int[][] dirs = { {1,0}, {-1,0}, {0,1}, {0,-1} };

-Variables primitivas

- playerX, playerY: posición del jugador
- exitX, exitY: posición de la meta
- size: tamaño dinámico del laberinto
- movimientos: contador de movimientos realizados

- Estructura de control

- Switch: para leer teclas presionadas
- Condiciones if: para validar los límites del laberinto
- Recursividad (DFS): genera el laberinto perfecto

- Árbol implícito por DFS

El algoritmo DFS genera caminos explorando celdas como si fueran nodos de un árbol.

Cada celda visitada es un "nodo hijo" del anterior, formando una estructura parecida a un árbol de expansión.

Esto permite:

- Un solo camino simple entre cualquier par de celdas
- Cero loops
- Maze perfecto garantizado

4. Diagramas

Diagrama de flujo general

1. Iniciar Applet
2. Generar laberinto con DFS

3. Dibujar mapa
4. Esperar teclas
5. ¿Movimiento válido?
    - Sí → mover jugador
6. ¿Jugador llegó a meta?
    - Sí → mostrar mensaje
7. ¿Tecla R presionada?
    - Sí → generar nuevo nivel
8. Repetir ciclo

Estructura general del código

MemoryMazeApplet
├── **init()** → Inicializa el applet
├── **paint(Graphics g)** → Dibuja el juego
├── **generarLaberintoGanable()** → Crea un laberinto válido
├── **generarMazeDFS()** → Algoritmo DFS del laberinto
├── **keyPressed(KeyEvent e)** → Movimiento y reinicio
├── **keyReleased() / keyTyped()** → Métodos vacíos requeridos por KeyListener

5. Incremento de dificultad por niveles

Cada nivel agrega complejidad mediante:

- Aumento del tamaño del laberinto

size += 2 por nivel.

- Reducción del tamaño de celdas

Permite que el mapa siga cabiendo en pantalla.

- Laberintos más largos y más ramificados

Porque el DFS se ejecuta en una matriz más grande.

- Mayor distancia entre entrada y salida

Hace que resolverlo requiera más movimientos.

El resultado es un sistema de niveles que aumenta la dificultad de forma natural y progresiva.

Conclusiones

El proyecto Memory Maze Applet demuestra cómo integrar estructuras de datos, algoritmos, recursividad y un entorno visual mediante Applets en Java.

Este trabajo logra:

- Generación de laberintos reales mediante **DFS**
- Representación del mapa con una matriz bidimensional
- Control de usuario por eventos del teclado
- Visualización mediante AWT
- Niveles progresivos y dinámicos
- Información adicional como movimientos e interfaz mejorada

Se aplicaron estructuras como:

- Arrays (matriz bidimensional)
- Arrays unidimensionales (direcciones del DFS)
- Árbol implícito en el algoritmo DFS
- Variables de estado y control

El proyecto culmina en un juego funcional, escalable y visualmente interactivo.

Bibliografía

- Oracle Java Documentation – The Java™ Platform, Standard Edition API Specification https://docs.oracle.com/javase/8/docs/api/
- Herbert Schildt, *Java: The Complete Reference, 9th Edition*, McGraw-Hill
- GeeksforGeeks – Java Applets and AWT Tutorial https://www.geeksforgeeks.org/awt-in-java/