

“El Juego de la Vida by: LOS GALACTICOS”

Ramiro Escobar; Rodrigo Mariano Valdez; Franco Fliegler; Augusto Lautaro Ozuna Veron; Tomas Ahlbom.

Facultad Regional Resistencia – Universidad Tecnológica Nacional, ISI A, @rodri.go.valdez2806@gmail.com

Resumen

El presente trabajo describe la implementación del Juego de la Vida como ejercicio práctico dentro del estudio de Paradigmas de Programación. El objetivo fue construir una versión funcional del modelo de Conway y, al mismo tiempo, entender con más profundidad cómo se organiza un programa cuando cada parte cumple un rol específico dentro del sistema. La estructura del juego se basa en representar cada célula con un estado simple y en definir cómo cambia ese estado según la cantidad de vecinas activas. Durante el desarrollo se revisaron distintos detalles de la lógica, como la forma de recorrer el tablero y la actualización de las generaciones, ya que pequeños errores podían alterar por completo el comportamiento esperado. Gracias a estas revisiones fue posible obtener un funcionamiento estable y coherente con las reglas originales.

El resultado final permite observar patrones típicos del juego y analizar cómo reglas muy básicas pueden producir configuraciones complejas a medida que avanzan las generaciones.

1. Introducción

El trabajo se realizó con el objetivo de crear una versión funcional del Juego de la Vida y entender mejor cómo organizar un programa a partir de reglas simples. La idea fue construir un tablero donde las células pudieran activarse o apagarse según ciertas condiciones, y lograr que todo avance de una generación a la siguiente de manera ordenada.

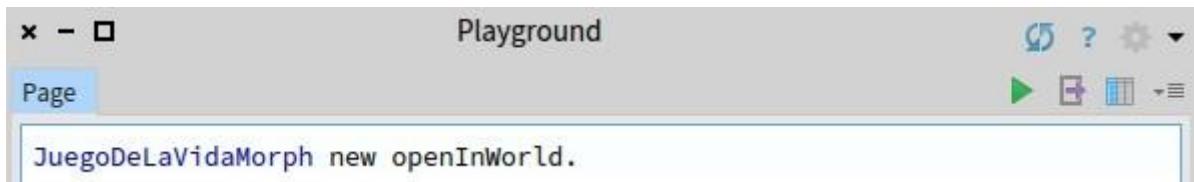
Para esto se definió cómo representar cada célula, cómo contar sus vecinas y qué debe ocurrir en cada actualización del tablero. A medida que el proyecto avanzaba, aparecieron detalles que fue necesario corregir, sobre todo en la parte donde cambia el estado de las células, ya que un pequeño error podía modificar todo el comportamiento del juego.

Esta introducción presenta el punto de partida del proyecto y el enfoque general del trabajo. Las partes siguientes explican cómo se armó el programa, qué problemas fueron surgiendo y qué decisiones permitieron obtener un funcionamiento estable.

2. Desarrollo

Abriendo el juego:

Al momento de iniciar el usuario deberá cargar el paquete en Pharo 7.0, con la ayuda del archivo JuegoDeLaVida.st. Luego deberá ejecutar la siguiente línea de código.



The screenshot shows the Pharo playground interface. The title bar says "Playground". The toolbar includes standard icons for file operations, help, and settings. The main text area contains the following code:

```
JuegoDeLaVidaMorph new openInWorld.
```

Fig 1. Código para abrir el juego.

El sistema le permitirá elegir el tamaño del tablero, teniendo como default 50, siendo el menor tamaño 30.

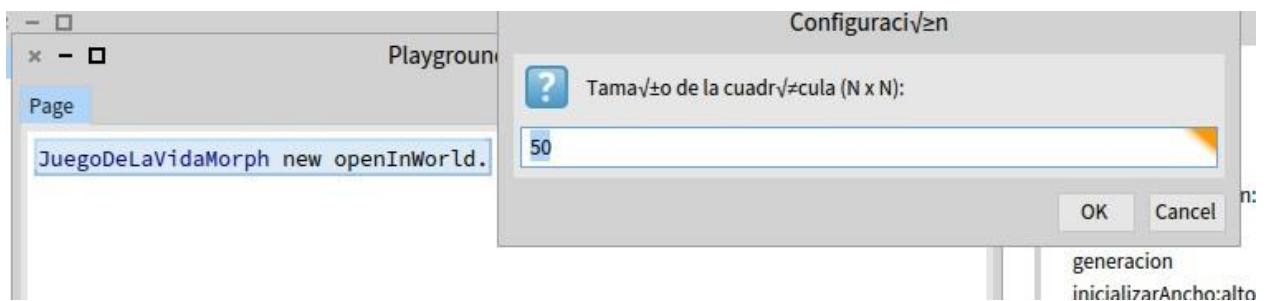


Fig 2. Tamaño de la grilla.

Automáticamente se abrirá el juego, permitiendo al usuario interactuar, pudiendo dibujar con el cursor las células haciendo click, Iniciar la simulación, Reiniciarla o Avanzar una generación.

La interfaz permite además visualizar la cantidad de células vivas y el orden de la generación actual.

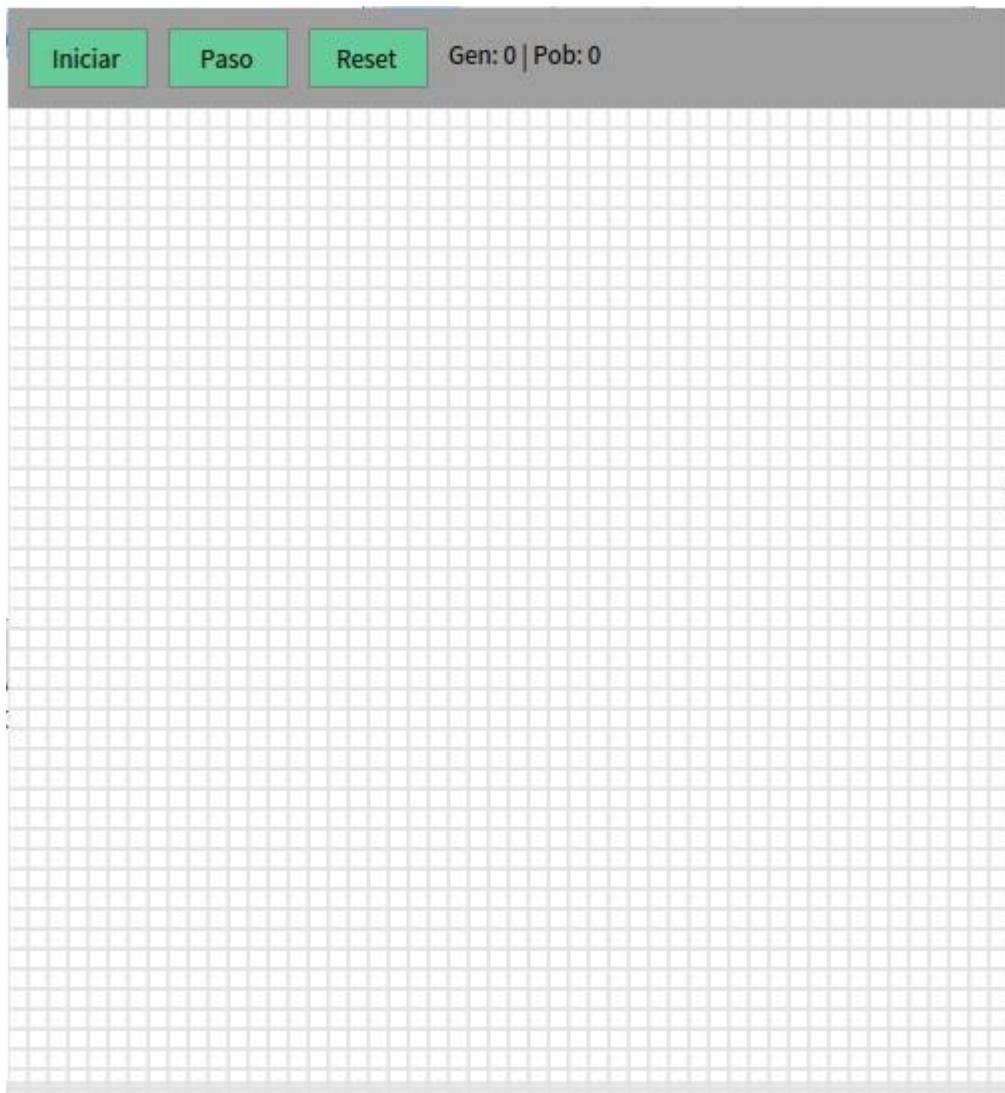


Fig 3. Interfaz del Juego Grilla 50x50.

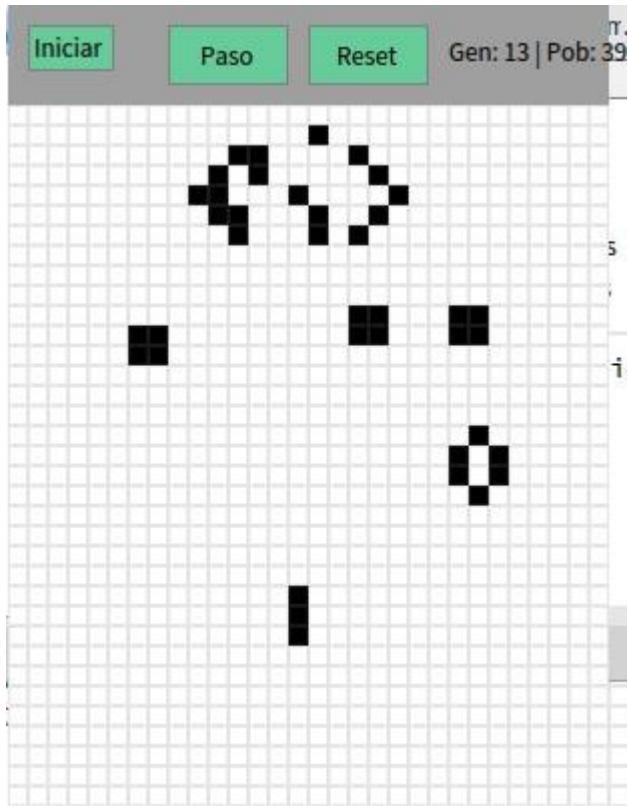


Fig 4. Ejecución del juego, Grilla 30x30.

Diseño del juego:

El desarrollo del juego se llevó a cabo en el entorno de programación Pharo, utilizando el lenguaje Smalltalk. El sistema se diseñó en tres clases principales: **Célula**, **Tablero**, y **JuegoDeLaVidaMorph**. Esta arquitectura de tres capas separa la lógica del modelo de la representación visual, garantizando la modularidad y la facilidad de mantenimiento.

1. Clase Celula

La clase **Celula** representa la unidad mínima del juego: una celda individual en la cuadrícula. Su principal responsabilidad es gestionar su propio estado de vida (viva o muerta) y determinar su destino en la próxima generación en base a las reglas del juego.

Métodos implementados en **Celula**:

Método	Categoría	Descripción
initialize	initialization	Inicializa la célula, estableciendo su estado inicial como muerta (viva := false).
estaViva	initialization	Retorna el valor booleano del atributo viva, indicando el estado actual de la célula.
revivir	EstadoCelula	Cambia el estado de la célula a viva (viva := true).

matar	EstadoCelula	Cambia el estado de la célula a muerta (viva := false).
alternar	EstadoCelula	Invierte el estado actual de la célula (si está viva, muere; si está muerta, revive) ⁷ . Se utiliza principalmente para la interacción del usuario (clics).
calcularProximoEstado : cantidadVecinos	EstadoCelula	Implementa las reglas del Juego de la Vida, retornando un booleano que indica si la célula debe vivir en la próxima generación. La regla de supervivencia se define como: (viva and: [cantidadVecinos between: 2 and: 3]) or: [viva not and: [cantidadVecinos = 3]].

Tabla 1. Métodos de la Celula.

2. Clase Tablero

La clase **Tablero** se encarga de contener la colección de instancias de Celula (la grilla), gestionar el flujo del juego y aplicar las reglas de transición de estado. Actúa como el modelo de datos del sistema, centralizando toda la lógica del juego.

Atributos de Objeto:

Atributo de Instancia	Descripción
grilla	Matriz (Array2D) que almacena todas las instancias de la clase Celula.
ancho, alto	Dimensiones de la cuadrícula de células.
generacion	Contador que registra el número de iteraciones o pasos de evolución completados.

Tabla 2. Atributos del Tablero.

Métodos implementados en **Tablero**:

Método	Categoría	Descripción
inicializarAncho: w alto: h	Juego	Inicializa las dimensiones (ancho y alto) de la cuadrícula con los valores proporcionados. Inmediatamente después, se llama a self reiniciar para construir la matriz de células.

reiniciar	Juego	Restablece el estado del tablero. Se crea una nueva matriz (Array2D) y se puebla con instancias de Celula completamente nuevas. El contador de la generacion se fija en cero.
evolucionar	Juego	Ejecuta un paso completo del Autómata Celular. Se utiliza una matriz temporal para guardar los nuevos Estados. Se calcula el próximo estado para cada célula llamando a calcularProximoEstado: y, una vez completado el cálculo
		global, se aplican los nuevos estados a la grilla y se incrementa el contador de generacion.
alternarEn: fila en: col	Juego	Permite cambiar el estado de una célula individual en la posición (fila, col). Se accede a la célula en esa coordenada y se le envía el mensaje alternar (revivir o matar).
contarVecinosEn: fila en: col	CalculoProximo Estado	Calcula el número de células vivas adyacentes a la célula en la posición dada. Se recorren las 8 posiciones circundantes, validando que las coordenadas estén dentro de los límites del ancho y alto del tablero.
ancho	Acceso	Retorna el valor actual de la dimensión horizontal del tablero.
alto	Acceso	Retorna el valor actual de la dimensión vertical del tablero.
generacion	Acceso	Retorna el número de veces que el tablero ha evolucionado, indicando el ciclo actual del juego.
poblacion	Acceso	Retorna el conteo total de células que se encuentran en estado vivo en la cuadrícula en ese momento. Se utiliza un bucle doble para iterar sobre todas las celdas y sumar las que cumplen con estaViva.
celulaEn: fila en: col	Acceso	Permite el acceso directo a la instancia de Celula que se encuentra en la posición especificada por (fila, col) dentro de la grilla.

Tabla 3. Métodos del Tablero.

3. Clase JuegoDeLaVidaMorph

La clase **JuegoDeLaVidaMorph** hereda de Morph y funciona como la capa de visualización (o Vista). Su responsabilidad principal es dibujar la cuadrícula, crear la interfaz de usuario (botones y paneles de información) y manejar la interacción del usuario (clics y simulación de tiempo), delegando toda la lógica del juego a la instancia de Tablero (modeloJuego)

Métodos implementados en **JuegoDeLaVidaMorph**:

Método	Categoría	Descripción
initialize	inicializacion	Método constructor principal. Llama a super initialize, establece el color de fondo y el layout. Instancia el modelo (Tablero) y se utilizan los métodos solicitarTamanoGrilla y configurarUI para iniciar el entorno.
solicitarTamanoGrilla	inicializacion	Se encarga de solicitar interactivamente al usuario las dimensiones (alto y ancho) deseadas para la cuadricula del juego. Una vez obtenidas, se utiliza esta información para inicializar el objeto Tablero (modeloJuego).
configurarUI	inicializacion	Define el diseño y la colocación de todos los elementos de la interfaz. Se crean, se posicionan y se añaden los morphs de botones (botonPausa, botonPaso, botonReset) y el panel de información (panelInfo).
actualizarInfoPanel	actualizacion	Se utiliza para refrescar el contenido del panel de información, mostrando los valores actuales de la Generación y la Población (células vivas), los cuales son consultados directamente al objeto modeloJuego.
drawOn: aCanvas	dibujo	Método sobreescrito de Morph que maneja el renderizado. Se itera sobre toda la grilla del modelo. Para cada célula, se dibuja un cuadrado en el aCanvas con el color correspondiente: negro si la célula está viva y blanco si está muerta.
mouseDown: evt	eventos	Captura el evento de clic del mouse del usuario. Se extraen las coordenadas de la posición del clic (evt position) y se delega la tarea a pintarCeldaEn: para que el estado de la célula sea alternado.
pintarCeldaEn: unPunto	interaccion	Traduce las coordenadas de píxeles (unPunto) a las coordenadas de fila y columna del Tablero. Posteriormente, se llama al método alternarEn:en: del modeloJuego para cambiar el estado de la célula seleccionada.
alternarSimulacion	simulacion	Invierte el estado del atributo booleano estaCorriendo. Si la simulación se activa, se llama a bucleSimulacion. También se encarga de actualizar el texto del botón de Pausa/Inicio.
bucleSimulacion	simulacion	Implementa el bucle principal de la simulación. Mientras estaCorriendo sea verdadero, se ejecuta avanzarSimulacion y se programa una nueva llamada a sí mismo después de un intervalo de tiempo (100 ms), manteniendo la simulación continua.

avanzarSimulacion	simulacion	Ejecuta un único paso de la simulación. Se envía el mensaje evolucionar al modeloJuego para que calcule la próxima generación, y luego se llama a self changed para forzar a la vista a redibujarse.
borrarGrilla	control	Detiene la simulación (si está corriendo) y llama al método reiniciar del modeloJuego, restableciendo el tablero a un estado completamente muerto y reiniciando el contador de generación.
botonPausa	ui creation	Método de apoyo que crea y configura el morph del botón para iniciar/pausar la simulación, asignándole el mensaje alternarSimulacion al ser presionado.
botonPaso	ui creation	Crea y configura el botón de Paso, el cual permite avanzar la simulación una sola generación a la vez, asignándole el mensaje avanzarSimulacion.
botonReset	ui creation	Crea y configura el botón de Reset (reiniciar), asignándole el mensaje borrarGrilla para restaurar el tablero a su estado inicial.
panelInfo	ui creation	Crea y configura el morph de texto que actúa como un panel de información, donde se muestra el estado de la generación y la población.

Tabla 4. Métodos de JuegoDeLaVidaMorph.

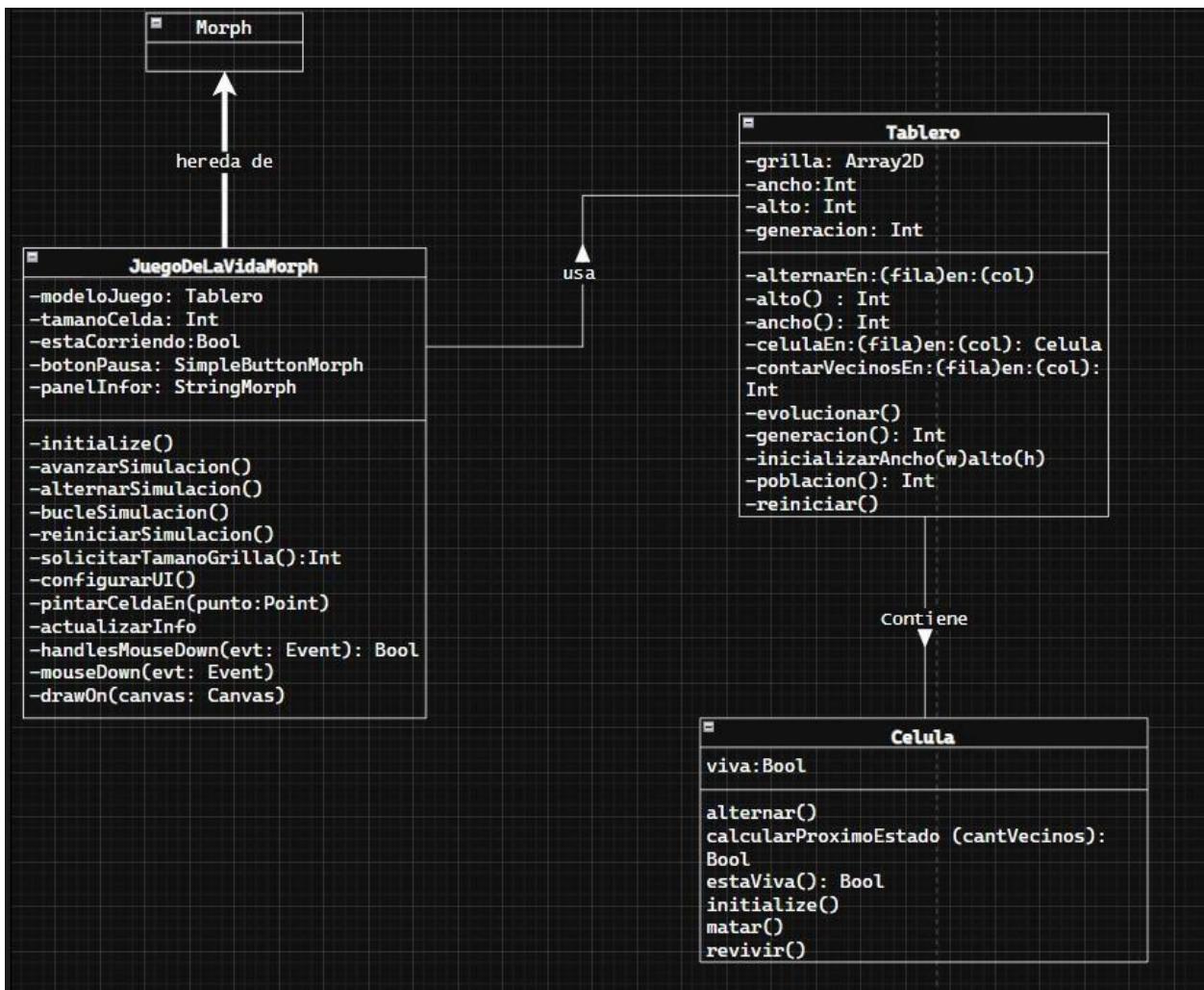


Fig 5. Diagrama de clases.

2.1 Implementación de la regla extra: HighLife

Para el agregado o regla extra del trabajo práctico, decidimos investigar variaciones del juego original.

Decidimos implementar la regla “HighLife”, descrita por Nathan Thompson, en la cual, a diferencia del juego original, esta variante permite que una célula muerta reviva si tiene exactamente 6 vecinos.

Esta modificación en el método “calcularProximoEstado” permite la aparición de una figura conocida como “El Replicador”, un patrón que se clona a sí mismo cada 12 generaciones.

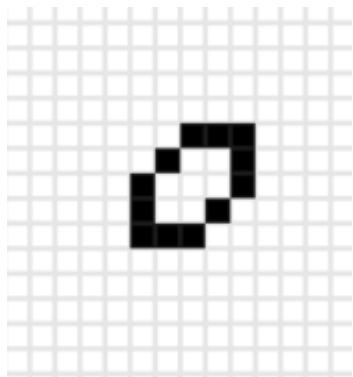


Fig 6. “El Duplicador”

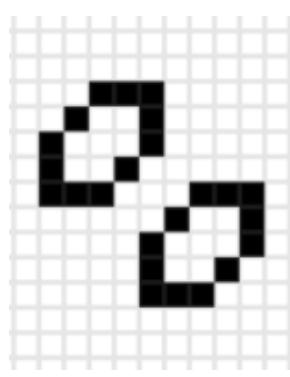


Fig 7. El Duplicador tras 12 generaciones

2.2 Implementación en Python

Para cumplir con el objetivo de análisis comparativo propuesto por la catedra, se replicó el escenario usando “Python” como lenguaje orientado a objetos alternativo.

Se utilizo la librería Pygame para la gestión gráfica y el ciclo de eventos, lo que representa una diferencia fundamental respecto al entorno de “Morph” que ya viene implementado en Pharo.

La estructura de clases se mantuvo idéntica para respetar el diseño original, pero la ejecución difiere: mientras que en Pharo el juego vive dentro de la imagen y se interactúa con mensajes en vivo, en Python se requiere de un ciclo infinito “while running” que escuche los eventos del teclado y mouse frame a frame.

Características	Smalltalk	Python
Entorno	Basado en Imagen	Basado en Archivos
Sintaxis	Mensajes (objeto mensaje)	Notacion de punto (Objeto.metodo())
Tipado	Dinámico	Dinámico
Gráficos	Morphic (Objetos Visuales nativos)	Pygame (Librería Externa)

Tabla 5. Comparación entre lenguajes utilizados

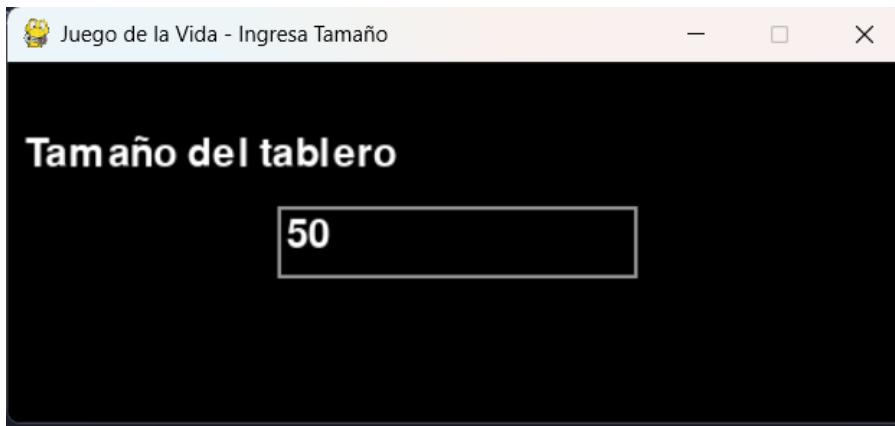


Fig 7. Configuración Tamaño de grilla del juego en Python

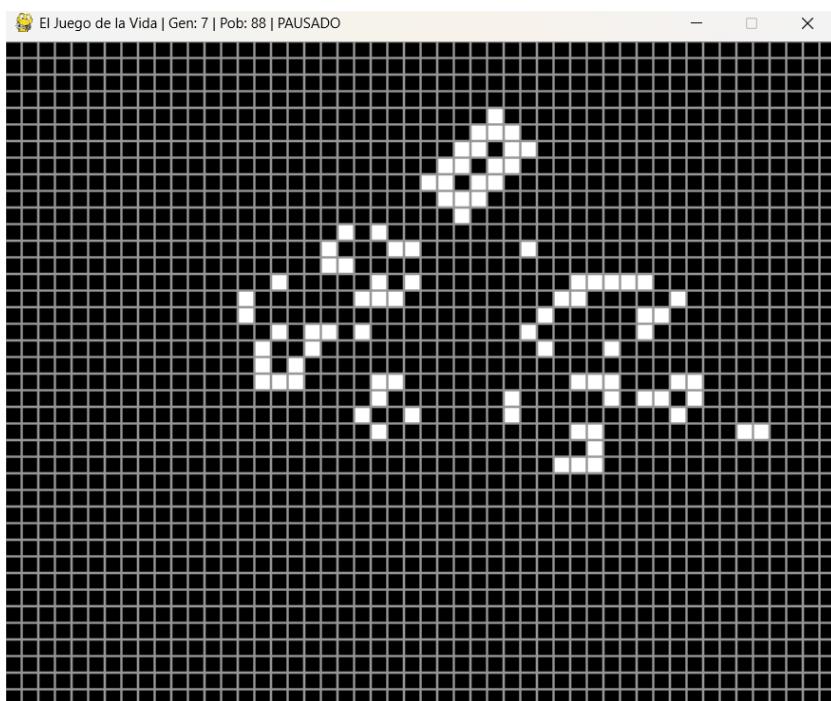


Fig 8. Ejecución del juego en su versión de Python, Grilla 50x50

3. Conclusiones

El desarrollo del Juego de la Vida de Conway en Smalltalk y el entorno Pharo fue una experiencia de aprendizaje fundamental, marcando nuestro primer encuentro real con la Programación Orientada a Objetos (POO). Lo que al inicio parecía un reto abrumador, se convirtió en una oportunidad para consolidar conceptos teóricos clave, como la herencia, el encapsulamiento y la delegación de responsabilidades.

Trabajar en Pharo nos obligó a pensar en términos de objetos y mensajes, una mentalidad muy diferente a la que estábamos acostumbrados. Descubrimos las ventajas de tener un sistema modular y cohesivo, donde cada clase, ya sea Celula, Tablero o JuegoDeLaVidaMorph, tenía una responsabilidad única y bien definida. Aprender a separar la lógica del juego (en Tablero) de la interfaz gráfica (en JuegoDeLaVidaMorph) fue el mayor aprendizaje en términos de diseño. Esta separación nos permitió modificar la parte visual sin alterar las reglas del juego.

El desafío más gratificante fue implementar la regla de evolución de las células. Ver cómo el método evolucionar en Tablero realizaba el cálculo de vecinos y la actualización de los estados en la matriz fue la prueba de fuego de que el modelo POO funcionaba correctamente.

Este proyecto no solo nos proporcionó una comprensión sólida de la POO, sino que también fortaleció nuestras habilidades para abordar problemas complejos mediante la descomposición en objetos simples.

Bibliografía

Conway, J. (1970). The Game of Life. *Scientific American*.

Google. (2025). Gemini. Modelo de lenguaje de gran escala (LLM) desarrollado por Google, utilizado para la asistencia en la redacción y la estructura del documento.

Pharo Project. (2024). Pharo Documentation. Recuperado de <https://pharo.org/documentation>

Wilensky, U. (1999). Cellular Automata: Concepts and Applications. The Center for Connected Learning and Computer Modeling, Northwestern University.