

# Implementación del Juego de la vida de Conway's en PowerDEVS

Lucio Mansilla - Brenda Dichiara

4 de agosto de 2023

## 1. Introducción / Problema

El Juego de la Vida de Conway's, comúnmente conocido como "Juego de la Vida", es un autómata celular que fue propuesto por el matemático británico John Horton Conway en 1970. Los autómatas celulares son modelos matemáticos para sistemas dinámicos que evolucionan en pasos de tiempo discretos (generaciones). A pesar de su simplicidad aparente, tienen la capacidad de simular sistemas complejos y mostrar comportamientos emergentes, siendo utilizados en diversos contextos, desde la física hasta la teoría de computación.

En el Juego de la Vida, cada celda que compone la cuadrícula bidimensional puede estar en uno de dos estados: *viva* o *muerta*. Las celdas interactúan con sus ocho vecinos adyacentes en horizontal, vertical y diagonal, transicionando entre los estados de vida y muerte de acuerdo a las siguientes reglas de evolución en su versión original:

- Cualquier celda viva con dos o tres vecinos vivos sobrevive para la siguiente generación.
- Cualquier celda viva con menos de dos vecinos vivos muere por soledad/aislamiento para la siguiente generación.
- Cualquier celda viva con más de tres vecinos vivos muere por sobrepoblación para la siguiente generación.
- Cualquier celda muerta con exactamente tres vecinos vivos nace para la siguiente generación.

Estas reglas originales son comúnmente denotadas como 23/3, representando las condiciones de supervivencia y nacimiento, respectivamente. Es decir, una celda viva sobrevive si tiene 2 o 3 vecinos vivos y una celda muerta nace si tiene exactamente 3 vecinos vivos.

Se han desarrollado una gran cantidad de variantes. En este sistema de reglas, la denominación "S/B" (Survive/Birth) representa las condiciones para la supervivencia y el nacimiento de las células. "S" son los números de vecinos que permiten a una celda viva sobrevivir, y "B" son los números de vecinos que permiten a una celda muerta nacer. Algunas variantes que podemos destacar:

- HighLife: 23/36. Similar al Juego de la Vida original, pero con la adición de que las células muertas con 6 vecinos vivos también nacen. Este juego es famoso por su patrón replicador", el primero que se descubrió y que puede replicarse a sí mismo.
- Seeds: 0/2. En esta variante, las células vivas no sobreviven y las células muertas nacen si tienen exactamente 2 vecinos vivos. Es conocido por su crecimiento muy rápido y la creación de estructuras complejas a partir de pequeñas semillas.

Es importante señalar que la mayoría de las  $2^{18}$  posibles reglas del espacio producen universos que son o bien demasiado caóticos, o bien demasiado desolados para ser de interés. Sin embargo, es posible observar la aparición de diversos patrones, algunos estáticos, otros oscilan entre varios estados y otros se desplazan por el tablero. Estos patrones serán el objeto de estudio en las secciones posteriores de este informe.

El propósito de este proyecto es explorar la dinámica del juego de una manera visual e interactiva. Para ello, se implementará el juego en PowerDEVS, una herramienta de simulación de eventos discretos basada en la teoría DEVS (Discrete Event System Specification).

## 2. Especificación DEVS de una celda

El DEVS que representa únicamente a una célula se define como sigue:

$$C = \langle X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle$$

donde

- $X = \text{GameState}$

La estructura de *GameState* se define formalmente como sigue:

- **board**: Un conjunto que representa el estado de cada célula en el tablero del juego. Cada celda en el tablero puede tener un valor de 0 (muerta) o 1 (viva).
- **rows**: El número total de filas en el tablero.
- **cols**: El número total de columnas en el tablero.
- **SR**: Representa las reglas de supervivencia.
- **BR**: Representa las reglas de nacimiento.

- $Y = \mathbb{N} \times \{0, 1\}$

- $S = \mathbb{N} \times \{0, 1\} \times \mathbb{R}_0^+$

El estado es una tupla  $(cid, ls, \sigma)$  donde

- $cid \in \mathbb{N}$  es el identificador de la célula.
- $ls \in \{0, 1\}$  es el estado de la célula (1 = viva, 0 = muerta)
- $\sigma \in \mathbb{R}_0^+$  es el tiempo restante para realizar una próxima salida.

- $\delta_{\text{int}}((cid, ls, \sigma)) = (cid, ls, \infty)$

- $\delta_{\text{ext}}((cid, ls, \sigma), e, (x, p)) = \begin{cases} (cid, 0, 1) & ls = 1 \wedge \text{alives} \notin x.SR \\ (cid, 1, 1) & ls = 0 \wedge \text{alives} \in x.BR \\ (cid, ls, 1) & \text{otherwise} \end{cases}$

donde  $\text{alives} = \text{countAlives}(x.rows, x.cols, cid, x.board)$

- $\lambda((cid, ls, \sigma)) = (cid, ls)$

- $ta((cid, ls, \sigma)) = \sigma$

Donde  $x.SR$  y  $x.BR$  representan las reglas de supervivencia y nacimiento respectivamente, las cuales se explican en detalle en la Sección 1: "Introducción / Problema".

- **countAlives:** Esta función es responsable de determinar la cantidad de células vecinas vivas para una célula específica. Se le proporcionan cuatro parámetros claves: el número de filas (rows) y columnas (cols) del tablero, el identificador único de la célula (cell\_id) y el estado actual del tablero (board). La función realiza tres pasos principales:
  - **Identificación:** Localiza la posición exacta de la célula dada en el tablero utilizando su identificador y las dimensiones del tablero.
  - **Exploración y Conteo:** Examina únicamente las células vecinas, excluyéndose a sí misma, y realiza un conteo de cuantas vecinas están vivas.
  - **Resultado:** Retorna la cantidad total de células vecinas vivas.

## 2.1. Mediator

Como podemos apreciar, la lógica de cada célula en DEVS es relativamente sencilla. Esto se debe, en gran medida, a la presencia de un patrón de diseño muy popular en la programación, conocido como *Mediator*. El Mediator permite desacoplar la lógica de las células, gestionar el tiempo de los eventos y distribuir la información del tablero. Es un patrón de diseño de comportamiento que se utiliza para reducir las comunicaciones complejas y las dependencias entre diferentes entidades.

En el contexto de DEVS, el *Mediator* se encarga de transmitir el estado inicial, así como cualquier cambio en el tablero a las células individuales. A su vez, las células informan al *Mediator* sobre cualquier cambio de estado. Este proceso permite que la información se distribuya de manera independiente entre todas las células.

Esta estrategia trae consigo una serie de ventajas significativas. En primer lugar, se simplifica el diseño de las células al no tener que lidiar con la lógica adicional para manejar los tiempos sincronizados de los eventos discretos (Por ejemplo, tener que primero actualizar y luego informar o viceversa). En segundo lugar, al eliminar la necesidad de que cada célula esté directamente conectada con todas las demás, se obtiene un sistema más flexible independiente y escalable, puesto que en un futuro solo basta con añadir una nueva célula y conectarla al *Mediator* para que esta pueda interactuar con el resto del sistema siempre que se obtenga un tablero de  $n$  filas y  $m$  columnas.

La especificación DEVS del *Mediator* se puede describir como:

$$M = \langle X, Y, S, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle$$

- $X = \mathbb{N} \times \{0, 1\}$
- $Y = \text{GameState}$
- $S = \text{GameState} \times \mathbb{R}_0^+$

definiendo  $\text{game} \in \text{GameState}$  entonces se tiene:

- $\delta_{\text{int}}(\text{game}, \sigma) = (\text{game}, \infty)$
- $\delta_{\text{ext}}(\text{game}, \sigma, (\text{cell\_id}, \text{life\_state})) = (\text{game.board}[\text{x.cell\_id}] \leftarrow \text{x.life\_state}, 1)$
- $\lambda(\text{game}) = \text{game}$
- $ta(\text{game}, \sigma) = \sigma$

## 2.2. Optimización

La optimización implementada en la especificación DEVS de la célula se basa en minimizar las comunicaciones innecesarias con el Mediador. Específicamente, evita enviar actualizaciones de estado al Mediador cuando el estado de una célula permanece sin cambios.

En una simulación con un tablero de 8x8 y un tiempo de ejecución de  $t = 10,000$ , la implementación de esta optimización redujo el tiempo de ejecución de 754 ms a 2 ms. Esta optimización puede tener un impacto aún mayor en tableros de juego más grandes y simulaciones que se ejecutan durante un mayor número de generaciones. Notablemente, es especialmente eficaz en patrones donde ciertas áreas del tablero permanecen estáticas durante largos períodos de tiempo. Por ejemplo, en simulaciones con el patrón "glider", se observaron mejoras de eficiencia similares.

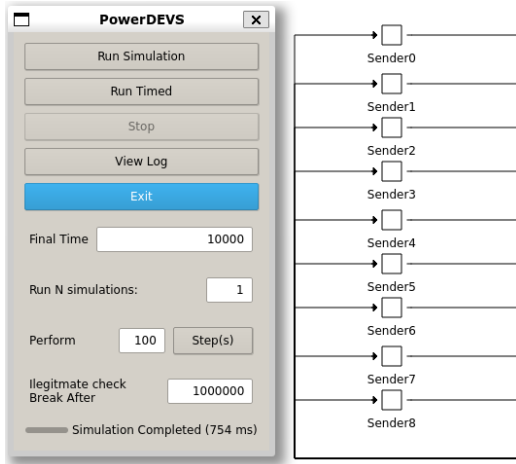


Figura 1: Simulación sin optimización.

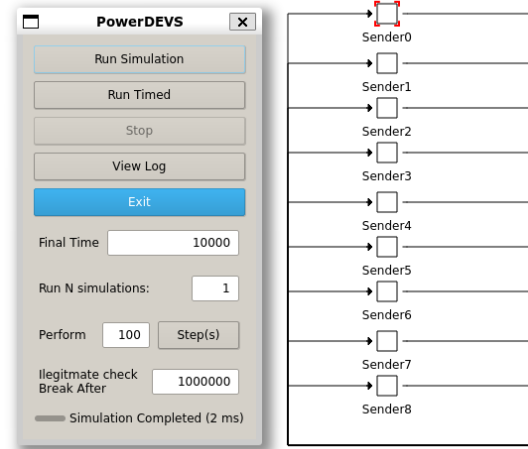


Figura 2: Simulación con optimización.

Esta estrategia aporta ventajas significativas. En primer lugar, reduce la cantidad de datos a procesar y transferir, lo que puede mejorar el rendimiento. En segundo lugar, al centrarse solo en los cambios de estado, se reduce la complejidad del sistema, facilitando su comprensión y depuración. Por último, hace que el sistema sea más escalable, ya que agregar más células no aumentará la cantidad de mensajes enviados si la mayoría de las células no cambian su estado.

## 3. Implementación en PowerDEVS

En PowerDEVS, el juego de la vida se implementa utilizando dos componentes clave: una entidad denominada 'célula' y un patrón de diseño denominado 'mediator'. Ambos son modelos atómicos y sus relaciones e interacciones definen el estado y el comportamiento del juego.

La célula se implementa como un modelo atómico simple con una única entrada y salida. Cada célula es una entidad única que cuenta con un identificador único que la representa en el tablero. Por otro lado, el Mediator es un modelo que juega un papel crucial para coordinar y gestionar las interacciones entre estas células.

El Mediator, también implementado como un modelo atómico, desempeña un papel central en la coordinación y el control de las interacciones entre las células. Este componente no solo gestiona las relaciones entre las células, sino que también es responsable de inicializarlas. El Mediator recibe como parámetro el nombre del archivo que contiene el estado inicial del tablero así como las reglas de supervivencia y nacimiento (1er parámetro). Además una de las funcionalidades importantes del Mediator es que permite definir el nombre de archivo de salida (2do parámetro), en el mismo se

registra el estado del tablero del juego en cada generación, facilitando su exportación y visualización en formatos como gif, mp4 o png.

En la Figura 3 a la izquierda se muestra la representación de una célula como un modelo atómico, y a la derecha, el Mediator.

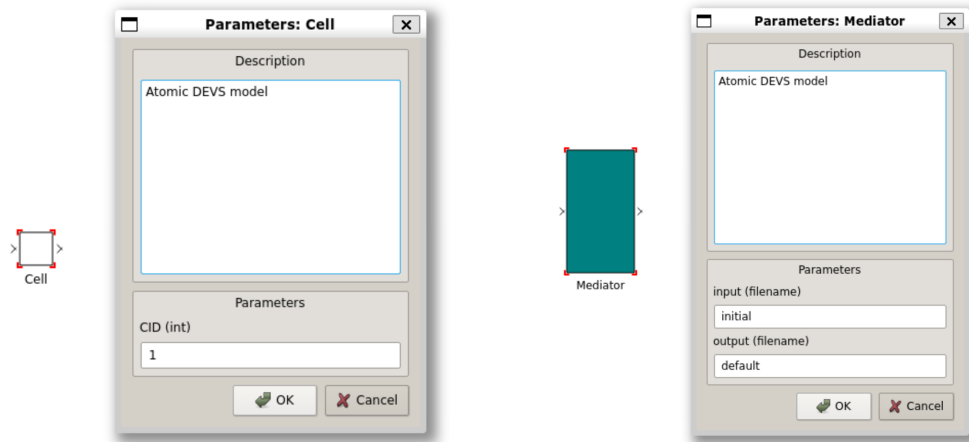


Figura 3: El Mediator y la Célula como modelos atómicos en PowerDEVS.

El trabajo conjunto de estas dos entidades da lugar a la dinámica del juego. A través del Mediator, el juego inicializa y coordina el estado de las células, gestionando su comportamiento a lo largo de cada generación. En la siguiente imagen, se proporcionará un ejemplo de un tablero de 3x3 inicializado por un Mediator, ilustrando cómo se interconectan múltiples células y cómo el Mediator permite que cada célula sea independiente de las demás, es decir, no se comunican directamente entre ellas.

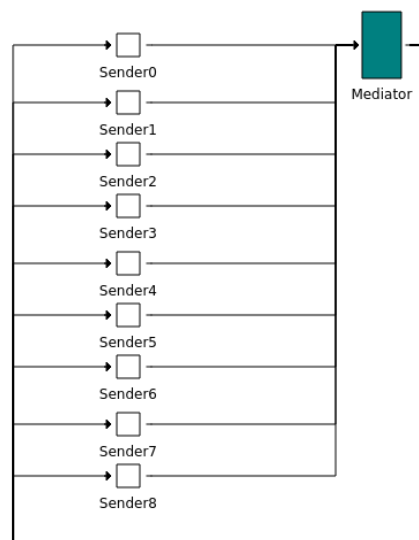


Figura 4: Tablero de ejemplo 3x3.

También se incorporó al proyecto una librería, en la misma se pueden encontrar los modelos

atómicos anteriormente mencionados en las 3 y 4, así como también varios modelos de tableros de ejemplo, como se puede observar en la figura 5.

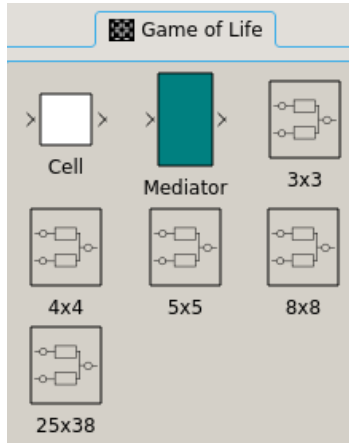


Figura 5: Librería

### 3.1. Automatización y Personalización de Tableros

Entendemos que la creación de tableros personalizados en PowerDEVS puede ser un proceso laborioso, especialmente para tableros de grandes dimensiones. Para facilitar este proceso, hemos desarrollado una herramienta de ayuda basada en una interfaz gráfica de usuario (GUI) implementada en Python.

Esta herramienta, que hemos llamado 'Settings GoL', permite generar tableros de dimensiones  $N \times M$  en formato PDM, los cuales pueden ser utilizados directamente en PowerDEVS. El proceso es sencillo e intuitivo, y reduce significativamente el tiempo y el esfuerzo requeridos para la configuración de tableros personalizados.

En la Figura 6, se muestra una vista previa de la interfaz de Settings GoL y de algunas de sus funcionalidades.

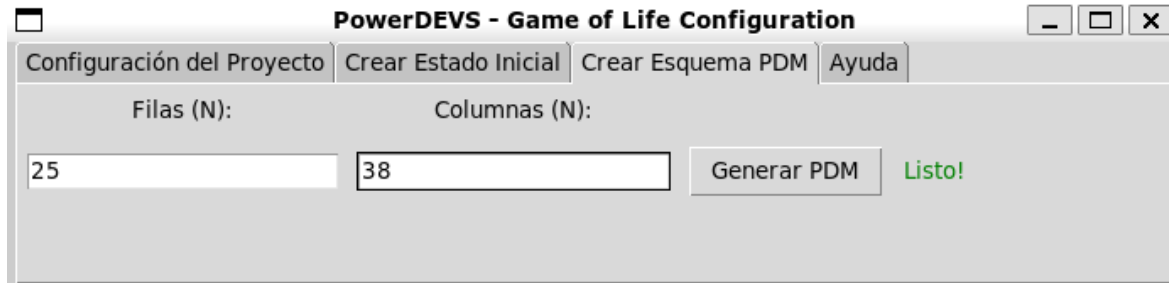


Figura 6: Vista previa de Settings GoL, pestaña generador de tableros PDM.

Para un entendimiento completo de cómo utilizar esta herramienta y sus otras funcionalidades se ha elaborado una guía detallada de uso que se encuentra en el anexo de este informe.

## 4. Patrones

El Juego de la Vida, concebido por el matemático británico John Horton Conway, es famoso por la vasta variedad de patrones que emergen de sus simples reglas, en particular, las reglas 23/3. Los

patrones se definen como configuraciones de células que se repiten tras un número determinado de generaciones, también conocido como su "periodo". Los patrones pueden ser estáticos, oscilantes o móviles, dependiendo de cómo cambian a lo largo del tiempo. En las secciones siguientes se presentarán ejemplos de estos patrones bajo las reglas 23/3.

#### 4.1. Patrones estáticos

Los patrones estáticos, también conocidos como "still lifes", son configuraciones de células que no cambian de una generación a la siguiente, es decir, su periodo es 1. A continuación, se presentan ejemplos de estos patrones:

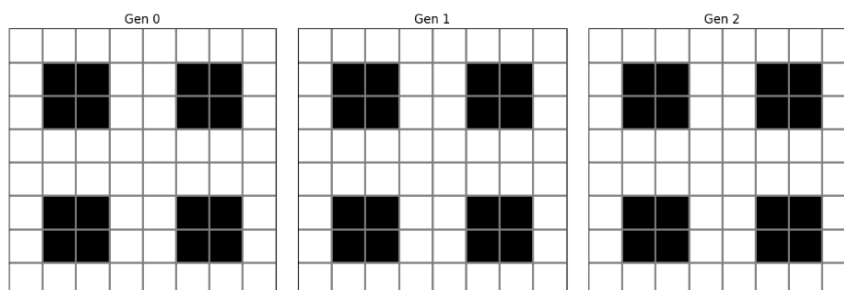


Figura 7: Patrón bloque: un patrón estático compuesto por un cuadrado de 2x2 células. Aparece frecuentemente debido a su simplicidad.

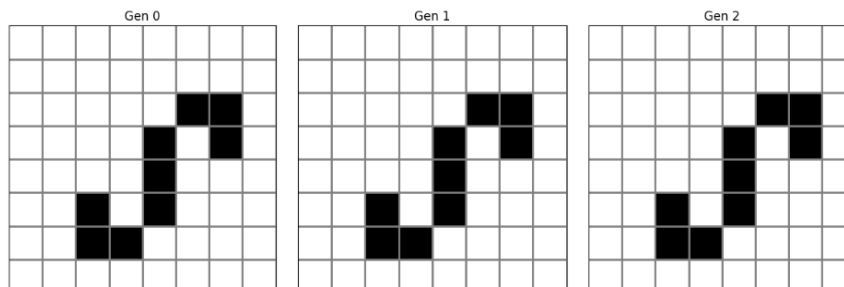


Figura 8: Patrón integral: este patrón estático se asemeja a la forma de un signo integral.

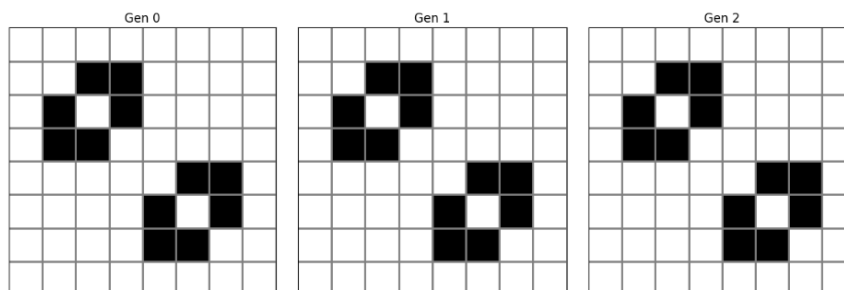


Figura 9: Patrón ship: este patrón estático se asemeja a la forma de una nave espacial.

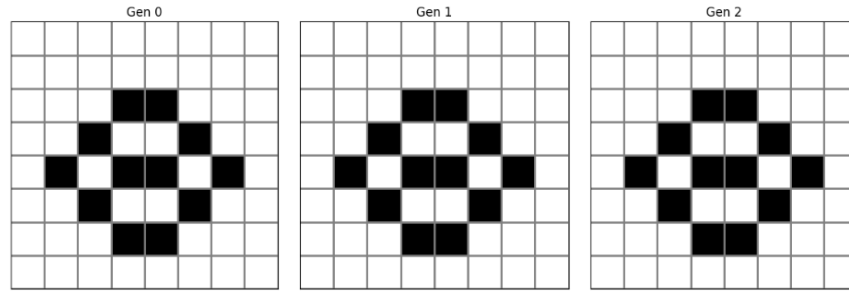


Figura 10: Patrón honeycomb: este patrón estático se asemeja a la estructura de un panal de abejas.

## 4.2. Osciladores

Los osciladores son patrones que vuelven a su configuración inicial después de un número fijo de generaciones, conocido como su período. A diferencia de los patrones estáticos, los osciladores cambian su apariencia durante su ciclo de vida, pero eventualmente vuelven a su estado original.

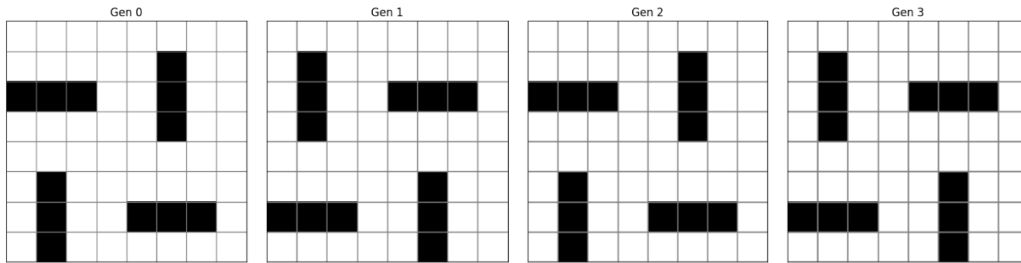


Figura 11: Patrón blinker: un oscilador de periodo 2. Oscila entre una orientación horizontal y vertical cada generación. Notar que en este caso hay 4 blinkers en el tablero.

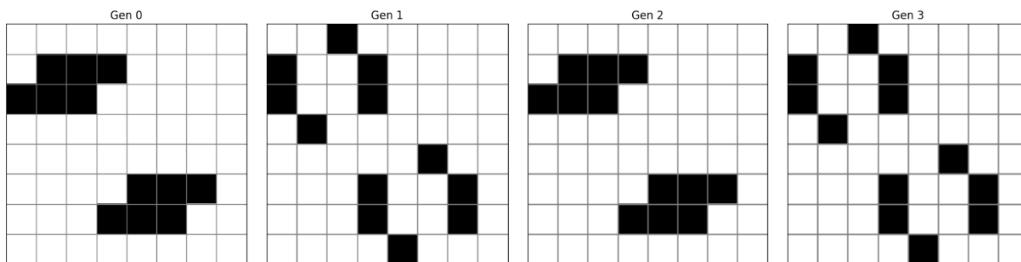


Figura 12: Patrón toad: un oscilador de periodo 2 que oscila entre dos configuraciones.



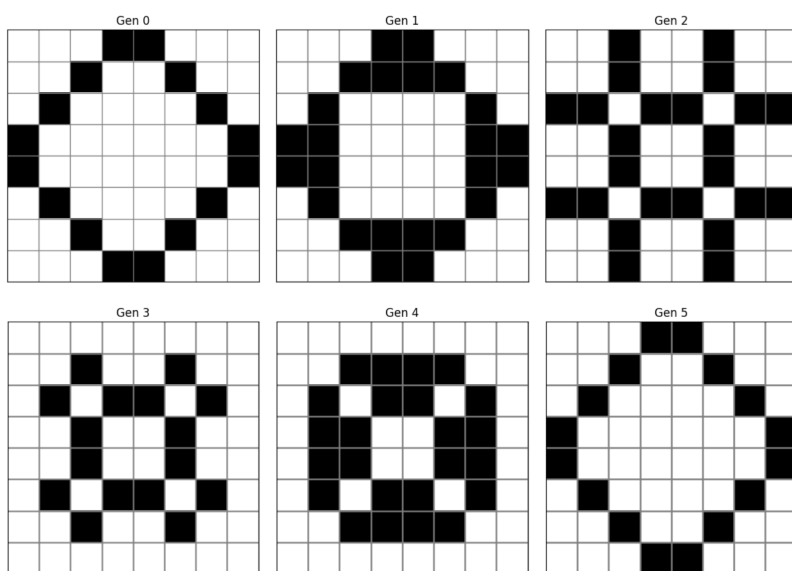


Figura 13: Patrón octagon: un oscilador de periodo 5 que se desplaza en un ciclo de rotación.

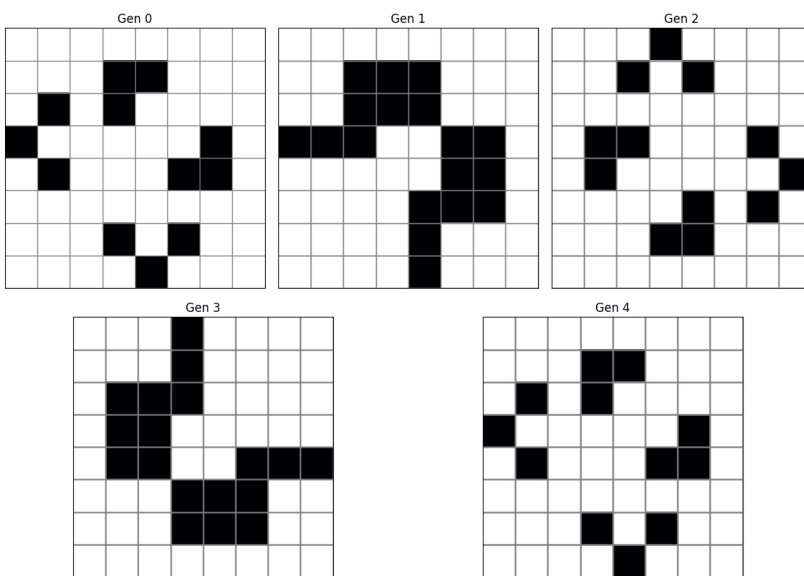


Figura 14: Patrón mazing: un oscilador de periodo 4 que oscila entre una forma cuadrada y una forma de cruz.

### 4.3. Naves espaciales

Las naves espaciales son patrones que se trasladan a través de la cuadrícula mientras oscilan. El ejemplo más conocido es la "nave ligera" (o "glider"<sup>en inglés</sup>), que se desplaza diagonalmente a través de la cuadrícula mientras oscila entre cuatro configuraciones diferentes.

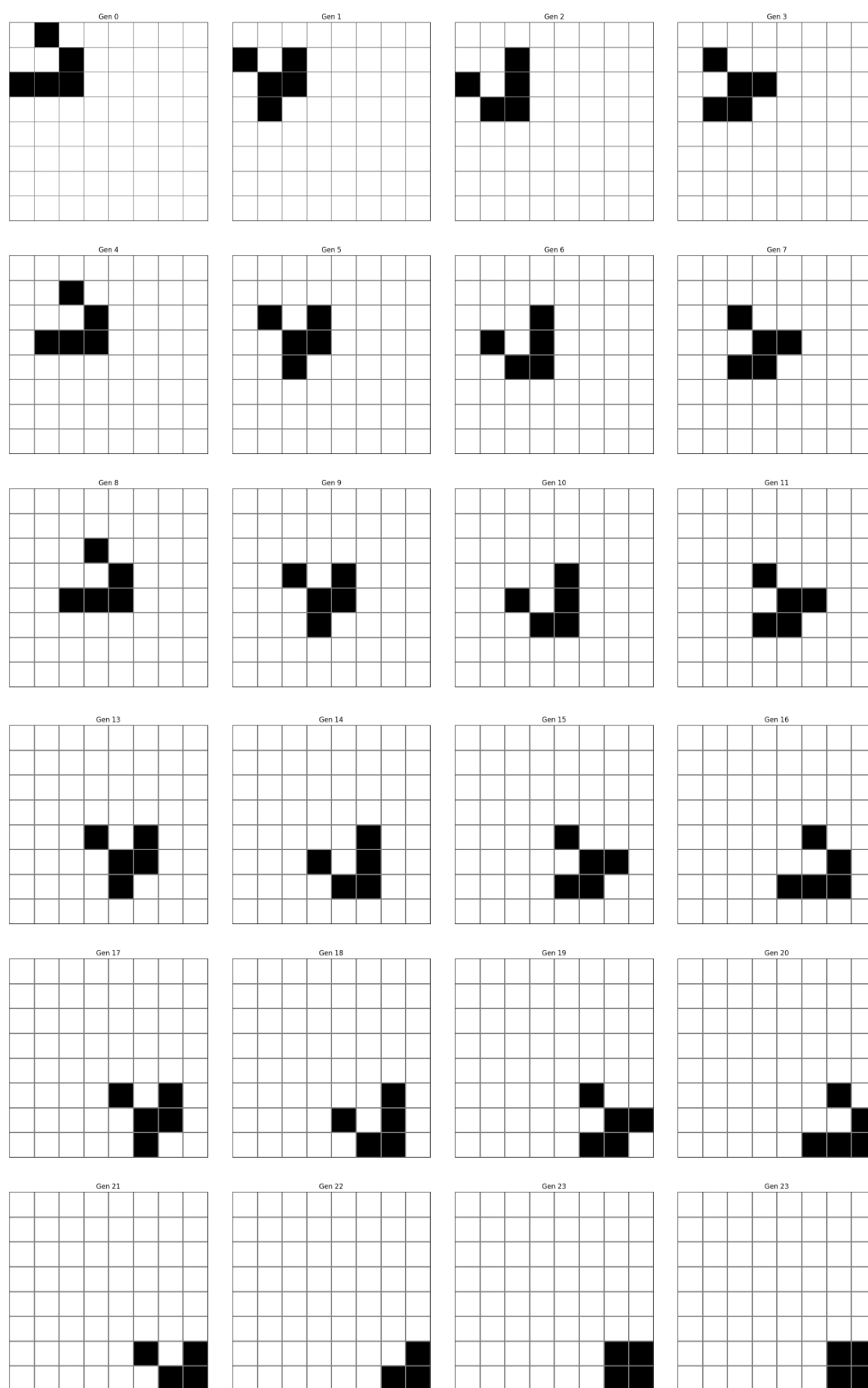


Figura 15: Patrón glider, generaciones 12-24: continúa su desplazamiento diagonal.

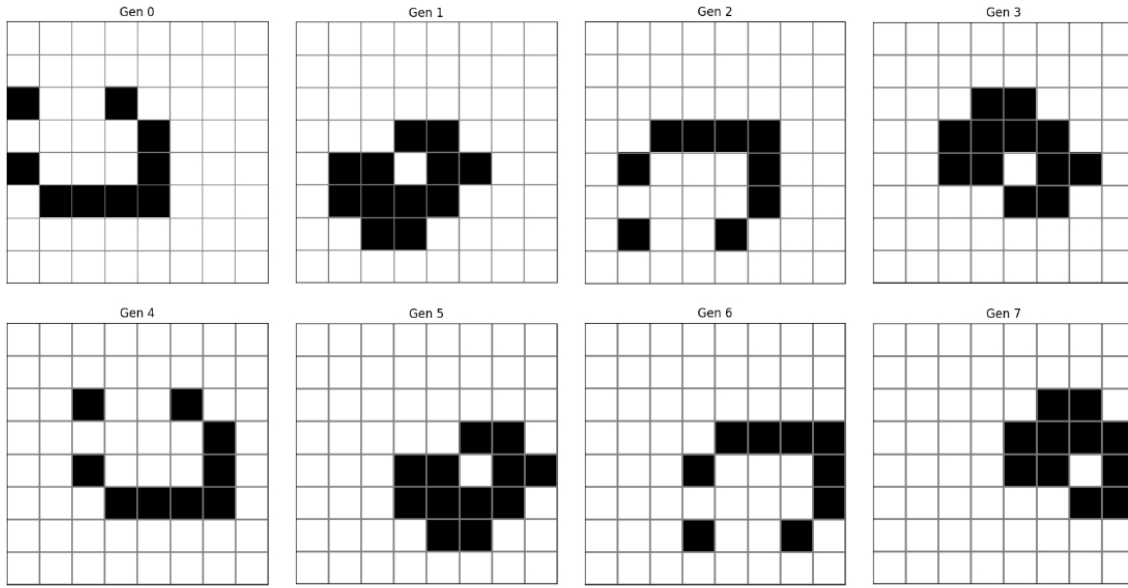


Figura 16: Patrón LWSS (Nave Ligera de Espacio Medio): una nave espacial que se desplaza horizontalmente mientras oscila en una secuencia de cuatro configuraciones.

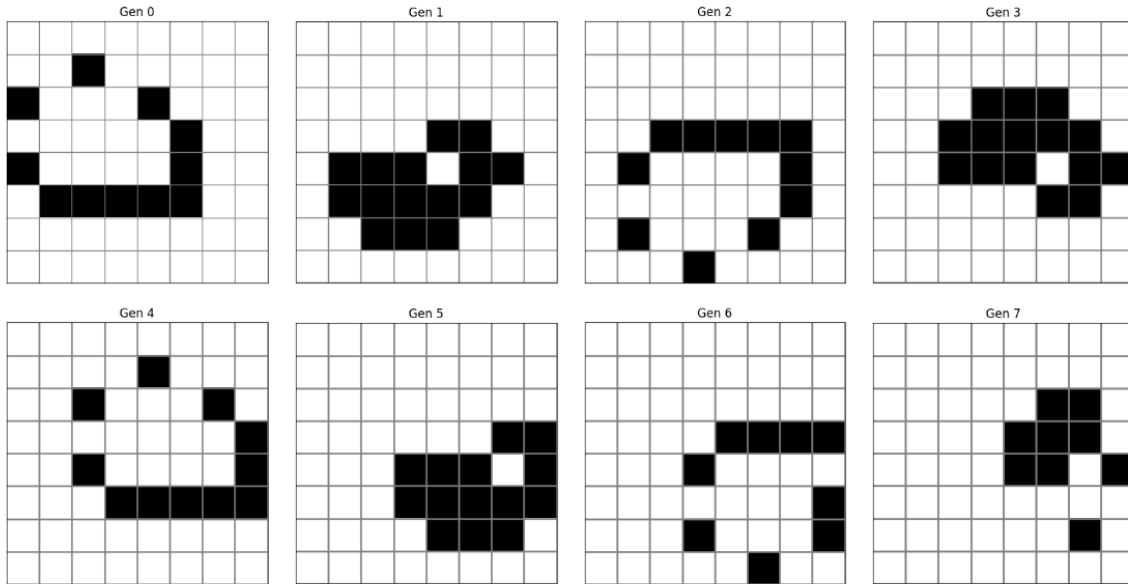


Figura 17: Patrón MWSS (Nave de Peso Medio del Espacio): se desplaza horizontalmente por el tablero, el tamaño intermedio entre las naves espaciales LWSS y HWSS.

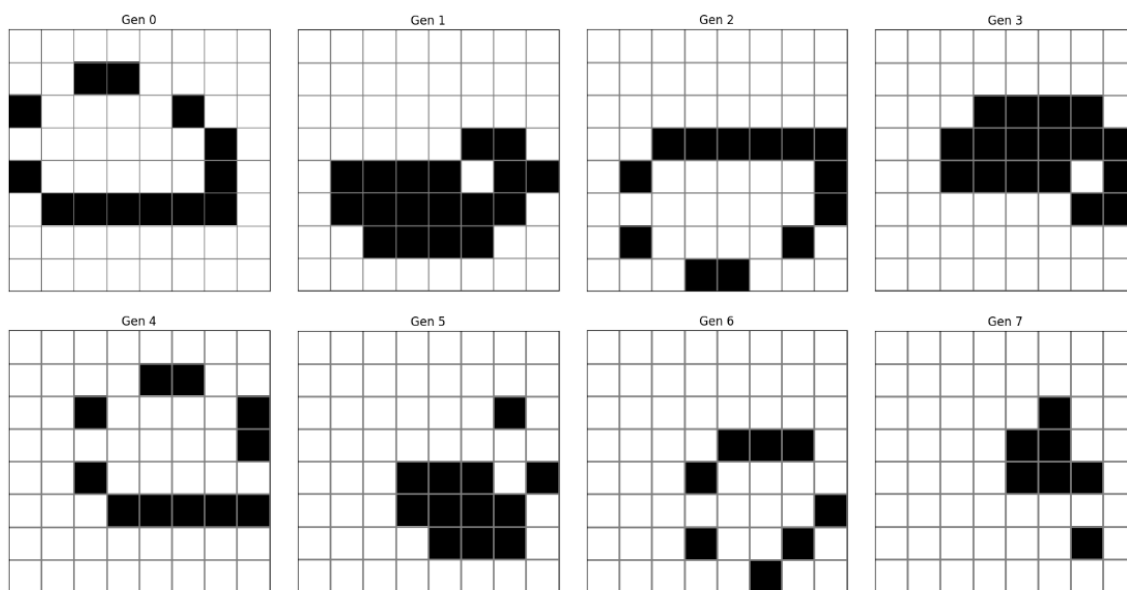


Figura 18: Patrón HWSS (Nave Pesada del Espacio): es la más grande de las tres naves espaciales básicas, se desplaza horizontalmente por el tablero hacia la derecha infinitamente.

#### 4.4. Methuselahs

Los Methuselahs son patrones que evolucionan durante un número grande de generaciones antes de estabilizarse en un patrón estático, un oscilador o una nave espacial. El ejemplo más conocido es el ".<sup>a</sup>corn", que evoluciona durante 5206 generaciones antes de estabilizarse en 633 células vivas, incluyendo 11 osciladores, 2 naves espaciales y 1 patrón estático.

#### 4.5. Patrones Propios

Sección dedicada a patrones que encontramos por accidente/pruebas.

#### 4.6. Patrones complejos

Además de estos patrones simples, también existen patrones más complejos en el Juego de la Vida. Algunos de estos pueden "disparar" naves espaciales, otros pueden construir "patrones adicionales, y otros pueden incluso comportarse como máquinas de Turing universales, lo que significa que pueden computar cualquier función computable.

Estos patrones ilustran la increíble diversidad de comportamientos que pueden surgir del Juego de la Vida. En las siguientes secciones, veremos cómo estos patrones pueden ser generados e investigados usando PowerDEVS.

### 5. Conclusiones