Universidad Nacional Experimental Del Táchira
Vicerrectorado Académico
Decanato de Docencia
Departamento de Ingeniería Informática
Simulación de Sistemas

# INFORME “Juego de la Vida”

Hecho por:

Kevin Gerardo Zambrano Castro V-29.929.008

Axel Orlando Porras González V- 29.545.523

José Gregorio Briceño V- 29.544.700

San Cristóbal, agosto de 2025

Informe del proyecto “Juego de la Vida” en Java

## Introducción

El proyecto Juego de la Vida es una simulación incremental de aprendizaje basada en árboles de progreso. A lo largo de la iteración se diseñó un juego con tres árboles principales (“Main”, “Dino” y “Espacio”), cada uno de ellos con nodos generadores de recursos, modificadores y mejoras. El objetivo era mejorar la jugabilidad y la usabilidad del proyecto original, aportando un sistema de menús, la posibilidad de pausar y cargar partidas, una interfaz gráfica atractiva y un mecanismo de guardado y restauración del estado.

El trabajo se desarrolló a lo largo de varios mensajes con el usuario, donde se analizaron los requisitos, se recibieron archivos de código fuente y recursos (imágenes y sonidos) y se realizaron modificaciones extensivas. El código se mantuvo dentro del esquema de Java y se integraron nuevas clases para implementar los nuevos flujos de menús y la serialización de estado.

## Desarrollo

### Estructura del proyecto

El proyecto se encuentra en el directorio src, organizado en paquetes:

- Main: contiene App.java, punto de entrada que instanciará MotorJuego.
- controller: incluye MotorJuego.java que gestiona el loop del juego, listeners de menús, lógica de guardado/carga y la música. Se añadieron interfaces como MenuListener, NewGameListener, LoadGameListener y PauseListener para separar las responsabilidades de los eventos.
- view: agrupa todas las vistas Swing. Ventana.java es la ventana principal que ahora soporta dos modos: menús y juego (con un CardLayout). Clases como MenuInicio, FormNuevoJuego, MenuCargar y PanelPausa implementan los distintos menús. ArbolPanel representa cada árbol y dibuja nodos y aristas. Panel_Info_Generado y Panel_Info_Nodo muestran información de recursos y de nodos seleccionados.
- model: contiene las entidades del juego. Arbol.java crea nodos a partir de archivos .dt. Billetera.java gestiona tokens y generación por tick. Nodo, Generador, Modificador_Click y Modificador_Nodo implementan las distintas clases de nodos.
- Misc: SaveManager.java implementa la persistencia de partidas (almacenando archivos JSON con el estado completo) y Utils.java con utilidades de formato.
- img y audio: contienen los recursos gráficos y sonoros.

**Mejoras y nuevas clases**

El trabajo principal fue incorporar un sistema de menús completo y un sistema de guardado y carga. Para ello se añadieron o modificaron varias clases:

1. Ventana.java:
    - Ahora usa CardLayout para alternar entre menús (MenuInicio, FormNuevoJuego, MenuCargar) y el modo juego.
    - Añadió un botón Pausa que sólo aparece en el modo juego. Este botón abre PanelPausa con opciones para continuar, guardar partida y volver al menú.
    - Se implementó un removePauseButton() para eliminar el botón de pausa cuando se entra en un menú, evitando que se quede visible en otras pantallas.
    - Los botones Mapa 1/2/3 se rediseñaron como RoundButton (botón redondeado) y se reposicionaron en la esquina superior derecha.
2. MenuInicio.java, MenuCargar.java y FormNuevoJuego.java:
    - Se crearon menús específicos con botones redondeados y paneles de fondo translúcido. Por ejemplo, FormNuevoJuego ahora incluye campos de texto para nombre de usuario y de partida y botones ← Volver e Iniciar Juego con el nuevo estilo.
    - MenuCargar muestra la lista de partidas guardadas con un JList estilizado; los botones de cargar y volver usan el mismo diseño.
3. PanelPausa.java:
    - Diálogo modal que se abre con ESC o con el botón Pausa. Muestra opciones para continuar, guardar partida (sin pedir ID, usa el nombre de la partida actual) y salir al menú.
4. MotorJuego.java:
    - Se centralizó la inicialización de menús y se rediseñó la navegación. Al crear una partida o cargarla, se llaman métodos para preparar el juego (prepararJuego()), generar árboles, registrar tokens y poner la música del árbol actual.
    - Se implementó snapshotState() y applyState() para serializar y deserializar el estado del juego. El método snapshotState() captura índice de árbol activo, valor del click, cantidades de tokens y estado de cada nodo (activo y cantidad de generador). applyState() aplica esos valores sin descontar recursos y recalcula costos de nodos. La lógica evita modificar indebidamente la generación o los tokens al cargar.
    - Se actualizó el PauseListener para que el método onSave() guarde el estado del juego actual en un archivo JSON (Misc.SaveManager.save), usando como ID el nombre de la partida.
5. Generador.java y Billetera.java:
    - Se añadieron métodos setCant(int) y recalcCostoDesdeBase() en Generador para poder fijar la cantidad de unidades generadas al cargar una partida sin tener que comprar repetidamente.

recalcCostoDesdeBase() ajusta el costo en base al costo inicial y la cantidad de unidades (usando 1.15^cant).
    - Billetera incorporó métodos setCantidad y setGeneracion para fijar directamente valores de tokens y generación al cargar; además, se añadieron métodos toJSON() y fromJSON() (opcional) para posible extensión.
6. ArbolPanel.java:
    - Se agregó un método getNodoById para buscar nodos por su ID y así restaurar su estado a partir de los archivos JSON de guardado.

**Guardado y carga**

Al pulsar Guardar partida en el menú de pausa, MotorJuego.snapshotState() compone un JSON simple con el estado del juego y lo guarda a través de SaveManager. El formato de guardado no usa librerías externas. La carga lee el JSON y, mediante applyState(), reconstituye el juego: reinicia el índice de árbol activo, restaura tokens, restablece el valor del click y activa los nodos que correspondan, fijando cantidades de generadores sin descontar costos. Esto permite cargar una partida y continuar exactamente donde se dejó, lo cual se verificó al probar el juego (por ejemplo, el contador de tokens mantiene los valores y los nodos siguen activos). También se rediseñó MenuCargar para listar archivos de guardado y cargar el seleccionado.

**Diseño gráfico y experiencia de usuario**

- Se renovó la interfaz con botones redondeados y fondos semi-transparentes para los menús y paneles de información. Los colores se basan en tonos verde menta y azul marino, manteniendo coherencia visual.
- Se escaló el logo de la pantalla de inicio (logo.png) y el fondo (background.png) a las dimensiones de la ventana.
- Los botones Mapa 1/2/3 y Pausa comparten el estilo visual de los menús, con bordes redondeados, borde negro y relleno gris claro.
- PanelPausa se rediseñó con botones redondeados y un contenedor oscuro translúcido con borde sutil, aportando un aspecto moderno.
- Panel_Info_Generado muestra los tokens generados por segundo y totales, con bordes redondeados y texto centrado.

**Compilación y ejecución**

El proyecto se compila con el JDK 21. Para asegurar que la ejecución se haga con la misma versión de Java, se debe usar el java.exe correspondiente al JDK 21 (por ejemplo, C:\Program Files\Eclipse Adoptium\jdk-21.x.x\bin\java.exe en Windows). El comando general es:

```
css
CopiarEditar
javac -encoding UTF-8 -d bin (Encuentra todos los .java en src y compílalos)
```

java -cp "bin;src" Main.App

En sistemas Unix el separador de classpath es :. Si al ejecutar sale un error UnsupportedClassVersionError, significa que se está usando un java diferente de javac; se debe ajustar el PATH para que apunte al JDK 21.

## Conclusión

El proyecto Juego de la Vida se transformó de una base inicial a un juego con una interfaz moderna, fluida y con funcionalidades de guardado y carga. A lo largo del desarrollo se trabajó en:

- Reorganización y mejora de la arquitectura, separando claramente controladores, vistas y modelos.
- Diseño de nuevos menús y elementos UI con bordes redondeados y tonos consistentes que mejoran la experiencia del usuario.
- Implementación de un sistema de persistencia, permitiendo reanudar partidas en cualquier momento, lo que enriquece la jugabilidad.
- Manejo de música y sonidos dependiendo del menú o árbol en el que se esté jugando.
- Corrección de errores y refactorización del código para asegurar que los nodos se restauren correctamente y que no haya errores de versiones de Java.

Como resultado, el juego no sólo ofrece una simulación atractiva con árboles de progreso, sino que también brinda una experiencia profesional con menús intuitivos, guardado y carga, y una estética cuidada, cumpliendo los objetivos del usuario y demostrando un manejo detallado de una aplicación Swing de tamaño medio.