UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO DE FIN DE GRADO

Optimización Heurística en el Juego de la Vida de Conway: Algoritmos Genéticos y Aprendizaje Automático para la Simulación de Sistemas Astrofísicos

Heuristic Optimization in Conway's Game of Life: Genetic Algorithms and Machine Learning for the Simulation of Astrophysical Systems

Presentado por: **Jorge Bravo Mateos**

Dirigido por: **Rafael del Vado Vírseda**

Grado en Ingeniería de Computadores

Curso académico 2024-25

Agradecimientos

A mi tutor, familia y amigos.

Resumen

Este Trabajo de Fin de Grado presenta el diseño y desarrollo de un universo simulado mediante un autómata celular modificado, inspirado en el clásico Juego de la Vida de Conway. A partir de una cuadrícula de células que representan distintos tipos de materia —como estrellas, planetas y agujeros negros— se han implementado reglas astrofísicas que permiten observar la evolución dinámica del sistema a lo largo de miles de generaciones. El modelo incorpora elementos clave, como la interacción gravitacional, mecanismos de detección de patrones y un módulo de optimización heurística basado en algoritmos genéticos. Esta combinación ha permitido encontrar configuraciones iniciales que favorecen la aparición de estructuras complejas y coherentes, similares a la red cósmica observada en el Universo real.

Palabras clave: *Optimización heurística, autómatas celulares, Juego de la Vida, algoritmos genéticos, aprendizaje automático, simulación, autoorganización, astrofísica.*

Abstract

This Bachelor's degree final thesis presents the design and development of a simulated universe based on a modified cellular automaton, inspired by Conway's classic Game of Life. A grid of cells representing different kinds of matter, such as stars, planets and black holes, has been used to implement astrophysical rules and observe the dynamic evolution of the system over thousands of generations. Key elements incorporated into the model include gravitational interaction, pattern detection mechanisms, and a heuristic optimization module based on genetic algorithms. This combination has made it possible to find initial configurations that favour the emergence of complex and coherent structures, similar to the cosmic network observed in the real Universe.

Keywords: *Heuristic optimization, cellular automata, Game of Life, genetic algorithms, machine learning, simulation, self-organisation, astrophysics.*

Índice general

1. Introducción **12**

1.1. Contexto del Trabajo 12

1.2. Motivación del Proyecto 13

1.3. Objetivos de la Memoria 13

1.4. Estructura del Documento 14

2. Contexto Teórico **19**

2.1. Autómatas Celulares y el Juego de la Vida 19

2.1.1. Definición de autómata celular 19

2.1.2. Reglas básicas del Juego de la Vida de Conway 20

2.1.3. Ejemplos conocidos de patrones 21

2.1.3.1. *Still lifes* 21

2.1.3.2. *Oscillators* 21

2.1.3.3. *Spaceships* 22

2.1.3.4. Otras configuraciones 22

2.2. Algoritmos de Optimización Heurística 23

2.2.1. Introducción a la optimización heurística 23

2.2.2. Algoritmos evolutivos e inspiración biológica 23

2.2.3. Algoritmos genéticos 24

2.3. Aprendizaje Automático 25

2.3.1. Aprendizaje supervisado vs. no supervisado 25

2.3.2. Extracción de características de patrones 25
2.3.3. Reducción de dimensionalidad 25
2.3.4. Clustering de configuraciones 26
2.3.5. Visualización y análisis de grupos 26
2.4. Analogía con Sistemas Astrofísicos 26
2.4.1. Formación y evolución de cúmulos galácticos 26
2.4.2. Dinámica gravitacional emergente 27
2.4.3. Simulación de estructuras autoorganizadas en astrofísica 28
2.4.4. El Juego de la Vida como modelo conceptual de organización espacial 29

3. El Juego de la Vida Clásico 31
3.1. Implementación del Juego de la Vida Clásico 31
3.1.1. Configuración del entorno de desarrollo 31
3.1.2. Estructura de datos y representación del universo 32
3.1.3. Aplicación de las reglas del Juego de la Vida 33
3.1.4. Sistema de visualización 33
3.2. Implementación del Algoritmo Genético 34
3.2.1. Diseño de la población inicial 34
3.2.2. Función de aptitud 35
3.2.3. Operadores genéticos 36
3.2.4. Visualización y análisis de resultados evolutivos 37
3.3. Implementación del sistema de detección de patrones 40
3.3.1. Definición de patrones conocidos 40
3.3.2. Algoritmos de matching y detección 40
3.3.3. Integración con la evaluación genética 42
3.3.4. Visualización de patrones detectados 42
3.4. Implementación del Aprendizaje Automático 45
3.4.1. Arquitectura del sistema integrado 45

3.4.2. Extracción de características avanzadas 45
3.4.3. Construcción del conjunto de datos 46
3.4.4. Modelo de red neuronal profunda 47
3.4.4.1. Arquitectura de la red neuronal 47
3.4.4.2. Flujo de datos 48
3.4.4.3. Funciones de activación y su razonamiento 49
3.4.4.4. Regularización: *dropout* 49
3.4.4.5. Función de Pérdida: *binary cross-entropy* 49
3.4.4.6. Proceso de optimización y entrenamiento 49
3.4.5. Integración con la interfaz gráfica 50

4. El Juego de la Vida Extendido 53

4.1. Juego de la Vida con Reglas Astrofísicas 53
4.1.1. Estructura de datos y representación del Universo 53
4.1.2. Inicialización del Universo 54
4.1.3. Implementación de la física del sistema 54
4.1.3.1. Cálculo de gravedad local 54
4.1.3.2. Reglas de evolución del Universo 55
4.1.3.3. Comportamientos específicos de cuerpos celestes 57
4.1.3.4. Dinámica de agujeros negros 58
4.1.4. Sistema de visualización 59
4.2. Incorporación de Algoritmos Genéticos 61
4.2.1. Diseño del sistema evolutivo 61
4.2.2. Configuración genética 61
4.2.2.1. Función de aptitud 61
4.2.2.2. Operadores genéticos 62
4.2.3. Ciclo evolutivo 63
4.2.4. Visualización de resultados 63

4.3. Integración del Aprendizaje Automático 64
4.3.1. Arquitectura de la red neuronal 64
4.3.2. Extracción de características 65
4.3.3. Modelo de autoencoder convolucional 65
4.3.4. Entrenamiento adaptativo del modelo 66
4.3.5. Integración con la interfaz de visualización 67

5. Resultados y Análisis **69**
5.1. Configuración Experimental 69
5.1.1. Parámetros iniciales 69
5.2. Resultados de Simulaciones Básicas 70
5.2.1. Dinámica de población por tipo de célula 70
5.2.2. Formación de estructuras complejas 71
5.3. Análisis de Sensibilidad de Parámetros 72
5.3.1. Influencia de las masas relativas 72
5.3.2. Efecto de la tasa de pérdida de masa en agujeros negros 72
5.3.3. Variación en las reglas de interacción 73
5.3.3.1. Limitación de masa máxima en agujeros negros 73
5.3.3.2. Estancamiento del sistema estelar 74
5.4. Optimización Mediante Algoritmos Genéticos 76
5.4.1. Configuración del algoritmo genético 76
5.4.2. Resultados de la evolución genética 76
5.4.3. Algoritmos genéticos con supresión de formación de agujeros negros 78
5.4.3.1. Modificaciones implementadas 78
5.4.3.2. Resultados comparativos 78
5.5. Escalabilidad 80

6. Conclusiones y Trabajo Futuro **83**
6.1. Conclusiones 83

6.2. Trabajo Futuro 84

Capítulo 1

Introducción

Este primer capítulo introduce el concepto de autoorganización en la Naturaleza, y cómo los autómatas celulares, ejemplificados mediante el Juego de la Vida, sirven como modelos computacionales para poder realizar su simulación y estudio. El presente proyecto se ve motivado por la búsqueda concreta de conexiones entre estos modelos simples de simulación y la complejidad de fenómenos astrofísicos reales de nuestro Universo. Los objetivos del trabajo se centrarán en esta simulación y análisis de los patrones resultantes, utilizando, para ello, técnicas de optimización heurística y aprendizaje automático. Finalmente, se muestra la planificación, estructura y descripción del desarrollo de la memoria en capítulos para facilitar la comprensión del lector.

1.1. Contexto del Trabajo

Es bien sabido que la Naturaleza tiene una gran capacidad para crear orden y complejidad a partir de procesos muy simples. Este fenómeno, conocido como *autoorganización*, se puede observar en múltiples sistemas naturales, desde la formación de cristales de hielo hasta la estructura de galaxias espirales. La autoorganización constituye así uno de los principios fundamentales que rigen la formación de estructuras complejas en la Naturaleza. Desde la escala microscópica hasta la cósmica, observamos sistemas que, espontáneamente, desarrollan patrones ordenados sin necesidad de un control externo. El presente trabajo se contextualiza dentro de la intersección entre la informática y la astrofísica, estudiando cómo modelos computacionales muy simples pueden llegar a imitar procesos naturales autoorganizativos y a revelar patrones similares a los observados en estructuras cósmicas.

Dentro de este contexto, los *autómatas celulares* representan uno de los modelos computacionales más potentes para estudiar fenómenos de autoorganización. Estos sistemas discretos, compuestos por celdas que evolucionan según reglas locales bien definidas, pueden generar comportamientos complejos partiendo de condiciones iniciales simples. El ejemplo más paradigmático es el *Juego de la Vida* del matemático inglés John Horton Conway (1937-2020), en el que, siguiendo únicamente cuatro reglas elementales, emergen patrones dinámicos como *osciladores*, *estructuras estables* y *naves espaciales*, los cuales se desplazan por el espacio del autómata celular.

La similitud entre los patrones generados por estos modelos computacionales y las estructuras observadas en sistemas naturales, particularmente en el ámbito astrofísico, no es algo casual, sugiriendo principios universales en los procesos de autoorganización con independencia de la escala o de la naturaleza del sistema. De hecho, el pionero alemán de la informática Konrad Zuse (1910-1995), propuso en 1969 la hipótesis de que el Universo es, en realidad, un gigantesco autómata celular universal (o, equivalentemente, una *máquina de Turing*), dando lugar así a la que hoy en día se conoce como la *Hipótesis de la Simulación* [1].

1.2. Motivación del Proyecto

El Juego de la Vida de Conway, a pesar de su aparente simplicidad, ha demostrado ser computacionalmente universal, es decir, un modelo computacional capaz de simular cualquier algoritmo, lo que le convierte, gracias a dicha versatilidad, en una herramienta imprescindible para modelar y simular procesos naturales. Adicionalmente, con la ayuda de técnicas de computación bioinspirada, como los *algoritmos genéticos*, inspirados en los principios de la selección natural, podemos hacer evolucionar las configuraciones iniciales, seleccionando aquellas que mejor se vayan adaptando a los criterios seleccionados [2].

Actualmente, a este estudio evolutivo le podemos sumar el uso de técnicas de *aprendizaje automático*, útiles para analizar, clasificar, identificar y descubrir, automáticamente, patrones a partir de los datos generados previamente. Gracias a estas técnicas, indispensables no solo para poder mejorar nuestra capacidad de comprender la dinámica de los autómatas celulares, es posible revelar principios aparentemente "escondidos" que permiten conectar los modelos computacionales con sistemas naturales; descubrimos así nuevas analogías con fenómenos naturales, como, en particular, los de carácter astrofísico que perseguimos estudiar en esta memoria.

1.3. Objetivos de la Memoria

El *objetivo principal* de este trabajo es la simulación de estructuras análogas a las observadas en sistemas astrofísicos, mediante el uso de algoritmos genéticos para poder optimizar, heurísticamente, tanto las configuraciones iniciales como las reglas de transición en variantes del Juego de la Vida. Para ello, orientaremos su evolución mediante funciones de aptitud (*fitness*) diseñadas para favorecer la emergencia de patrones y estructuras semejantes a las estructuras cósmicas.

Otro de los objetivos de la memoria consiste en analizar los patrones obtenidos a través de diversas técnicas de aprendizaje automático, con el propósito de clasificar su comportamiento y estructura, y ver si existen paralelismos entre los patrones generados por nuestros autómatas celulares y fenómenos astrofísicos y cosmológicos reales.

1.4. Estructura del Documento

El presente documento se organiza en cinco capítulos:

- **Capítulo 2: Contexto teórico.** Como su nombre indica, este capítulo se centra en la definición general de autómata celular, y en la explicación particular del funcionamiento del Juego de la Vida de Conway. Introduce los principios de la optimización heurística, con especial énfasis en los algoritmos genéticos y en sus operadores de selección, cruce y mutación. Por último, explora las conexiones teóricas entre estos sistemas discretos y los fenómenos astrofísicos.
- **Capítulo 3: El Juego de la Vida clásico.** Presenta una implementación avanzada del autómata celular clásico mediante la integración de algoritmos genéticos, con un sistema adicional propio para la detección de patrones, al que se le añade, adicionalmente, un sistema de detección de patrones basado en aprendizaje automático.
- **Capítulo 4: El Juego de la Vida extendido.** Expande el modelo clásico mediante la ampliación de dos a siete estados celulares, la adición de masas y dinámicas físicas inspiradas en procesos cósmicos, y la integración de algoritmos genéticos. Mediante ellos es posible optimizar configuraciones iniciales para la mejora de los resultados. Además, se integra en el sistema una red neuronal convolucional, para la clasificación automática de patrones.
- **Capítulo 5: Resultados y análisis.** En este capítulo se incluye un análisis sistemático del modelo extendido finalmente diseñado, evaluando la sensibilidad de sus parámetros y su influencia en la dinámica del Universo. Asimismo, se ajustan varias reglas para observar su impacto en el proceso evolutivo; con ello, se busca comprender el comportamiento del sistema y la razón de las configuraciones elegidas.
- **Capítulo 6: Conclusiones y trabajo futuro**. Finalmente, en este último capítulo se extraen las principales conclusiones a partir de los resultados obtenidos y del análisis realizado en los capítulos anteriores. A partir de estas conclusiones, esbozamos algunas de las líneas de trabajo que pueden resultar más interesantes desarrollar en un futuro próximo.

Repositorio de código: el código fuente completo, junto con las imágenes y resultados de las ejecuciones de este trabajo, se encuentran disponibles públicamente en el siguiente GitHub:

`https://github.com/jbrma/TFG_JorgeBravo`.

Capítulo 1

Introduction

This first chapter introduces the concept of self-organisation in Nature, and how cellular automata, exemplified by the Game of Life, serve as computational models for their simulation and study. The present project is motivated by the concrete search for connections between these simple simulation models and the complexity of real astrophysical phenomena in our Universe. The objectives of the work will focus on this simulation and analysis of the resulting patterns, using heuristic optimization and machine learning techniques. Finally, the planning, structure and description of the development of the work are shown in chapters to facilitate the reader's understanding.

1.1. Context of the Work

It is well known that Nature has a great capacity to create order and complexity from very simple processes. This phenomenon, known as *self-organisation*, can be observed in many natural systems, from the formation of ice crystals to the structure of spiral galaxies. Self-organisation is thus one of the fundamental principles governing the formation of complex structures in nature. From the microscopic to the cosmic scale, we observe systems that spontaneously develop ordered patterns without the need for external control. The present work is contextualised within the intersection between computer science and astrophysics, studying how very simple computational models can emulate natural self-organising processes and reveal patterns similar to those observed in cosmic structures.

In this context, *cellular automata* represent one of the most powerful computational models for studying self-organising phenomena. These discrete systems, composed of cells that evolve according to well-defined local rules, can generate complex behaviours starting from simple initial conditions. The most paradigmatic example is the *The Game of Life* of the English mathematician John Horton Conway (1937-2020), in which, following only four elementary rules, dynamic patterns emerge such as *oscillators*, *stable structures* and *spaceships*, which move through the space of the cellular automaton.

The similarity between the patterns generated by these computational models and the structures observed in natural systems, particularly in the astrophysical field, is not casual, suggesting universal principles in the processes of self-organisation independently of the scale or nature of the system. In fact, the German computer pioneer Konrad Zuse (1910-1995), proposed in 1969 the hypothesis that the Universe is, in reality, a gigantic universal cellular automaton (or, equivalently, a *Turing machine*), thus giving rise to what is known today as the *Simulation Hypothesis* [1].

1.2. Project Motivation

Conway's Game of Life, despite its apparent simplicity, has proven to be computationally universal; in other words, a computational model capable of simulating any algorithm, which makes it, thanks to this versatility, an essential tool for modelling and simulating natural processes. Additionally, with the help of bio-inspired computation techniques, such as *genetic algorithms*, inspired by the principles of natural selection, we can make the initial configurations evolve, selecting those that best adapt to the selected criteria [2].

Currently, to this evolutionary study we can add the use of *machine learning* techniques, useful for analysing, classifying, identifying and automatically discovering patterns from previously generated data. Thanks to these techniques, which are indispensable not only to improve our ability to understand the dynamics of cellular automata, it is possible to reveal apparently "hidden" principles that allow us to connect computational models with natural systems; we thus discover new analogies with natural phenomena, such as, in particular, those of an astrophysical nature that we aim to study in this work.

1.3. Objectives of the Work

The *main objective* of this work is the simulation of structures similar to those observed in astrophysical systems, using genetic algorithms to heuristically optimise both the initial configurations and the transition rules in variants of the Game of Life. To this end, we will guide their evolution by fitness functions designed to favour the emergence of patterns and structures similar to cosmic structures.

Another of the goals of this work is to analyse the patterns obtained through various machine learning techniques, in order to classify their behaviour and structure, and see if there are any parallels between the patterns generated by our cellular automata and real astrophysical and cosmological phenomena.

1.4. Structure of the Document

This document is organised into five chapters:

- **Chapter 2: Theoretical context.** As the name suggests, this chapter focuses on the general definition of cellular automaton, and on the particular explanation of the operation of Conway's Game of Life. It introduces the principles of heuristic optimization, with special emphasis on genetic algorithms and their selection, crossover and mutation operators. Finally, it explores the theoretical connections between these discrete systems and astrophysical phenomena.
- **Chapter 3: The Classical Game of Life.** It presents an advanced implementation of the classical cellular automaton by integrating genetic algorithms, with an additional proprietary system for pattern detection, to which a machine learning-based pattern detection system is additionally added.
- **Chapter 4: The Extended Game of Life.** It expands the classical model by extending it from two to seven cell states, adding masses and physical dynamics inspired by cosmic processes, and integrating genetic algorithms. Through these

it is possible to optimise initial configurations for improved results. In addition, a convolutional neural network is integrated into the system for automatic pattern classification.

- **Chapter 5: Results and analysis.** This chapter includes a systematic analysis of the finally designed extended model, evaluating the sensitivity of its parameters and their influence on the dynamics of the Universe. Several rules are also adjusted to observe their impact on the evolutionary process, in order to understand the behaviour of the system and the reason for the chosen configurations.
- **Chapter 6: Conclusions and future work.** Finally, this last chapter draws the main conclusions from the results obtained and the analysis carried out in the previous chapters. Based on these conclusions, we suggest some of the lines of work that may be most interesting to develop in the near future.

Code repository: The full source code, along with images and results of runs of this work are publicly available on the following GitHub:

`https://github.com/jbrma/TFG_JorgeBravo`.

Capítulo 2

Contexto Teórico

Este capítulo establece los fundamentos conceptuales y metodológicos que sirven de contexto teórico para nuestro trabajo. Sus objetivos son tres: presentar los autómatas celulares y el Juego de la Vida como sistema computacional de referencia, describir técnicas de optimización heurística y aprendizaje automático aplicables al análisis y detección de patrones, y explorar analogías de los modelos teóricos presentados con la astrofísica. En concreto, la **Sección 2.1** introduce formalmente los conceptos de autómata celular y el Juego de la Vida, basándonos en el libro *The Recursive Universe* [3]. Las **Secciones 2.2** y **2.3** están enfocadas a los algoritmos genéticos [4] y a el uso del aprendizaje automático, respectivamente. Finalmente, la **Sección 2.4** establece paralelismos interesantes con fenómenos astrofísicos, integrando ambas perspectivas, computacional y cosmológica.

2.1. Autómatas Celulares y el Juego de la Vida

Para entender cómo funciona el Juego de la Vida, antes es necesario dar una breve explicación sobre qué son los *autómatas celulares*, ya que estos son la base computacional del propio juego.

2.1.1. Definición de autómata celular

Los **autómatas celulares** son modelos matemáticos que simulan el comportamiento de sistemas complejos mediante la aplicación de unas reglas simples aplicadas a una cuadrícula de *células*. Cada célula puede estar en uno de varios estados posibles; estos estados son alterados en el tiempo según unas reglas previamente definidas.

Formalmente, un autómata celular puede definirse como una tupla compuesta por:

- Una *red* o *rejilla*, que suele representarse como una cuadrícula bidimensional.
- Un *vecindario*, es decir, el conjunto de posiciones alrededor de una célula que influyen en su evolución.
- Una *función de transición*, que determina el nuevo estado de una célula a partir de su estado actual y el de sus vecinos.

La evolución del sistema ocurre en pasos discretos llamados *generaciones*. En cada generación, todas las células de la cuadrícula actualizan su estado simultáneamente, aplicando la función de transición determinada.

El aspecto que más caracteriza a los autómatas celulares es su capacidad de lograr emerger y visualizar una serie de propiedades, que surgen de la propia dinámica local a través del paso del tiempo, y no desde un inicio [5].

2.1.2. Reglas básicas del Juego de la Vida de Conway

El **Juego de la Vida**, creado por John Horton Conway en 1970, es uno de los autómatas celulares más famosos y estudiados. Se desarrolla en una cuadrícula bidimensional infinita, donde cada célula puede estar *viva* o *muerta*. Su evolución depende de una serie de reglas muy simples basadas en el estado de las células vecinas, es decir, de sus 8 células adyacentes. Las reglas fundamentales son:

- **Supervivencia:** una célula viva con 2 o 3 células vecinas vivas permanece viva en la siguiente generación.
- **Soledad:** una célula viva con una o menos células vecinas vivas muere.
- **Sobrepoblación:** una célula viva con más de 3 células vecinas vivas muere.
- **Nacimiento:** una célula muerta con exactamente 3 células vecinas vivas revive.

Estas reglas, aparentemente simples, pueden llegar a generar estructuras muy complejas a partir de un estado inicialmente aleatorio. Hasta ahora, se han descubierto y documentado 1.720 patrones en la página Web destinada al Juego de la Vida [6], en la cual se siguen haciendo aportaciones de nuevos hallazgos a día de hoy. A continuación, se muestra una clasificación de los patrones más importantes (*still lifes, oscillators, spaceships*), así como de otras estructuras.

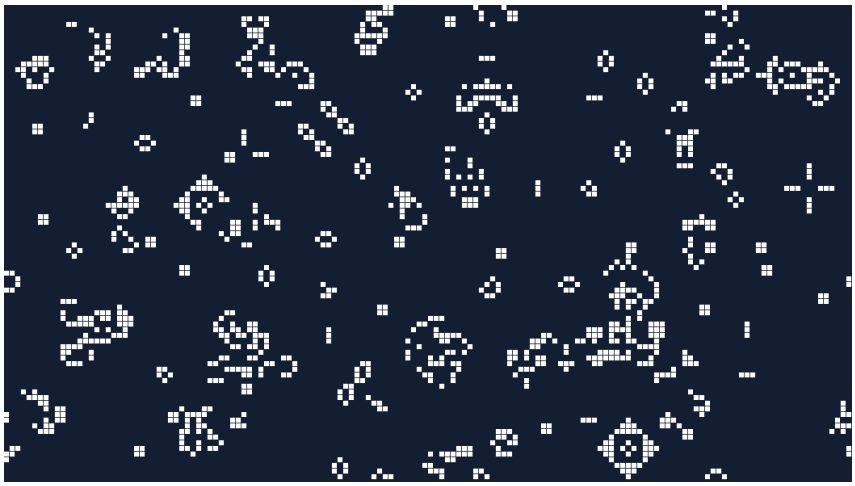


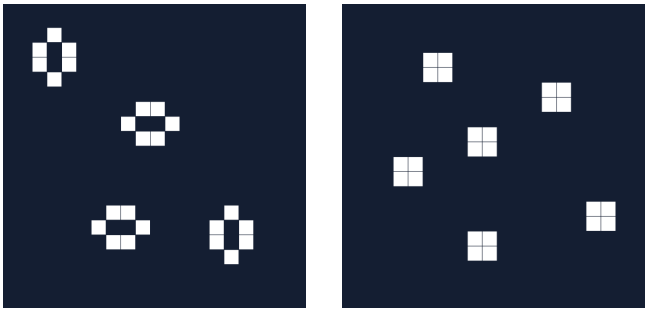
Figura 2.1: Ejemplo de un universo del Juego de la Vida.

2.1.3. Ejemplos conocidos de patrones

Esta subsección explora patrones importantes en el Juego de la Vida, clasificados por su comportamiento dinámico. Se analizan estructuras estáticas (*still lifes*), oscilantes (*oscillators*), móviles (*spaceships*), y configuraciones complejas, como el *Gosper Glider Gun*, donde se destacan sus propiedades emergentes.

2.1.3.1. *Still lifes*

Como se puede deducir por su nombre (*vidas estáticas*), este patrón está formado por un conjunto de células que forman estructuras estáticas, es decir, que no cambian de una generación a la siguiente. Por su estructura interna, y suponiendo que no hay colisiones con otras formas de vida, se mantienen estables en el transcurso del tiempo. Dos ejemplos comunes pueden ser los patrones *beehives* y *blocks*:



(a) Beehives (b) Blocks

Figura 2.2: *Still lifes.*

2.1.3.2. *Oscillators*

Los *osciladores* son patrones que alternan entre dos o más estados de forma cíclica; tras una serie de iteraciones, vuelven a tener la misma disposición de células que al principio. Tienen tres parámetros:

- *Período*: el número de iteraciones necesarias para volver al estado inicial.
- *Stator*: las células que permanecen vivas durante todo el proceso.
- *Rotor*: las células que cambian su estado en algún momento del proceso.

El patrón más común es el denominado *blinker* (**Figura 2.3**), el cual se compone de 3 células vivas que cambian de orientación (horizontal o vertical) en cada iteración. Este patrón tiene período 2, que es el más pequeño posible para los osciladores, pero existen osciladores mucho más complejos con infinidad de períodos, operando principios fundamentales de autoorganización.

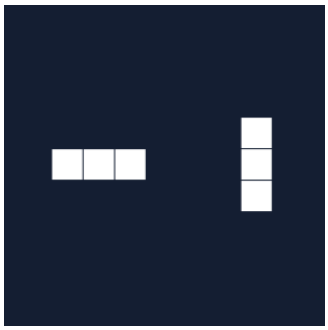


Figura 2.3: Dos estados del *blinker*.

2.1.3.3. *Spaceships*

Al igual que en los osciladores, estos patrones vuelven a su estado inicial tras un determinado número de iteraciones, pero su principal diferencia es que su posición final en el tablero es diferente; esto permite que dichos patrones puedan moverse a lo largo de todo el universo.

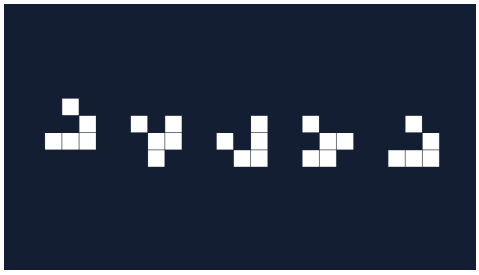


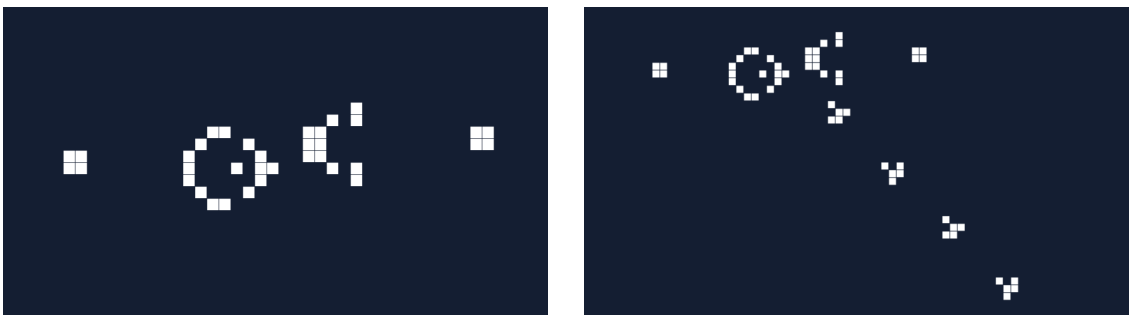
Figura 2.4: Paso a paso de la ejecución del *glider*.

A este tipo de patrones se les añade un parámetro más, la *velocidad*, la cual se calcula cogiendo la distancia (número de celdas entre la posición inicial y final) y dividiéndola entre su período. En el caso del *glider*, como podemos observar en la **Figura 2.4**, cada 4 iteraciones vuelve a su estado original.

2.1.3.4. Otras configuraciones

El *Gosper Glider Gun* (**Figura 2.5**), descubierto por Bill Gosper en 1970, fue el primer patrón conocido capaz de generar *gliders* de manera ilimitada. Esta configuración de 36 células produce un nuevo *glider* cada 30 iteraciones, demostrando que el Juego de la Vida puede crear patrones que crecen indefinidamente. El descubrimiento del *Gosper Glider Gun* tuvo implicaciones significativas [7]:

- **Computación universal:** demostró que el Juego de la Vida es *Turing-completo*, lo que significa que puede calcular cualquier "función computable", y, por tanto, simular una computadora universal.
- **Patrones complejos:** abrió la puerta a la creación de estructuras más complejas, como:
  - *Breeder*: patrones que genera múltiples *glider guns*, creciendo cuadráticamente.
  - *Replicadores*: estructuras capaces de copiarse a sí mismas.
- **Máquinas de Turing:** se han podido construir máquinas de Turing completas dentro del Juego de la Vida, demostrando así su poder computacional.



(a) Estado inicial. (b) Estado tras 120 generaciones.

Figura 2.5: *Gosper Glider Gun*.

Estas configuraciones avanzadas demuestran que, a partir de reglas simples, pueden emerger comportamientos extremadamente complejos. Este fenómeno es un tema central, el cual tiene implicaciones en campos que van, desde la biología o las matemáticas, hasta la física y la informática. En este trabajo, veremos cómo también tiene implicaciones en el campo de la cosmología y la astrofísica.

2.2. Algoritmos de Optimización Heurística

En esta sección examinamos algunas de las técnicas heurísticas para la exploración eficiente de espacios de búsqueda complejos. En concreto, nos centraremos en los algoritmos genéticos (**Sección 2.2.3**), detallando su inspiración biológica e implementación.

2.2.1. Introducción a la optimización heurística

Un **problema de optimización** consiste en encontrar la mejor solución dentro de un conjunto de alternativas posibles, definido por restricciones y objetivos específicos. Formalmente, se busca maximizar o minimizar una **función objetivo**, la cual define, formalmente, el problema de optimización. En problemas complejos, los métodos clásicos de optimización exacta suelen volverse computacionalmente inviables, debido al crecimiento exponencial del espacio de búsqueda.

La **optimización heurística** aborda estos desafíos mediante estrategias que sacrifican garantías de optimalidad a cambio de eficiencia computacional, permitiendo encontrar soluciones de alta calidad en tiempos razonables, aunque no garantiza la optimalidad de las soluciones encontradas, ni determina lo lejos que se está de poder alcanzarlas. La **función de aptitud** es la encargada de transformar las soluciones candidatas en valores numéricos para guíar a la selección natural, detallada en las siguientes secciones.

2.2.2. Algoritmos evolutivos e inspiración biológica

Los **algoritmos evolutivos** establecen estrategias mediante procesos que imitan el comportamiento de los seres vivos, para dirigir, en un espacio de búsqueda, la evolución de un conjunto de soluciones de un problema con la intención de acercarse a su solución óptima. Para ello, emplean mecanismos y operadores inspirados en la evolución biológica, como la *selección natural*, *la mutación* y el *cruzamiento*, que permiten mantener la diversidad y explorar nuevas regiones del espacio de soluciones.

Entre las metaheurísticas evolutivas, uno de los representantes más populares son los **algoritmos genéticos** (GA) [8]. Estos algoritmos se inspiran en los principios de herencia y selección natural formulados por Darwin, y han demostrado ser eficaces en la resolución de problemas complejos y de gran escala. No obstante, existen otros tipos de algoritmos evolutivos, como las **nubes de partículas** (PSO) y la **evolución diferencial** (DE), cada uno de ellos con sus propios mecanismos para la generación y mejora de soluciones.

2.2.3. Algoritmos genéticos

Los algoritmos genéticos constituyen uno de los grupos de algoritmos más conocidos dentro de las metaheurísticas evolutivas. A medida que van pasando *generaciones* (iteraciones), los GA mantienen una población de individuos, cada uno representando una posible solución al problema. Los valores de dichos individuos evolucionan hacia mejores soluciones, es decir, individuos mejor adaptados al entorno definido por la función objetivo. Este proceso se desarrolla definiendo una serie de pasos fundamentales:

- **Selección** de individuos (equivalente a la *selección natural*), donde las soluciones más aptas tienen mayor probabilidad de reproducirse.
- **Cruzamiento** (o *cruce*), que combina la información genética de dos individuos para generar descendencia.
- **Mutación**, que introduce variaciones aleatorias para mantener la diversidad de la población y evitar una convergencia prematura.

La combinación de estos operadores permite que la población evolucione, progresivamente, hacia regiones del espacio de búsqueda con soluciones de mayor calidad. A continuación, se muestra un pseudocódigo muy simple de un algoritmo genético básico:

Algorithm 1 Pseudocódigo de un Algoritmo Genético

```
1: Inicialización: generar una población inicial de individuos.
2: Salida: evaluar la aptitud de cada individuo y devolver el mejor individuo encontrado.

3: while la condición de finizalización NO se cumple do
4:    Selección: seleccionar una población de padres basados en su aptitud.
5:    for cada par de padres do
6:       Cruce: cruzar los padres para crear uno o más descendientes.
7:    end for
8:    for cada individuo do
9:       Mutación: mutar con una probabilidad determinada.
10:   end for
11:   Evaluar la aptitud de cada individuo.
12: end while
13: Devolver: el mejor individuo encontrado.
```

El pseudocódigo sigue un flujo evolutivo estándar. Tras inicializar una población (línea 1), evalúa su aptitud (línea 2). El bucle principal (líneas 3-11) itera hasta cumplir condiciones de terminación (generaciones, convergencia). En cada iteración, selecciona padres (línea 4), los cruza (líneas 5-7), aplica mutaciones (líneas 8-10), y actualiza la población. Este proceso combina explotación (selección) con exploración (cruce/mutación).

2.3. Aprendizaje Automático

El **aprendizaje automático** (del inglés, **Machine Learning**, ML [9]) constituye un pilar fundamental en el análisis de sistemas complejos como los autómatas celulares. Esta sección explora las metodologías clave para procesar patrones emergentes en el Juego de la Vida, desde la preparación de los datos hasta la interpretación de los resultados.

2.3.1. Aprendizaje supervisado vs. no supervisado

El aprendizaje automático se divide en dos paradigmas principales, según la naturaleza de los datos [10]:

- **Aprendizaje supervisado:** utiliza conjuntos de entrenamiento etiquetados, donde cada muestra tiene una salida conocida, es decir, ya tiene una mínima comprensión sobre los valores de salida que son correctos. Resulta adecuado para problemas de clasificación (por ejemplo, la identificación de tipos de patrones en el Juego de la Vida) y regresión (por ejemplo, para predecir la evolución temporal de configuraciones).
- **Aprendizaje no supervisado:** funciona de forma independiente, operando sobre datos sin etiquetas y sin necesidad de directrices concretas, descubriendo así estructuras intrínsecas, mediante técnicas como la denominada *clustering*. Resulta crucial para analizar patrones emergentes que no hayan sido catalogados previamente.

2.3.2. Extracción de características de patrones

La *extracción* es un proceso en el que se transforman datos "crudos" (esto es, configuraciones celulares) en representaciones numéricas, de manera que sean procesables por los algoritmos [11]. Este proceso selecciona los atributos que sean más representativos del conjunto a analizar, de manera que se preserve la información relevante.

2.3.3. Reducción de dimensionalidad

Las **técnicas de reducción de la dimensionalidad** [12] sirven para representar un conjunto de datos utilizando un menor número de características, sin perder la parte esencial de los datos originales. Esto se consigue eliminando características redundantes o irrelevantes, y, con ello, creando un modelo con un número menor de variables.

- **PCA (Análisis de Componentes Principales):** es un método lineal que combina y transforma las características originales "condensándonlas" en tan solo unas pocas componentes, capaces de representar la mayoría (o la totalidad) de la varianza del conjunto original.
- **t-SNE (t-*Distributed Stochastic Neighbor Embedding*):** es una técnica de reducción no lineal que se centra en preservar las similitudes entre pares de puntos de datos en un espacio de dimensiones inferiores [13].

2.3.4. Clustering de configuraciones

El *clustering* es una técnica de aprendizaje no supervisado, utilizada para identificar grupos de configuraciones similares en un conjunto de datos. Dos de los algoritmos más populares para esta tarea son *K-Means* y *DBSCAN*, cada uno con características y aplicaciones distintas [14].

K-Means agrupa datos en un número fijo de *clusters* similares en tamaño. *DBSCAN*, en cambio, detecta automáticamente grupos de cualquier forma y tamaño, sin necesidad de definir su número; además, es robusto frente a valores atípicos, aunque depende de la elección de sus parámetros. *K-Means* es más eficiente en datos limpios y bien separados; *DBSCAN* es preferible cuando hay ruido o *clusters* irregulares.

2.3.5. Visualización y análisis de grupos

Una vez aplicadas las técnicas de reducción de dimensionalidad y *clustering*, la visualización es esencial para interpretar los resultados y entender cómo se agrupan los patrones generados.

2.4. Analogía con Sistemas Astrofísicos

La relación entre sistemas computacionalmente simples, como el Juego de la Vida, y fenómenos astrofísicos complejos, como el nacimiento de estrellas, ofrece una perspectiva que permite estudiar la autoorganización en la naturaleza. Aunque el Juego de la Vida se basa en reglas muy simples, su capacidad para generar patrones emergentes refleja los principios universales observados en la formación de galaxias, cúmulos estelares o en la propia red cósmica.

A través de esta analogía, se pretende explorar cómo las dinámicas del juego pueden servir como metáfora para comprender mecanismos astrofísicos, destacando similitudes como la acumulación de materia o la autoorganización.

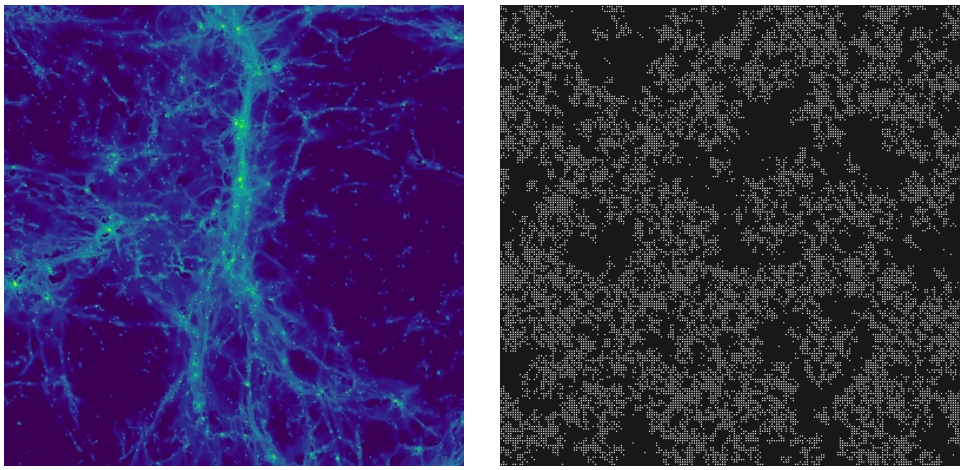
2.4.1. Formación y evolución de cúmulos galácticos

En astrofísica, los cúmulos galácticos se originan de la agregación de materia a niveles cósmicos. Inicialmente, pequeñas alteraciones de densidad en el Universo fueron creciendo debido a la gravedad, atrayendo gas, estrellas y materia hasta formar estructuras jerárquicas [15]. Este proceso, regido por las leyes físicas universales, puede crear patrones complejos como *filamentos* y *vacíos* en la red cósmica.

En el Juego de la Vida, patrones como *bloques* o *gliders* se generan desde reglas muy básicas (nacimiento, muerte y supervivencia). Análogamente, la "atracción" implícita en dichas reglas (por ejemplo, células que sobreviven si tienen 2 o 3 vecinas) simula una dinámica de acumulación selectiva, donde ciertas configuraciones iniciales evolucionan hacia estructuras estables. Así, la formación de un *bloque estático* en el juego refleja

la estabilización de un cúmulo galáctico: ambos son puntos de equilibrio en sistemas dinámicos.

En la **Figura 2.6**, se puede observar las analogías estructurales entre una simulación de una red cósmica a gran escala y una simulación del Juego de la Vida con algoritmos genéticos; esta simulación ha sido el resultado de una de las primeras pruebas que realizamos con la fusión del autómata celular y los algoritmos evolutivos.



(a) Simulación de una red cósmica. (b) Juego de la Vida con GA.

Figura 2.6: Analogías estructurales entre (a) red cósmica y (b) patrones autoorganizados.

(a) Se trata de una simulación cosmológica del Universo lejano. La imagen muestra la luz emitida por los átomos de hidrógeno en la red cósmica en una región de unos 15 millones de años-luz de diámetro. Se pueden observar varias fuentes puntuales, que son galaxias en proceso de formación de sus primeras estrellas [16].

(b) Es una de las primeras pruebas realizadas en el presente trabajo, fusionando las reglas básicas del Juego de la Vida con algoritmos genéticos, modificando sus parámetros, y favoreciendo la formación de pequeños cúmulos de vida.

2.4.2. Dinámica gravitacional emergente

La **emergencia** en astrofísica se refiere a la aparición de estructuras complejas a nivel macroscópico que no pueden predecirse fácilmente. Estas estructuras surgen a partir de la interacción de componentes individuales más simples. La gravedad actúa como fuerza de largo alcance, generando movimiento orbital y colisiones entre los cuerpos, las cuales, a su vez, inducen turbulencias e inestabilidades en el gas de los cúmulos, amplificando los campos magnéticos y acelerando las partículas cósmicas.

Al igual que cada par de masas se atrae según una regla sencilla, podemos definir estas dinámicas mediante un principio clave: los comportamientos macroscópicos complejos pueden surgir de reglas microscópicas simples; este principio resulta fundamental para explicar la autoorganización gravitacional.

2.4.3. Simulación de estructuras autoorganizadas en astrofísica

La **autoorganización** se define como un proceso espontáneo en el que un sistema desarrolla una estructura coherente a partir de interacciones entre sus componentes, sin ningún control externo. Dada la complejidad de la dinámica gravitacional, numerosos estudios computacionales se han visto motivados para modelar la formación de estructuras galácticas mediante diversos *esquemas de simulación*. Destacan, principalmente:

- **Autómatas celulares en cosmología:** aunque su uso en cosmología no es tan común como en otros campos, sí se han explorado algunos modelos CA para los fenómenos galácticos. Por ejemplo, *Lejeune y Perdang* (1996) diseñaron un autómata celular bidimensional de múltiples estados, para simular patrones de formación estelar en una región fuera del centro de una galaxia [17].

  Este modelo demuestra que, tanto la dinámica (circulación de gas) como la evolución física (formación de estrellas), pueden estar unificadas para así obtener una simulación más realista del funcionamiento de las galaxias.

- **Simulaciones de N-cuerpos:** en escalas mayores, las simulaciones de N-cuerpos han sido la herramienta estándar. Por ejemplo, el proyecto *Millennium* es una de las simulaciones cosmológicas más ambiciosas [18], en la que se rastrea la evolución de la distribución de materia en un "pequeño" espacio del Universo, utilizando más de 10 mil millones de partículas para representar la materia oscura. Esto permite modelar cómo las estructuras cósmicas se formaron y evolucionaron a lo largo de 13,6 mil millones de años, desde poco después del *Big Bang* hasta el presente.

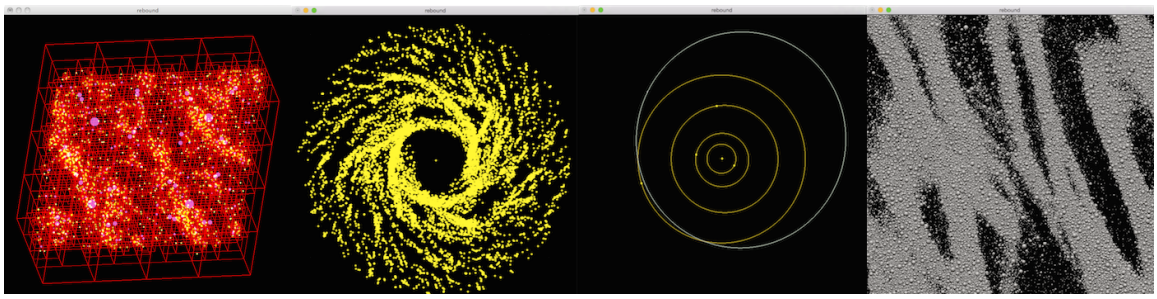


Figura 2.7: Ejemplos de simulaciones físicas generadas con *Rebound*.

  Además, existen códigos públicos como *Rebound* [19], que fueron diseñados para la dinámica de colisiones, como los anillos planetarios, pero que también son capaces de representar, detalladamente, el problema clásico de los N-cuerpos, así como simular la interacción gravitacional entre partículas con una precisión muy alta.

- **Análisis de redes y percolación:** otro de los sistemas es el estudio de la estructura cósmica mediante *teoría de grafos*. Aunque este tipo de estudios son más analíticos, muestran cómo la red cósmica puede asemejarse a un grafo auto-organizado, donde los nodos de alta densidad (cúmulos) se unen mediante filamentos a grandes distancias.

2.4.4. El Juego de la Vida como modelo conceptual de organización espacial

Tal y como se mencionó en la sección del Juego de la Vida, esta simulación se construye mediante un autómata celular bidimensional, el cual está completamente determinado por un estado inicial. Además, a pesar de que sus reglas son extremadamente simples, el sistema genera una variedad de patrones dinámicos de gran complejidad. En el *universo gravitacional*, las reglas locales (la gravitación entre partículas) son análogas a las reglas de nacimiento/muerte del juego. En ambos casos, la simplicidad de las reglas contrasta con la complejidad del resultado final: la red cósmica tridimensional de galaxias y materia oscura, por un lado, y el tejido bidimensional de células vivas y muertas en el juego, por otro.

Conceptualmente, ambos sistemas son deterministas, y, sin embargo, su evolución es muy difícil de predecir en detalle. Además, como ya hemos indicado, el Juego de la Vida es computacionalmente universal (es decir, Turing-completo), lo que significa que puede simular cualquier proceso computable. Esta característica señala su potencia para generar comportamientos enrevesados, del mismo modo que el cosmos real puede considerarse como un sistema "computacional" regido por leyes físicas.

Es indiscutible que hay claras diferencias; el Juego de la Vida es *discreto*, en una malla infinita 2D, y no contiene física newtoniana en absoluto. En contraposición, las galaxias se mueven en un espacio tridimensional *continuo* con fuerzas de largo alcance y, probablemente, sea finito. Aunque el Juego de la Vida básico no reproduce las ecuaciones de la gravedad ni los detalles físicos del Universo, sí comparte los principios de organización emergente y autoorganización espacial. Como tal, funciona como una *metáfora instructiva*: destaca cómo patrones pueden aparecer "de la nada" (sin agente externo) en sistemas regidos por reglas elementales, un tema central tanto en complejidad computacional como en astrofísica.

Capítulo 3

El Juego de la Vida Clásico

Este capítulo detalla la metodología que hemos empleado para implementar, de manera versátil, un sistema evolutivo clásico basado en el Juego de la Vida de Conway. El objetivo principal consistirá en desarrollar una herramienta que permita estudiar la emergencia de patrones complejos mediante algoritmos genéticos [2, 4]. En consecuencia, desarrollaremos tres componentes principales en la implementación: una implementación básica del Juego de la Vida, un algoritmo genético para la evolución de sus patrones, y un sistema de detección de estructuras conocidas. El desarrollo que seguiremos se basa en los fundamentos teóricos expuestos en el capítulo anterior, utilizando, principalmente, las bibliotecas *Pygame* para la visualización, *NumPy* para las estructuras de datos matriciales, y *Matplotlib* para el análisis gráfico de resultados en Python.

3.1. Implementación del Juego de la Vida Clásico

La primera fase del desarrollo consiste en implementar una versión clásica del Juego de la Vida utilizando Python. Esta implementación servirá de base para incorporar, posteriormente, los algoritmos genéticos y el sistema de detección de patrones.

3.1.1. Configuración del entorno de desarrollo

Para la implementación del autómata celular se utilizan, principalmente, *Pygame* y *NumPy*. *Pygame* proporciona una interfaz gráfica interactiva, mientras que *NumPy* facilita la gestión eficiente de matrices para representar el universo celular:

```
import pygame
import numpy as np
import time
pygame.init()
width, height = 800, 600
screen = pygame.display.set_mode((height, width))
color_bg = 20, 30, 50  # background
screen.fill(color_bg)
```

Código 3.1: Configuración del entorno.

Se genera una ventana cuadrada de 800×800 píxeles con un fondo oscuro. Esta resolución proporciona suficiente espacio para visualizar, claramente, la evolución del autómata celular, manteniendo un rendimiento adecuado incluso en equipos con recursos limitados.

3.1.2. Estructura de datos y representación del universo

El universo del Juego de la Vida se representa mediante una matriz bidimensional de 80×80 celdas, donde cada celda puede tener uno de dos estados: *viva* (1) o *muerta* (0):

```
cellsX, cellsY = 80, 80  # numero de celdas
dimCW = width / cellsX
dimCH = height / cellsY
# Estado de las celulas. Vivas = 1, Muertas = 0
gameState = np.zeros((cellsX, cellsY))
```

Código 3.2: Inicialización del estado del juego.

Esta implementación utiliza una matriz *NumPy* inicializada con ceros (todas las células muertas). Las dimensiones de cada celda (líneas 36-37) se calculan dividiendo el tamaño total de la ventana entre el número de celdas, lo que permite adaptar la visualización a diferentes resoluciones.

Una vez inicializada la matriz, se procede a la configuración de patrones conocidos que demuestran las diferentes dinámicas posibles. Por ejemplo:

```
# Block
gameState[5, 20] = 1
gameState[5, 21] = 1
gameState[6, 20] = 1
gameState[6, 21] = 1
# Blinker
gameState[5, 3] = 1
gameState[5, 4] = 1
gameState[5, 5] = 1
```

Código 3.3: Ejemplos de patrones.

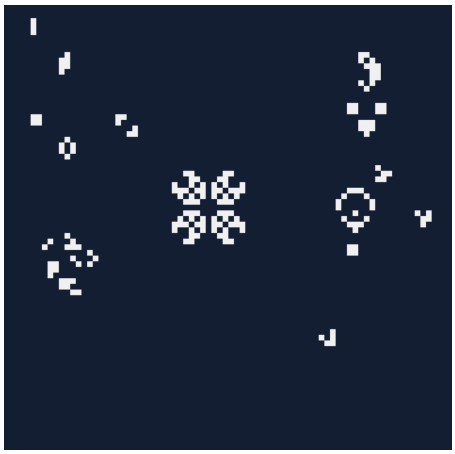


Figura 3.1: Muestra de algunos de los patrones.

3.1.3. Aplicación de las reglas del Juego de la Vida

La parte principal de la implementación consiste en la aplicación de las reglas clásicas del Juego de la Vida, para calcular el estado de cada celda en la siguiente generación:

```
1
2 newGameState = np.copy(gameState)
3
4 for y in range(0, cellsX):
5     for x in range(0, cellsY):
6         if not pause:
7             num_neigh = gameState[(x-1) % cellsX, (y-1) % cellsY] +
                 gameState[(x) % cellsX, (y-1) % cellsY] + # [...]
8
9             if gameState[x, y] == 0 and num_neigh == 3:
10                newGameState[x, y] = 1
11            elif gameState[x, y] == 1 and (num_neigh < 2 or num_neigh
                  > 3):
12                newGameState[x, y] = 0
```

Código 3.4: Implementación de las reglas del Juego de la Vida.

Para cada celda, se calcula el número de celdas vecinas vivas utilizando la operación módulo (%). En la línea 7 se implementan las *condiciones de borde cíclicas*, lo que significa que el universo se "envuelve" sobre sí mismo en los bordes, evitando así efectos no deseados y simulando un universo infinito.

Las reglas implementadas corresponden a las cuatro reglas clásicas del Juego de la Vida:

1. Una célula muerta con exactamente 3 vecinas vivas nace (línea 9).
2. Una célula viva con menos de 2 vecinas vivas muere por soledad (línea 11).
3. Una célula viva con más de 3 vecinas vivas muere por sobrepoblación (línea 11).
4. Una célula viva con 2 o 3 vecinas vivas sobrevive (implícito en la implementación).

Es importante destacar que el cálculo del nuevo estado se realiza sobre una copia de la matriz original (**newGameState**, línea 2), para evitar interferencias durante el proceso de actualización, ya que todas las celdas deben actualizarse simultáneamente.

3.1.4. Sistema de visualización

La visualización del estado del juego se implementa dentro del mismo bucle que calcula las generaciones, renderizando cada célula como un cuadrado en la pantalla:

```
1 # Cuadrado
2 poly = [(x * dimCW, y * dimCH),
3         ((x+1) * dimCW, y * dimCH),
4         ((x+1) * dimCW, (y+1) * dimCH),
5         (x * dimCW, (y+1) * dimCH)]
6
```

```
7  if newGameState[x,y] == 0:
8      pygame.draw.polygon(screen, (20, 30, 50), poly, width=1)
9  else:
10     pygame.draw.polygon(screen, (240, 240, 240), poly, width=0)
```

Las células vivas se representan como cuadrados blancos sólidos, mientras que las células muertas se muestran del mismo color que el fondo, como podemos observar en la **Figura 3.1**.

3.2. Implementación del Algoritmo Genético

La segunda fase del desarrollo de la implementación incorpora algoritmos genéticos para evolucionar patrones iniciales con comportamientos interesantes en el Juego de la Vida [2]. Esta sección detalla cómo se implementa el proceso evolutivo para descubrir configuraciones con propiedades emergentes [4].

3.2.1. Diseño de la población inicial

El proceso evolutivo comienza con la generación de una población inicial de individuos aleatorios:

```
1  # Parametros del Juego de la Vida
2  GRID_SIZE = 30
3  MAX_GENERATIONS = 20
4
5  # Parametros del Algoritmo Genetico
6  POPULATION_SIZE = 30
7  CHROMOSOME_SIZE = GRID_SIZE * GRID_SIZE
8  MUTATION_RATE = 0.01
9  NUM_EVOLUTIONS = 50  # Cuantas veces se ejecuta el algoritmo genetico
10
11 def random_individual():
12     """Crea un individuo aleatorio (una cuadricula de 0s y 1s)."""
13     return np.random.choice([0, 1], size=(GRID_SIZE, GRID_SIZE))
14
15 # Inicializa poblacion
16 population = [random_individual() for _ in range(POPULATION_SIZE)]
```

Código 3.5: Implementación de la población inicial.

Para esta implementación, se utiliza un universo más pequeño (30×30) que en la versión básica, lo que permite una evolución más rápida. Cada individuo se representa mediante una matriz binaria, donde cada elemento corresponde al estado inicial de una célula.

La población inicial consta de 30 individuos (línea 6) generados aleatoriamente, con igual probabilidad de células vivas y muertas. Este enfoque garantiza una amplia diversidad genética desde el principio.

3.2.2. Función de aptitud

La función de aptitud es un componente crítico que determina qué individuos tienen más probabilidades de reproducirse. Se han implementado dos versiones:

- **Función de aptitud básica:** evalúa la supervivencia de células después de simular un número fijo de generaciones:

```
def fitness(individual):
    grid = individual.copy()
    for _ in range(MAX_GENERATIONS):
        grid = game_of_life_step(grid)
return np.sum(grid)
```

- **Función de aptitud avanzada:** evalúa comportamientos más complejos, como el movimiento o la oscilación del patrón:

```
def fitness2(individual):
    grid = individual.copy()
    history = []
    movement_score = 0
    oscillation_score = 0

    last_com = None  # centro de masa anterior

    for gen in range(MAX_GENERATIONS):
        grid = game_of_life_step(grid)
        history.append(grid.copy())

        # Medir movimiento del centro de masa
        y, x = np.nonzero(grid)
        if len(x) > 0:
            com = (np.mean(y), np.mean(x))
            if last_com:
                dy = com[0] - last_com[0]
                dx = com[1] - last_com[1]
                movement_score += np.sqrt(dx**2 + dy**2)
            last_com = com

        # Medir oscilacion: comparando con generaciones
           anteriores
        for prev in history[:-1]:
            if np.array_equal(prev, grid):
                oscillation_score += 1
                break

    # Celulas vivas al final (actividad)
    alive_cells = np.sum(grid)

    # Ponderar cada componente
    score = alive_cells + movement_score * 2 + oscillation_score
        * 5
    return score
```

La función avanzada incorpora tres componentes principales:

- **Supervivencia:** cuántas células permanecen vivas al final de la simulación (línea 30).
- **Movimiento:** calculado como la distancia recorrida por el centro de masa del patrón (líneas 14-21).
- **Oscilación**: detectada cuando un estado futuro es idéntico a un estado anterior (líneas 24-27).

Estas componentes se ponderan para favorecer patrones como los *gliders*, que combinan movimiento y oscilación. La ponderación asigna mayor valor a la oscilación ($\times 5$) que al movimiento ($\times 2$) (línea 33), ya que la oscilación es una característica más distintiva de los patrones complejos.

3.2.3. Operadores genéticos

Los *operadores genéticos* permiten generar nuevos individuos a partir de la población existente. Se han implementado tres operadores clásicos:

1. **Selección:** se seleccionan los individuos con mayor aptitud para reproducirse:

```
# Seleccion: top 50%
selected = [ind for _, ind in scored_population[:POPULATION_SIZE
    // 2]]
```

2. **Cruzamiento (*crossover*):** se combinan dos individuos para generar un descendiente:

```
def crossover(parent1, parent2):
    """Cruza dos padres para generar un hijo."""
    point = random.randint(0, CHROMOSOME_SIZE-1)
    flat1 = parent1.flatten()
    flat2 = parent2.flatten()
    child_flat = np.concatenate((flat1[:point], flat2[point:]))
    return child_flat.reshape((GRID_SIZE, GRID_SIZE))
```

3. **Mutación:** se introducen pequeñas variaciones aleatorias en los nuevos individuos:

```
def mutate(individual):
    """Muta un individuo con cierta probabilidad."""
    mutation_mask = np.random.rand(GRID_SIZE, GRID_SIZE) <
       MUTATION_RATE
    individual[mutation_mask] = 1 - individual[mutation_mask]
    return individual
```

El método de *selección* implementado es *elitista*, conservando el 50 % superior de la población. El *cruzamiento* utiliza un enfoque de *punto único*, donde los cromosomas se dividen en un punto aleatorio. La *mutación* invierte, aproximadamente, el 1 % de las células, según la *tasa de mutación* definida.

El *bucle principal* del algoritmo genético integra estos operadores para evolucionar la población:

```
for generation in range(NUM_EVOLUTIONS):
    # Evaluar
    scored_population = [(fitness2(ind), ind) for ind in population]
    scored_population.sort(reverse=True, key=lambda x: x[0])

    if scored_population[0][0] > best_fitness:
        best_fitness = scored_population[0][0]
        best_individual = scored_population[0][1]

    print(f"Gen {generation}: Mejor fitness = {scored_population
        [0][0]}")

    selected = [ind for _, ind in scored_population[:POPULATION_SIZE
        // 2]]

    # Reproduccion
    new_population = []
    while len(new_population) < POPULATION_SIZE:
        p1, p2 = random.sample(selected, 2)
        child = crossover(p1, p2)
        child = mutate(child)
        new_population.append(child)

    population = new_population
```

Este proceso se repite durante 50 generaciones, mostrando por pantalla el mejor *fitness* de cada una de ellas (línea 10), y manteniendo el registro del mejor individuo encontrado.

3.2.4. Visualización y análisis de resultados evolutivos

Para visualizar los resultados del algoritmo genético, se desarrolló una interfaz gráfica utilizando *Matplotlib*:

```
class Simulation:
    def __init__(self, best_individual):
        self.fig = plt.figure(figsize=(10, 6))
        self.gridspec = plt.GridSpec(3, 2, width_ratios=[1, 1],
            height_ratios=[8, 8, 2], hspace=0.3)

        # Panel izquierdo: Mejor patron inicial
        self.ax_initial = self.fig.add_subplot(self.gridspec[0:2, 0])
        self.ax_initial.set_title("Mejor Patron Inicial", pad=20)
        self.ax_initial.imshow(best_individual, cmap='Greys')
        self.ax_initial.axis('off')

        # Panel derecho: Simulacion en vivo
        self.ax_sim = self.fig.add_subplot(self.gridspec[0:2, 1])
        self.ax_sim.set_title("Simulacion Evolutiva", pad=20)
        self.ax_sim.axis('off')
```

Código 3.6: Clase que implementa la interfaz gráfica.

La *interfaz* se divide en tres áreas principales:

- Un *panel izquierdo* que muestra el mejor patrón inicial encontrado.
- Un *panel derecho* que muestra la evolución de este patrón en tiempo real.
- Un *área inferior* para información textual y controles.

La simulación se implementa mediante el método `update_frame`, que calcula cada paso del Juego de la Vida, y actualiza la visualización:

```
def update_frame(self, frame):
    if self.sim_running:
        self.grid = game_of_life_step(self.grid)
        self.img.set_data(self.grid)
        self.current_step += 1
        self.update_info()
    return [self.img, self.info_text]
```

La animación se gestiona mediante `matplotlib.animation.FuncAnimation`, que llama, periódicamente, a la función `update_frame`.

A continuación, se muestran algunas imágenes del mejor patrón generado por el algoritmo genético (de entre 50 generaciones) en sus distintas fases de evolución, llegando, finalmente, a un estado únicamente compuesto por patrones estáticos, lo que quiere decir que hemos encontrado un universo que perdurará en el tiempo indefinidamente.

Figura 3.2: Estado inicial.

Figura 3.3: Estado tras 30 generaciones.

Figura 3.4: Estado tras 100 generaciones.

Figura 3.5: Estado tras 230 generaciones (última generación antes del equilibrio).

3.3. Implementación del sistema de detección de patrones

La tercera fase del desarrollo de la implementación incorpora un *sistema de detección de patrones* específicos en el Juego de la Vida, permitiendo identificar estructuras conocidas y analizar propiedades emergentes.

3.3.1. Definición de patrones conocidos

Los patrones se definen como matrices binarias en un diccionario que asocia cada tipo con sus posibles configuraciones:

```
KNOWN_PATTERNS = {
    "glider": [
        np.array([ [0,1,0], [0,0,1], [1,1,1] ])
    ],
    "blinker": [
        np.array([ [1,1,1] ]),
        np.array([ [1], [1], [1] ])
    ],
    "block": [
        np.array([ [1,1], [1,1] ])
    ],
    "beehive": [
        np.array([ [0,1,1,0], [1,0,0,1], [0,1,1,0] ]),
        np.array([ [0,1,0], [1,0,1], [1,0,1], [0,1,0] ])
    ]
}
```

Para cada tipo de patrón, se incluyen todas sus posibles orientaciones, con el fin de garantizar una detección robusta, independientemente de la rotación.

3.3.2. Algoritmos de matching y detección

Se implementaron tres funciones principales para la detección de patrones:

1. **Detección básica de patrones:** busca coincidencias exactas entre plantillas y subregiones del *grid*:

```
def match_pattern(grid, pattern):
    """Busca un patron dentro del grid."""
    for i in range(grid.shape[0] - pattern.shape[0] + 1):
        for j in range(grid.shape[1] - pattern.shape[1] + 1):
            subgrid = grid[i:i+pattern.shape[0], j:j+pattern.
               shape[1]]
            if np.array_equal(subgrid, pattern):
                return True
    return False
```

```
9
10 def detect_known_patterns(grid):
11     """Detecta si hay algun patrun conocido en el grid."""
12     matches = []
13     for name, templates in KNOWN_PATTERNS.items():
14         for pat in templates:
15             if match_pattern(grid, pat):
16                 matches.append(name)
17     return matches
```

Código 3.7: Funciones básicas para la detección de patrones.

2. **Detección de osciladores:** identifica patrones que se repiten después de cierto número de generaciones:

```
1 def detect_oscillator_period(history):
2     """Detecta el periodo de oscilacion del patron (si existe).
         """
3     for i in range(1, len(history)):
4         if np.array_equal(history[0], history[i]):
5             return i  # patron se repite tras i generaciones
6     return None  # no hay ciclo detectado
```

Código 3.8: Función para la detección de osciladores.

3. **Detección de** *glider guns***:** identifica estructuras que generan *gliders* periódicamente:

```
1 def detect_glider_gun(history):
2     """Detecta si el patron actua como una glider gun."""
3     if len(history) < 10:
4         return False
5
6     # Requiere que al menos un glider aparezca lejos del centro
7     last = history[-1]
8     glider_detected = False
9
10    for name, templates in KNOWN_PATTERNS.items():
11        if name != "glider":
12            continue
13        for pat in templates:
14            if match_pattern(last, pat):
15                glider_detected = True
16
17    if not glider_detected:
18        return False
19
20    # Verificamos si el patron base (centro) sigue similar
21    center_start = history[0][5:15, 5:15]
22    center_end = history[-1][5:15, 5:15]
23    static_center = np.sum(np.abs(center_start - center_end)) <
         5  # casi sin cambio
24    return static_center and glider_detected
```

Código 3.9: Función para la detección de *glider guns*.

Estos algoritmos utilizan comparaciones directas de matrices para detectar coincidencias exactas. Para detectar osciladores, se compara cada nuevo estado con los estados anteriores, almacenados en un historial. La detección de *glider guns* se basa en dos criterios: la presencia de *gliders* en la periferia y la estabilidad del patrón central.

3.3.3. Integración con la evaluación genética

El sistema de detección de patrones se integra con el algoritmo genético mediante una versión mejorada de la función de aptitud:

```
1  def fitness(individual):
2      """Evalua si el patron se mueve, oscila o se mantiene dinamico.
           """
3      grid = individual.copy()
4      history = []
5      movement_score = 0
6      oscillation_score = 0
7
8      # ... (codigo similar a fitness2) ...
9
10     # BONUS si se detecta un patron especifico
11     detected = detect_known_patterns(grid)
12     if "glider" in detected:
13         score += 100  # gran recompensa
14     elif "blinker" in detected:
15         score += 40
16     elif "block" in detected:
17         score += 20
18
19     # Detectar oscilador de periodo N
20     period = detect_oscillator_period(history)
21     if period and period > 2:
22         score += 50 + period * 2  # recompensa creciente
23
24     # Detectar glider gun
25     if detect_glider_gun(history):
26         score += 300  # recompensa alta
27
28     return score
```

Esta función asigna *bonificaciones* significativas a los patrones más interesantes, especialmente a los *gliders* (100 puntos) (línea 12) y *glider guns* (300 puntos) (línea 25). Los osciladores con períodos mayores también reciben bonificaciones proporcionales a su período (líneas 21-22), incentivando así la evolución de comportamientos complejos.

3.3.4. Visualización de patrones detectados

Para facilitar el análisis visual, se ha implementado una versión mejorada de la interfaz gráfica, que resalta los patrones reconocidos mediante rectángulos rojos:

```
def update_frame(self, frame):

if self.sim_running and self.current_step < self.max_generations:
    self.grid = game_of_life_step(self.grid)
    self.history.append(self.grid.copy())
    self.current_step += 1

    # Dibujar patrones detectados
    for patch in self.patches:
        patch.remove()
    self.patches.clear()

    for name, templates in KNOWN_PATTERNS.items():
        for pat in templates:
            for i in range(self.grid.shape[0] - pat.shape[0] + 1):
                for j in range(self.grid.shape[1] - pat.shape[1] + 1)
                    :
                    if np.array_equal(self.grid[i:i+pat.shape[0], j:j
                        +pat.shape[1]], pat):
                        rect = plt.Rectangle((j-0.5, i-0.5), pat.
                            shape[1], pat.shape[0],
                                             linewidth=1, edgecolor='
                                                red', facecolor='none'
                                                )
                        self.ax_sim.add_patch(rect)
                        self.patches.append(rect)
```

Las líneas 3 a 6 actualizan el estado de la interfaz, ejecutando un paso del Juego de la Vida (`game_of_life_step()`, línea 4), guardando el histórico de estados (`history`, línea 5), e incrementando el contador de generaciones (`current_step`, línea 6).

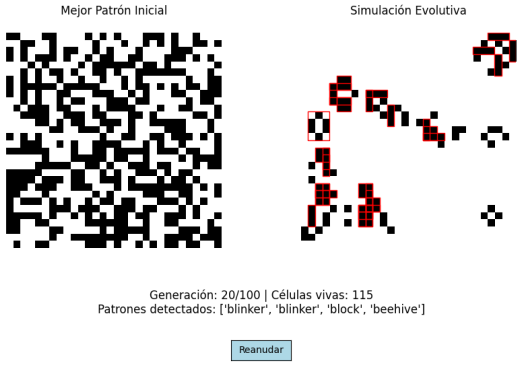
Por ejemplo, si aparece un *glider* en la posición (5,10), se vería un cuadrado rojo alrededor de esa sección 3x3 de la cuadrícula durante el *frame* en que está presente. El sistema puede detectar múltiples instancias de un mismo patrón en diferentes posiciones, y múltiples tipos de patrones, simultáneamente. Además, la interfaz proporciona información textual detallada sobre los patrones detectados y sus propiedades dinámicas:

```
def update_info(self):

    info = (
        f"Generacion: {self.current_step}/{self.max_generations} | "
        f"Celulas vivas: {np.sum(self.grid)}\n"
        f"Patrones detectados: {detect_known_patterns(self.grid)}"
    )

    if self.current_step >= self.max_generations:
        period = detect_oscillator_period(self.history)
        is_gun = detect_glider_gun(self.history)
        patterns_found = detect_known_patterns(final_history[-1])

        info += ("\nAnalisis del mejor patron:")
```

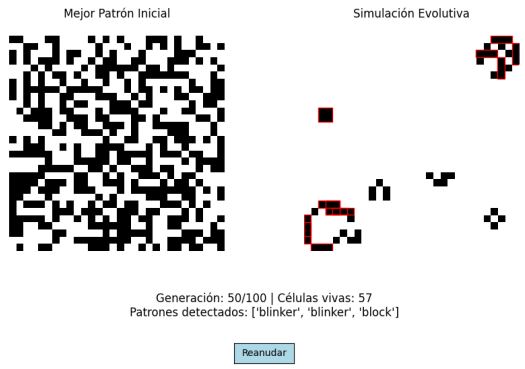
```
15
16          if period:
17              info += (f"-> Oscilador de periodo {period}")
18          if is_gun:
19              info += ("-> Glider gun detectada!")
20          if patterns_found:
21              info += (f"-> Contiene: {', '.join(patterns_found)}")
```

A continuación, se muestran imágenes de cómo va evolucionando el mejor patrón inicial que se ha conseguido de una ejecución de 50 generaciones, junto con los patrones detectados.

(a) Estado inicial.



(b) Estado tras 20 generaciones.



(c) Estado tras 50 generaciones.

(d) Estado tras 77 generaciones (última generación antes del equilibrio).

Figura 3.6: Simulación del mejor patrón inicial.

3.4. Implementación del Aprendizaje Automático

Esta sección detalla la integración de técnicas de *Aprendizaje Automático* (en inglés, *Machine Learning*, ML) para el reconocimiento automático de patrones en el Juego de la Vida. El sistema combina **redes neuronales** con el algoritmo evolutivo previamente implementado, creando un ciclo de retroalimentación donde el modelo ML guía el proceso evolutivo.

3.4.1. Arquitectura del sistema integrado

El sistema sigue una arquitectura híbrida que combina algoritmos evolutivos con redes neuronales profundas. Está formado por tres componentes principales:

- **Extracción de características:** calcula métricas clave de los patrones.
- **Red neuronal:** clasifica patrones y guía la evolución.
- **Interfaz de visualización:** muestra los resultados en tiempo real.

3.4.2. Extracción de características avanzadas

Se implementa una función mejorada de extracción de características que incluye métricas espaciales y dinámicas:

```
def extract_features(grid):
    # Caracteristicas basicas
    alive = np.sum(grid)
    y, x = np.nonzero(grid)
    com_x = np.mean(x) if len(x) > 0 else 0
    com_y = np.mean(y) if len(y) > 0 else 0
    density = alive / (grid.size)

    # Caracteristicas espaciales
    variance_x = np.var(x) if len(x) > 0 else 0
    variance_y = np.var(y) if len(y) > 0 else 0
    symmetry_h = np.sum(grid == np.fliplr(grid))/grid.size
    symmetry_v = np.sum(grid == np.flipud(grid))/grid.size

    # Componentes conectados
    labeled, num_components = label(grid)

    # Caracteristicas dinamicas (requieren simulacion)
    grid_evo = grid.copy()
    stability = 0
    for _ in range(5):
        grid_evo = game_of_life_step(grid_evo)
        stability += np.sum(grid_evo == grid)

    return # array de las caracteristicas
```

Código 3.10: Función para la extracción de características avanzadas.

El bloque de características básicas (líneas 3-7) tiene como propósito cuantificar la distribución de células vivas. Sus parámetros son: `alive`, que guarda el número total de células vivas, `com_x` y `com_y`, para las coordenadas del centro de masa, y `density`, para la densidad de población. Por ejemplo, un *bloque estático* tendrá alta densidad (0,25 en 2x2) y centro de masa fijo.

En las métricas espaciales (líneas 10-13), tenemos `variance_x` y `variance_y` para la dispersión horizontal y la dispersión vertical, respectivamente. En las dos siguientes líneas se encuentran `symmetry_h` para la simetría horizontal, y `symmetry_v` para la simetría vertical; ambas van de 0 a 1. Un par de casos típicos podrían ser:

- *Blinker*: alta simetría vertical/horizontal (1,0 en orientación vertical).
- *Glider*: baja simetría (0,0).

En la línea 16 se define una característica para saber el número de componentes conectados, y, finalmente, se evalúa la estabilidad dinámica de los patrones (líneas 19-23):

- `stability = 1.0` : Patrón estático (ej.: *Block*).
- `stability = 0.5` : Oscilador (ej.: *Blinker*).
- `stability = 0.0` : Patrón caótico/móvil.

Esta implementación permite cuantificar, matemáticamente, patrones cuya complejidad desafía el análisis tradicional, facilitando así su estudio mediante técnicas computacionales avanzadas.

3.4.3. Construcción del conjunto de datos

El *dataset* se genera y estructura, automáticamente, durante la evolución de los algoritmos genéticos, integrando características cuantitativas y etiquetas cualitativas:

```
1  def build_dataset(population_history, steps=20):
2      X, y = [], []
3      all_labels = set()
4
5      for gen in population_history:
6          for individual in gen:
7              history = simulate_evolution(individual, steps)
8              features = extract_features(individual)
9
10             # Deteccion de patrones complejos
11             labels = detect_complex_patterns(history)
12
13             X.append(features)
14             y.append(labels)
15             all_labels.update(labels)
16
17     # Codificacion multi-etiqueta
18     label_mapping = {label: idx for idx, label in enumerate(sorted(
           all_labels))}
```

```
19      y_binary = np.zeros((len(y), len(label_mapping)), dtype=int)
20
21      for i, labels in enumerate(y):
22          for label in labels:
23              y_binary[i, label_mapping[label]] = 1
24
25      return np.array(X), y_binary, label_mapping
```

Código 3.11: Función para la creación del *dataset*.

Para la *recolección de datos* (líneas 3-11), se itera sobre todas las generaciones de la población evolutiva, se simula cada individuo durante 20 generaciones (`steps`), se extraen características estáticas/dinámicas, y se detectan patrones complejos mediante el análisis histórico.

En la *codificación multi-etiqueta* (líneas 18-19) se crea un mapeo único de etiquetas detectadas; un mismo individuo puede recibir múltiples etiquetas simultáneas. Finalmente, se codifica una *matriz de características* y una *matriz de etiquetas*, donde `1` es la *presencia de patrón*, y `0` la *ausencia*.

3.4.4. Modelo de red neuronal profunda

La *red neuronal profunda* (en inglés, *deep neural network*) es la parte esencial del módulo de aprendizaje automático, siendo responsable de la clasificación y reconocimiento automático de los patrones del Juego de la Vida. En este apartado, se realiza un análisis exhaustivo y argumentado de la arquitectura, el flujo de datos, las funciones de activación, la regularización, la función de pérdida, el proceso de entrenamiento, y la integración de la red en el ciclo evolutivo y visual del sistema.

3.4.4.1. Arquitectura de la red neuronal

El modelo sigue una estructura secuencial con una capa de entrada, dos capas ocultas densas con regularización por *dropout*, y una capa de salida con activación *sigmoide*:

```
1  def train_model(X, y_binary):
2      model = Sequential([
3          tf.keras.layers.Input(shape=(X.shape[1],)),
4          Dense(128, activation='relu'),
5          Dropout(0.4),
6          Dense(64, activation='relu'),
7          Dropout(0.3),
8          Dense(y_binary.shape[1], activation='sigmoid')
9      ])
10
11     model.compile(
12         optimizer='adam',
13         loss='binary_crossentropy',
14         metrics=['accuracy']
15     )
```

Código 3.12: Arquitectura del modelo.

La red recibe como entrada un *vector de características* extraídas de la configuración del autómata (ver **Sección 3.4.2**). La arquitectura exacta es la siguiente:

- **Capa de entrada:** 9 neuronas, correspondientes a las 9 características extraídas del patrón (número de células vivas, centro de masa, densidad, varianzas, simetrías, número de componentes conectados).
- **Primera capa oculta:** 128 neuronas densas (*fully connected*) con activación ReLU (línea 3).
- **Dropout:** 40 % de desactivación aleatoria de neuronas en la primera capa oculta (línea 4).
- **Segunda capa oculta:** 64 neuronas densas con activación ReLU (línea 5).
- **Dropout:** 30 % de desactivación aleatoria de neuronas en la segunda capa (línea 6).
- **Capa de salida:** 8 neuronas (una por cada clase de patrón relevante: *glider, blinker, block, beehive, glider_gun, oscillator, extinct, random*), con activación sigmoide para clasificación multi-etiqueta (línea 7).

La elección de dos capas ocultas con 128 y 64 neuronas responde al *principio de reducción progresiva de la dimensionalidad*. Este enfoque permite que la red aprenda representaciones cada vez más abstractas y relevantes para la tarea de clasificación, filtrando así información irrelevante y facilitando la generalización [20].

Todos los términos más técnicos se irán explicando a medida que avanza el capítulo.

3.4.4.2. Flujo de datos

Durante la *predicción*, el flujo de datos sigue una secuencia bien definida:

1. El vector de características de entrada (`batch_size`, 9) es procesado por la primera capa densa, produciendo una representación de tamaño (`batch_size`, 128).

2. Se aplica *dropout*, desactivando, aleatoriamente, el 40 % de las neuronas durante el entrenamiento, lo que fuerza a la red a aprender representaciones redundantes y robustas.

3. La salida pasa a la segunda capa densa, reduciéndose a (`batch_size`, 64), seguida de otro *dropout* del 30 %.

4. Finalmente, la capa de salida transforma la representación a (`batch_size`, 8), donde cada neurona produce una probabilidad independiente de pertenencia a cada clase de patrón mediante la función sigmoide.

Esto nos permite que la red procese, eficientemente, lotes de configuraciones, y produzca predicciones multi-etiqueta en paralelo, aspecto esencial para el análisis de poblaciones evolutivas completas y la visualización dinámica.

3.4.4.3. Funciones de activación y su razonamiento

- **ReLU en capas ocultas:**

  La función de activación ReLU (en inglés, *Rectified Linear Unit*) es la elección estándar en redes profundas, por su eficiencia computacional y su capacidad para mitigar el *problema del gradiente desvanecido* (*vanishing gradient*) [21]. La función ReLU, también llamada *función de activación rectificadora*, toma un valor de entrada x y devuelve:

  $$f(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{si } x \leqslant 0 \end{cases}$$

  Al introducir no-linealidad, sin afectar la dimensionalidad de la representación, permite que la red aprenda funciones complejas y diferenciables, para así capturar la variedad de patrones presentes.

- **Sigmoide en la capa de salida:**

  La función sigmoide transforma cada *logit* (valor escalar que representa la "fuerza" o "evidencia" a favor de una clase particular) de la capa de salida, en una probabilidad independiente. Esta elección es fundamental para la clasificación multi-etiqueta, ya que un mismo patrón puede pertenecer, simultáneamente, a varias clases (por ejemplo, ser un *oscilador* y contener un *bloque*) [22].

3.4.4.4. Regularización: *dropout*

El modelo implementa *dropout* en ambas capas ocultas, con tasas del 40 % y 30 %. *Dropout* es una técnica de regularización que previene el sobreajuste al desactivar, aleatoriamente, una fracción de las neuronas durante el entrenamiento, forzando así a la red a no depender, excesivamente, de ninguna neurona o ruta específica [23]. Esto equivale a entrenar un *ensamble* de redes neuronales más pequeñas y promediar sus predicciones, lo que mejora la capacidad de generalización del modelo.

3.4.4.5. Función de Pérdida: *binary cross-entropy*

La función de pérdida utilizada es la *entropía cruzada binaria* (*binary cross-entropy*) (línea 13 del **Código 3.12**), que es la elección natural para problemas de clasificación multi-etiqueta con salidas sigmoides. Esta función mide la discrepancia entre las probabilidades predichas para cada clase y los valores verdaderos (0 o 1) en cada etiqueta, penalizando fuertemente las predicciones incorrectas con alta confianza.

3.4.4.6. Proceso de optimización y entrenamiento

El modelo se ha entrenado utilizando el optimizador *Adam* (línea 12 del **Código 3.12**) con una tasa de aprendizaje de 0,001. *Adam* es reconocido por su capacidad para adaptarse dinámicamente a los gradientes de cada parámetro, acelerando su convergencia. El entrenamiento se realiza con una partición del 30 % de los datos para validación, lo que permite monitorizar el sobreajuste.

3.4.5. Integración con la interfaz gráfica

Se ha implementado una interfaz gráfica, más avanzada, utilizando `Tkinter` con una estructura de pestañas que permite visualizar diferentes aspectos del sistema:

- **Estructura de múltiples pestañas**

```
self.notebook = ttk.Notebook(root)
self.notebook.pack(expand=True, fill="both", padx=20, pady=20)

self.create_best_tab()
self.create_population_tab()
self.create_fitness_tab()
```

  De esta forma, podemos unificar todas las funcionalidades, permitiendo al usuario: visualizar y animar el mejor patrón encontrado, ver todos los patrones de la población final, y analizar la evolución de *fitness* durante las generaciones.

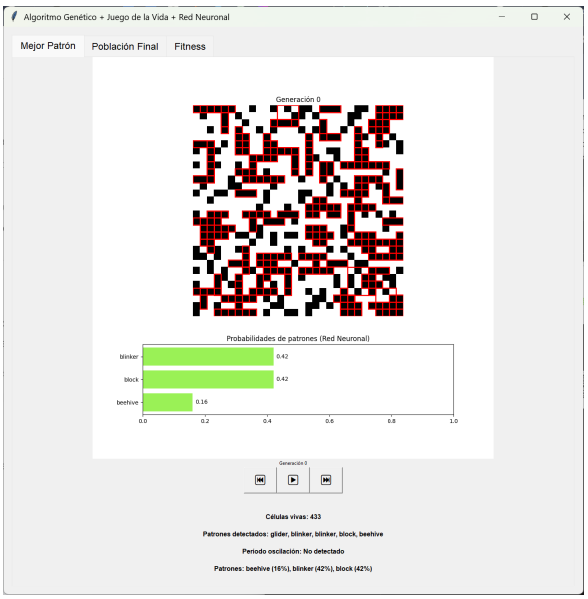
- **Visualización del mejor patrón con análisis en tiempo real**

  Al igual que en anteriores módulos, podemos ver cómo evoluciona el mejor patrón de entre 50 ejecutados en tiempo real, esta vez pudiendo retroceder o avanzar las generaciones de una en una.

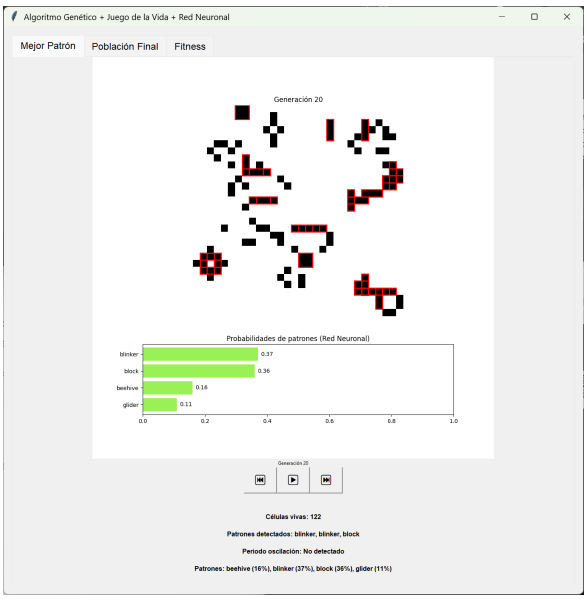
```
def plot_frame(self, idx):
    # Actualizar metricas
    alive = np.sum(grid)
    patterns = detect_known_patterns(grid)
    period = detect_oscillator_period(best_history[:idx+1]) if
       idx > 0 else 0

    # Prediccion con ML
    predictions = predict_pattern(grid, self.model, self.
       label_mapping)

    # Actualizar etiquetas y visualizacion
    self.lbl_cells.config(text=f"Celulas vivas: {alive}")
    # ...
```

  Se añade un gráfico de barras mostrando la clasificación de los patrones presentes en el estado actual. Los porcentajes que muestra el gráfico, reflejan la confianza del modelo en que un patrón está presente.

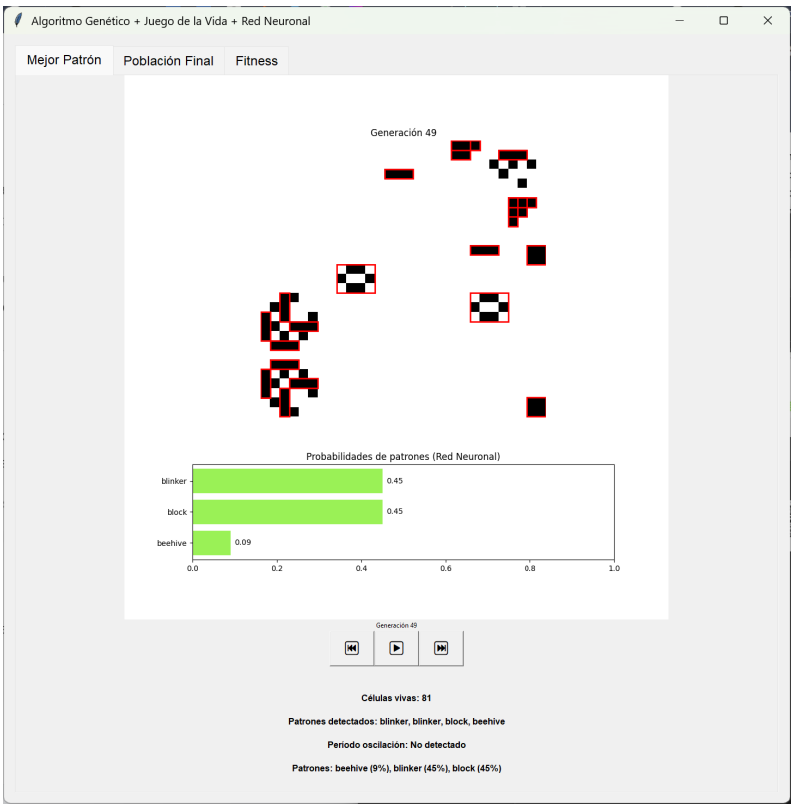
A continuación, se muestran imágenes de cómo va evolucionando el mejor patrón inicial que se ha conseguido de una ejecución de 50 generaciones, junto con los patrones detectados y la clasificación de la red neuronal en ese estado determinado.



(a) Estado inicial.



(b) Estado tras 20 generaciones.



(c) Estado tras 50 generaciones.

Figura 3.7: Mejor patrón inicial.

En la pestaña "Población Final", se muestra la población completa de la última generación, es decir, los 30 patrones que han sobrevivido y han sido seleccionados tras todas las generaciones de evolución.

Figura 3.8: Población final.

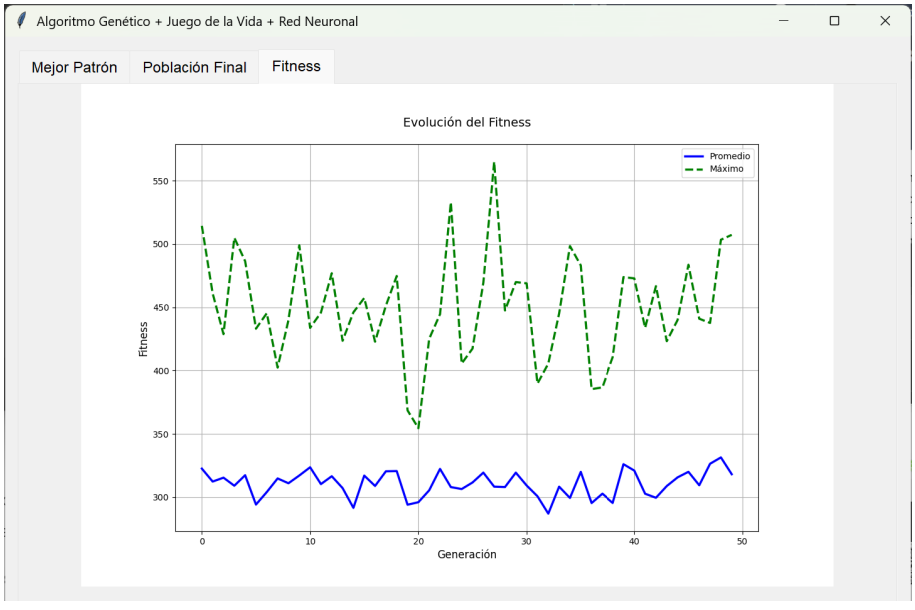


Figura 3.9: Evolución del *fitness*.

Capítulo 4

El Juego de la Vida Extendido

En este capítulo se detalla la metodología empleada para implementar una versión avanzada del Juego de la Vida inspirada en fenómenos astrofísicos. El objetivo principal consiste en expandir el autómata celular clásico presentado en el capítulo anterior hacia un sistema que permita simular interacciones cósmicas complejas, con múltiples tipos de células capaces de representar diferentes cuerpos celestes. A diferencia del modelo tradicional binario (células vivas o muertas), esta implementación introduce siete estados posibles, cada uno con sus propias reglas de comportamiento e interacción gravitacional. La implementación se ha desarrollado utilizando, principalmente, las bibliotecas *NumPy*, para las estructuras de datos matriciales, *Matplotlib* para la visualización interactiva, y técnicas adicionales de programación orientada a objetos para gestionar la complejidad del sistema.

4.1. Juego de la Vida con Reglas Astrofísicas

La primera fase del desarrollo consiste en implementar una versión del Juego de la Vida que incorpore conceptos astrofísicos. Esta implementación servirá como base para integrar, más tarde, algoritmos genéticos que optimicen la emergencia de estructuras cósmicas.

4.1.1. Estructura de datos y representación del Universo

Una diferencia fundamental respecto al modelo clásico es la introducción de múltiples tipos de células, cada una representando un cuerpo o fenómeno astrofísico diferente. En lugar del modelo binario (vivo/muerto), se definen ahora siete estados posibles:

```
TIPOS = {
    0: ('SPACE', 0.0, (0,0,0)),              # Vacio cosmico
    1: ('STAR', 10.0, (1,1,0)),              # Estrella
    2: ('PLANET', 4.0, (0,0.7,0.7)),         # Planeta
    3: ('ASTEROID', 1.0, (0,0,1)),           # Asteroide
    4: ('ENERGY', 0.5, (1,0.2,0)),           # Energia
    5: ('BLACK_HOLE', 30.0, (0.1,0.1,0.4)), # Agujero negro
    6: ('ANTIMATTER', 4.0, (0.3,0.7,1)) }   # Antimateria
```

Código 4.1: Configuración de tipos celulares.

Además, se asignan masas relativas a cada tipo de célula, lo que permitirá implementar interacciones gravitacionales entre ellas. Esta estructura permite modelar fenómenos astronómicos complejos como la formación estelar, la acreción planetaria o los efectos de los agujeros negros, proporcionando un marco más sofisticado que el modelo binario original.

4.1.2. Inicialización del Universo

El Universo se inicializa con una distribución probabilística de los diferentes tipos de células, reflejando la rareza relativa de cada cuerpo celeste:

```
def create_universe():
    return np.random.choice(
        [SPACE, STAR, PLANET, ASTEROID, ENERGY, BLACK_HOLE,
           ANTIMATTER],
        size=(SIZE, SIZE),
        p=[0.77, 0.03, 0.07, 0.07, 0.05, 0.002, 0.008]
    )

def initialize_masses(grid):
    masses = np.zeros_like(grid, dtype=float)
    for t, v in MASSES.items():
        masses[grid == t] = v
    return masses
```

El Universo se genera, aleatoriamente, con una alta predominancia del espacio vacío (77 %), mientras que los cuerpos celestes más masivos, como los agujeros negros, son extremadamente raros (0,2 %). Esta distribución intenta modelar la baja densidad media del Universo observable.

Adicionalmente, se crea una matriz paralela `masses` que almacena la masa actual de cada célula, permitiendo que estos valores evolucionen, de forma independiente, a lo largo de la simulación.

4.1.3. Implementación de la física del sistema

La segunda fase del desarrollo incorpora reglas físicas inspiradas en fenómenos astronómicos reales. Estas reglas determinan cómo interactúan los diferentes cuerpos celestes, y cómo evolucionan a lo largo del tiempo.

4.1.3.1. Cálculo de gravedad local

Una característica distintiva de esta implementación es la incorporación de efectos gravitacionales locales. Cada célula ejerce una influencia gravitacional proporcional a su masa sobre las células vecinas:

```
def local_gravity(grid, x, y):
    # Calcula la fuerza gravitatoria local en la celda (x, y)
    radius = 2
    fx, fy = 0.0, 0.0
```

```
5       for i in range(max(0, x-radius), min(SIZE, x+radius+1)):
6           for j in range(max(0, y-radius), min(SIZE, y+radius+1)):
7               if i == x and j == y:
8                   continue
9               mass = MASSES[grid[i, j]]
10              dx, dy = i - x, j - y
11              dist = np.sqrt(dx**2 + dy**2) + 0.1
12              f = mass / (dist**2)
13              fx += f * dx / dist
14              fy += f * dy / dist
15      return fx, fy
```

Esta función calcula la fuerza gravitacional resultante sobre una célula, simulando una versión simplificada de la *Ley de Gravitación Universal* de Newton: la fuerza es inversamente proporcional al cuadrado de la distancia (línea 11), y se suma, vectorialmente, para obtener la dirección resultante.

El radio de influencia se limita a 2 celdas adyacentes (línea 3) para mantener el carácter local de las interacciones. La pequeña constante de 0,1 que se suma a la distancia (línea 11) evita divisiones por cero, y suaviza los efectos gravitacionales a distancias muy cortas.

4.1.3.2. Reglas de evolución del Universo

El núcleo de la simulación reside en la función `update_universe`, que aplica todas las reglas de interacción y evolución para cada célula:

```
1  def update_universe(grid, masses):
2      # Aplica las reglas de evolucion para cada celda
3      new_grid = grid.copy()
4      new_masses = masses.copy()
5
6      for x in range(SIZE):
7          for y in range(SIZE):
8              cell = grid[x, y]
9              mass = masses[x, y]
10             neighbors = grid[max(0, x-1):x+2, max(0, y-1):y+2]
11             neighbor_masses = masses[max(0, x-1):x+2, max(0, y-1):y
                   +2]
12
13             # Fusion de estructuras del mismo tipo
14             same_type = (neighbors == cell)
15             if cell != SPACE and np.sum(same_type) > 1:
16                 total_mass = np.sum(neighbor_masses[same_type])
17                 new_masses[x, y] = total_mass
18
19
20             # Aniquilacion materia-antimateria
21             if cell != SPACE and ANTIMATTER in neighbors:
22                 new_grid[x, y] = ENERGY
23                 new_masses[x, y] = MASSES[ENERGY]
24                 continue
```

```
25
26              if cell == ANTIMATTER and np.any((neighbors > SPACE) & (
                    neighbors != ANTIMATTER)):
27                  new_grid[x, y] = ENERGY
28                  new_masses[x, y] = MASSES[ENERGY]
29                  continue
```

Las primeras reglas implementan dos fenómenos físicos fundamentales:

1. **Fusión de estructuras del mismo tipo** (líneas 14-17): cuando células del mismo tipo se agrupan, aumentan su masa acumulada, simulando procesos de *acreción* (crecimiento de un objeto, estructura o cuerpo por adición de materia desde el exterior).

2. **Aniquilación materia-antimateria** (líneas 21-29): cuando la materia (cualquier célula no vacía) entra en contacto con antimateria, ambas se convierten en energía, modelando la aniquilación de partículas en física de partículas.

El resto de la función implementa reglas específicas para cada tipo de célula:

```
1
2  # Reglas para cada tipo
3  if cell == SPACE:
4      # Energia puede formar asteroides
5      if np.count_nonzero(neighbors == ENERGY) >= 3 and np.random.rand
           () < 0.02:
6          new_grid[x, y] = ASTEROID
7          new_masses[x, y] = MASSES[ASTEROID]
8
9      # Formacion de asteroides por energia y otros asteroides
10     if np.count_nonzero(neighbors == ENERGY) >= 2 and np.
           count_nonzero(neighbors == ASTEROID) >= 2 and np.random.rand()
            < 0.01:
11         new_grid[x, y] = ASTEROID
12         new_masses[x, y] = MASSES[ASTEROID]
13
14     # Formacion de planetas a partir de asteroides
15     if np.count_nonzero(neighbors == ASTEROID) >= 4 and np.random.
           rand() < 0.01:
16         new_grid[x, y] = PLANET
17         new_masses[x, y] = MASSES[PLANET]
18
19     # Formacion de estrellas a partir de planetas
20     if np.count_nonzero(neighbors == PLANET) >= 4 and np.random.rand
           () < 0.01:
21         new_grid[x, y] = STAR
22         new_masses[x, y] = MASSES[STAR]
23
24     # Formacion de agujero negro por colapso estelar
25     if np.count_nonzero(neighbors == STAR) >= 5 and np.random.rand()
           < 0.005:
26         new_grid[x, y] = BLACK_HOLE
```

```
27          new_masses[x, y] = MASSES[BLACK_HOLE]
28
29      # Emision de energia por estrellas
30      if np.count_nonzero(neighbors == STAR) >= 2 and np.random.rand()
           < 0.01:
31          new_grid[x, y] = ENERGY
32          new_masses[x, y] = MASSES[ENERGY]
33
34      # Colision de agujeros negros
35      if np.count_nonzero(neighbors == BLACK_HOLE) >= 2 and np.random.
           rand() < 0.01:
36          new_grid[x, y] = BLACK_HOLE
37          new_masses[x, y] = MASSES[BLACK_HOLE]
```

Las reglas para el espacio vacío modelan procesos de formación astrofísica [24]:

- **Acumulación de energía** (y otros asteroides) formando asteroides (líneas 5-12).
- **Acreción** de asteroides formando planetas (líneas 15-17).
- **Colapso gravitacional** de planetas formando estrellas (líneas 20-22).
- **Colapso de estrellas masivas** formando agujeros negros (líneas 27-29).
- **Emisión de energía** a partir de estrellas (líneas 32-34).
- **Fusión de agujeros negros** (líneas 37-39).

Estas reglas se aplican con probabilidades bajas, simulando la rareza de estos eventos en sistemas astronómicos reales.

4.1.3.3. Comportamientos específicos de cuerpos celestes

Cada tipo de cuerpo celeste tiene comportamientos propios, según varios fenómenos astrofísicos reales:

```
1  elif cell == STAR:
2      # Colapso a agujero negro si la masa es grande
3      if mass > 25 and np.random.rand() < 0.01:
4          new_grid[x, y] = BLACK_HOLE
5          new_masses[x, y] = MASSES[BLACK_HOLE]
6
7      # Las estrellas emiten energia
8      if np.random.rand() < 0.02:
9          for i in range(max(0, x-1), min(SIZE, x+2)):
10             for j in range(max(0, y-1), min(SIZE, y+2)):
11                 if grid[i, j] == SPACE and np.random.rand() < 0.2:
12                     new_grid[i, j] = ENERGY
13                     new_masses[i, j] = MASSES[ENERGY]
14
15 elif cell == PLANET:
16     # El planeta puede convertirse en estrella si la masa es grande
17     if mass > 10 and np.random.rand() < 0.01:
18         new_grid[x, y] = STAR
19         new_masses[x, y] = MASSES[STAR]
```

```
20
21      # El planeta puede fragmentarse en asteroides
22      if np.random.rand() < 0.001:
23          new_grid[x, y] = ASTEROID
24          new_masses[x, y] = MASSES[ASTEROID]
25
26  elif cell == ASTEROID:
27      # El asteroide se dispersa
28      if np.random.rand() < 0.002:
29          new_grid[x, y] = SPACE
30          new_masses[x, y] = 0
31
32  elif cell == ENERGY:
33      # La energia puede formar asteroides
34      if np.count_nonzero(neighbors == ENERGY) >= 3 and np.random.rand
           () < 0.01:
35          new_grid[x, y] = ASTEROID
36          new_masses[x, y] = MASSES[ASTEROID]
37
38      # La energia se disipa
39      if np.random.rand() < 0.01:
40          new_grid[x, y] = SPACE
41          new_masses[x, y] = 0
```

Las estrellas actúan como fuentes de energía que irradian a su entorno (líneas 8-13), y pueden colapsar en agujeros negros si alcanzan suficiente masa (líneas 3-5). Los planetas pueden evolucionar hacia estrellas (núcleo fusión) o fragmentarse en asteroides (colisiones). La energía puede condensarse en materia (asteroides) o disiparse en el espacio.

4.1.3.4. Dinámica de agujeros negros

Los agujeros negros reciben un tratamiento especial, modelando, tanto sus propiedades de absorción como la *evaporación* o *radiación de Hawking* [25], proceso en el que los agujeros negros tienden a perder energía y masa mediante la emisión de partículas, lo que conlleva, en ocasiones, a su *desintegración*.

```
1   elif cell == BLACK_HOLE:
2       # Perdida de masa gradual
3       if np.random.rand() < PROB_MASS_LOSS:
4           mass_lost = mass * MASS_LOSS_RATE
5           new_masses[x, y] -= mass_lost
6
7       # El agujero negro absorbe objetos cercanos
8       for i in range(max(0, x-1), min(SIZE, x+2)):
9           for j in range(max(0, y-1), min(SIZE, y+2)):
10              if (i != x or j != y) and grid[i, j] not in [SPACE,
                   BLACK_HOLE]:
11                  if np.random.rand() < 0.5:
12                      new_grid[i, j] = SPACE
13                      new_masses[i, j] = 0
14
```

```
    # El agujero negro puede emitir energia (radiacion Hawking)
    if np.random.rand() < 0.001:
        for i in range(max(0, x-1), min(SIZE, x+2)):
            for j in range(max(0, y-1), min(SIZE, y+2)):
                if grid[i, j] == SPACE and np.random.rand() < 0.2:
                    new_grid[i, j] = ENERGY
                    new_masses[i, j] = MASSES[ENERGY]

    # Desintegracion final
    if new_masses[x, y] < 1:
        # Liberar energia residual
        remaining_energy = new_masses[x, y]
        energy_per_cell = remaining_energy / 8
        for dx in [-1, 0, 1]:
            for dy in [-1, 0, 1]:
                if dx == 0 and dy == 0:
                    continue
                nx, ny = x + dx, y + dy
                if 0 <= nx < SIZE and 0 <= ny < SIZE:
                    if new_grid[nx, ny] == SPACE:
                        new_grid[nx, ny] = ENERGY
                        new_masses[nx, ny] = energy_per_cell
                    elif new_grid[nx, ny] == ENERGY:
                        new_masses[nx, ny] += energy_per_cell

        new_grid[x, y] = SPACE
        new_masses[x, y] = 0
```

Esta implementación modela cuatro aspectos clave de los agujeros negros:

1. **Pérdida gradual de masa** (líneas 3-5), simulando la evaporación de Hawking.
2. **Absorción de materia circundante** (líneas 8-13), capturando objetos cercanos.
3. **Emisión ocasional de energía** (líneas 16-21), modelando la radiación de Hawking.
4. **Desintegración final** (líneas 24-38), cuando su masa desciende por debajo de un umbral.

La **desintegración final** se implementa distribuyendo la energía residual del agujero negro entre las celdas vecinas, simulando la explosión final teórica que ocurriría al final de la vida de un agujero negro.

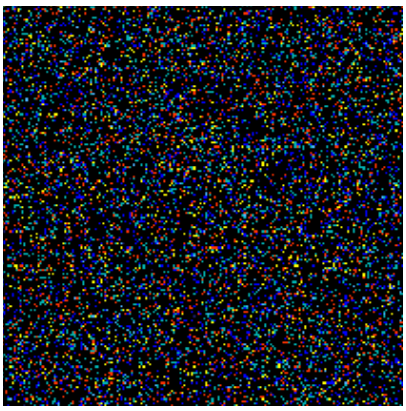
4.1.4. Sistema de visualización

La tercera fase de la implementación se centra en el desarrollo de una interfaz gráfica interactiva para visualizar y controlar la simulación. La interfaz es muy parecida al modelo clásico, por lo que solo se detallarán las diferencias significativas. La representación visual del universo se implementa mediante una función que transforma la matriz de estados en una imagen RGB:

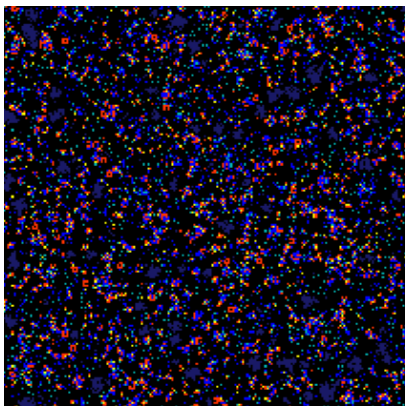
```python
def render(grid):   # Convierte el grid en una imagen RGB
    img = np.zeros((SIZE, SIZE, 3))
    for t, color in COLORS.items():
        return img[grid == t] = color
def count_cells(grid):  # Cuenta los tipos de celda
    counts = Counter(grid.flatten())
    return [counts.get(t, 0) for t in range(7)]
```

La función `render` asigna, a cada tipo de célula, un color distintivo definido en el diccionario `COLORS`. Los agujeros negros se representan en azul oscuro, las estrellas en amarillo, los planetas en turquesa, los asteroides en azul, y la energía en rojo. La función `count_cells` contabiliza la cantidad de cada tipo de célula en el Universo, información que se mostrará en el panel de estadísticas.

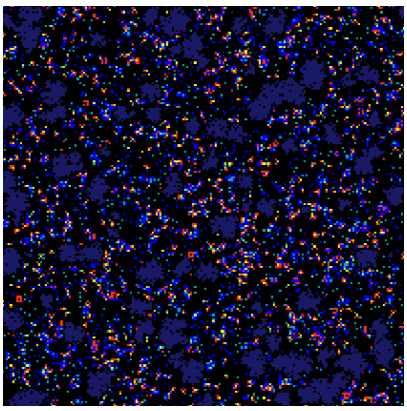
A continuación (**Figura 4.1**), se muestra una secuencia de imágenes que ilustran la dinámica evolutiva del sistema simulado. En ellas, se puede observar cómo los agujeros negros de diferentes puntos del Universo incrementan, progresivamente, su masa, gracias a la absorción de la materia de su alrededor. Este crecimiento continuo los lleva a "dominar", de manera más marcada, el entorno. El estado capturado en la última imagen evidencia que los agujeros negros han alcanzado tal magnitud, que han llegado a ocupar la totalidad del 'marco del Universo' observable.



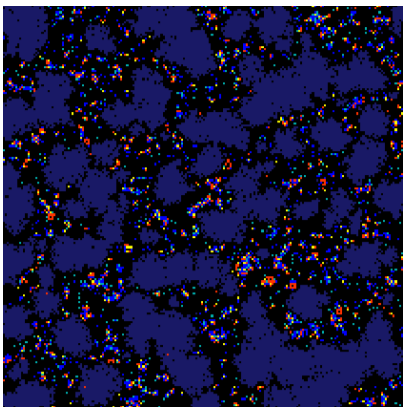
(a) Estado inicial.



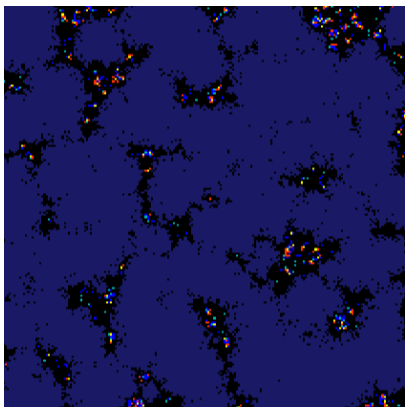
(b) 300 generaciones.



(c) 500 generaciones.



(d) 800 generaciones.



(e) 1.200 generaciones.

(f) 2.860 generaciones.

Figura 4.1: Evolución del universo simulado.

4.2. Incorporación de Algoritmos Genéticos

La integración de algoritmos genéticos en el nuevo diseño del autómata celular representa un avance fundamental para conseguir mejores resultados, a partir de configuraciones iniciales aleatorias. Así, este sistema integrado nos permite combinar las técnicas de computación evolutiva con dinámicas celulares concretas, inspiradas en procesos cósmicos.

4.2.1. Diseño del sistema evolutivo

El sistema evolutivo implementado se estructura en tres capas fundamentales, cada una de ellas con funciones específicas, pero interconectadas:

1. **Capa celular:** implementa las reglas y dinámicas del autómata celular extendido con los siete tipos de células cósmicas (espacio vacío, estrellas, planetas, asteroides, energía, agujeros negros y antimateria). Esta capa es responsable de simular los procesos físicos locales que controlan la evolución del Universo en cada paso de tiempo.
2. **Capa genética:** aplica los operadores evolutivos (selección, cruce, mutación) a las diferentes configuraciones del Universo, mejorando, progresivamente, hacia formaciones cósmicas específicas.
3. **Capa de interfaz:** traduce las matrices numéricas en representaciones visuales interactivas, permitiendo observar y analizar la evolución de las estructuras cósmicas y el progreso del algoritmo genético.

4.2.2. Configuración genética

La configuración del algoritmo genético requiere un equilibrio entre *exploración* y *explotación* del espacio de búsqueda. Los parámetros principales se definen de la siguiente forma:

```
POP_SIZE = 10 # Tamano de la poblacion
SIM_STEPS = 20 # Numero de pasos para evaluar cada universo
MUTATION_RATE = 0.01 # Tasa de mutacion
GEN_STEPS = 20  # Generaciones evolutivas
```

La elección de estos valores equilibra diversidad genética y coste computacional; en el siguiente capítulo veremos cómo ajustar estos parámetros con el objetivo de comparar y conseguir el resultado óptimo.

4.2.2.1. Función de aptitud

La función de aptitud (*fitness*) es el componente más crítico del algoritmo genético, pues define qué características del Universo consideramos "deseables". La implementación evalúa múltiples aspectos del estado final del Universo después de la simulación:

```
def fitness(final_grid, final_masses):
```

```
2     total_mass = np.sum(final_masses)
3     stars = np.sum(final_grid == STAR)
4     planets = np.sum(final_grid == PLANET)
5     return total_mass + 10 * (stars + planets)
```

La masa total (`total_mass`) indica la acumulación de materia en el Universo; a esta se le suma la cantidad de estrellas y planetas que hay (líneas 4-5) para beneficiar la aparición de sistemas estelares (capaces de crear "vida").

4.2.2.2. Operadores genéticos

A continuación, se detallan las componentes (u operadores) que conforman el algoritmo genético, encargados de hacer evolucionar la población hacia soluciones óptimas. Cada una de ellas cumple un papel específico en el proceso evolutivo:

- **Selección**

  La función de selección establece una forma elitista donde los universos con mayor *fitness* tienen más probabilidad de reproducirse, y, los peores, se descartan.

```
1 sorted_indices = np.argsort(scores)[::-1]
2 selected = [population[i] for i in sorted_indices[:POP_SIZE//2]]
3
4 while len(new_pop) < POP_SIZE:
5     p1, p2 = random.sample(selected, 2)
6     child = crossover(p1, p2)
7     child = mutate(child)
8     new_pop.append(child)
```

  Después de la selección, se eligen dos padres al azar entre los mejores (línea 5), se cruzan para formar un hijo, pasa por el proceso de mutación, y, finalmente, se añade a la nueva población.

- **Cruce**

  La función de cruce mantiene los bloques espaciales coherentes mediante la división por *punto de cruce aleatorio*, donde se elige una fila al azar dentro del rango del Universo (sin incluir los bordes) como punto de corte (línea 2), y después se apilan verticalmente para formar el hijo.

```
1 def crossover(parent1, parent2):
2     crossover_point = np.random.randint(1, SIZE-1)
3     child = np.vstack((parent1[:crossover_point], parent2[
          crossover_point:]))
4     return child
```

- **Mutación**

  La función de mutación nos permite introducir en los hijos una variabilidad controlada, respetando la distribución inicial del Universo.

```
1 def mutate(universe):
2     mutation_mask = np.random.rand(SIZE, SIZE) < MUTATION_RATE
3     random_values = np.random.randint(0, 7, size=(SIZE, SIZE))
4     return np.where(mutation_mask, random_values, universe)
```

4.2.3. Ciclo evolutivo

El ciclo evolutivo integra todas las componentes anteriores en un proceso iterativo que optimiza, gradualmente, la población:

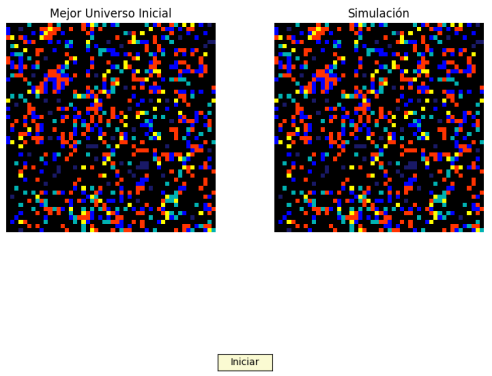
```
1  def evolve_population(population):
2      for _ in range(GEN_STEPS):
3          # Evaluacion
4          scores = [simular_y_evaluar(universe) for universe in
               population]
5          # Seleccion elitista
6          selected = [population[i] for i in sorted_indices[:POP_SIZE
               //2]]
7          # Reproduccion
8          new_pop = [crossover(*random.sample(selected, 2)) for _ in
               range(POP_SIZE)]
9          # Mutacion
10         population = [mutate(child) for child in new_pop]
11         return mejor_universo
```

Este flujo implementa una estrategia evolutiva en la que se preserva el mejor individuo encontrado durante todas las generaciones (líneas 6–11). Utiliza selección *elitista* (manteniendo la mitad superior de la población ordenada por *fitness*) y reproducción mediante cruce aleatorio entre los padres seleccionados, seguido de mutación por celda.

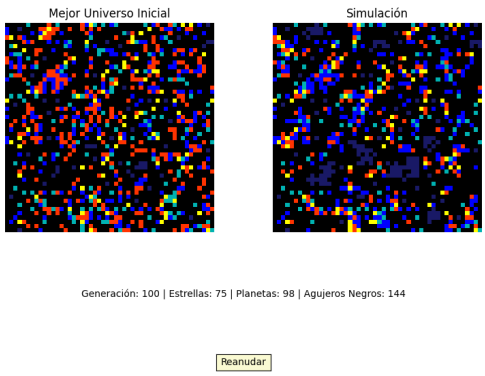
Además, el ciclo evolutivo está acoplado con la simulación del Universo: cada individuo es evaluado tras una simulación de `SIM_STEPS` pasos, y este proceso se repite durante `GEN_STEPS` generaciones. Esta estructura de doble escala temporal —una interna de simulación celular y otra externa de evolución genética— permite explorar configuraciones iniciales que generan dinámicas complejas a medio plazo.

4.2.4. Visualización de resultados

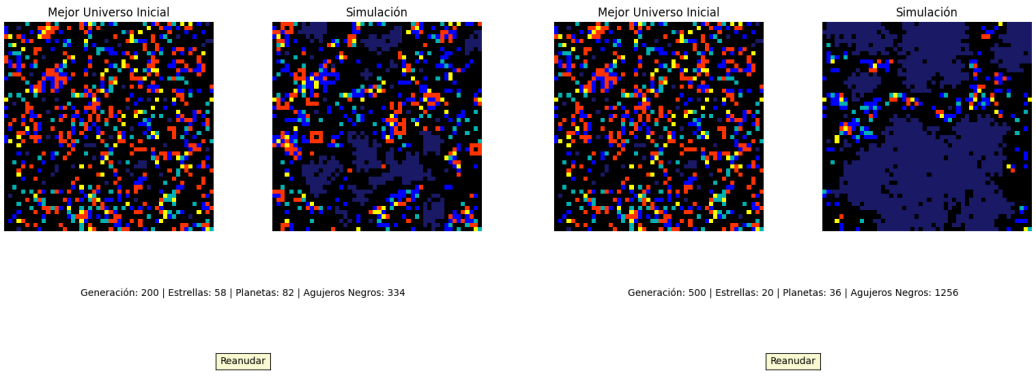
Para analizar los resultados, se implementa una interfaz de visualización que permite ver el estado inicial del mejor Universo encontrado. Además, se incluye una animación que simula la evolución del mejor Universo identificado.



(a) Estado inicial.



(b) Estado tras 100 iteraciones.



(c) Estado tras 200 iteraciones. (d) Estado tras 500 iteraciones.

Figura 4.2: Simulación del mejor universo evolutivo.

4.3. Integración del Aprendizaje Automático

La incorporación de técnicas de aprendizaje automático en el diseño del autómata celular extendido, representa un cambio importante a la hora de analizar la emergencia de estructuras cósmicas. Este nuevo sistema integrado nos va a permitir combinar la potencia de las redes neuronales con las dinámicas celulares inspiradas en procesos astrofísicos, permitiendo así una clasificación automática de los patrones identificados durante su evolución.

4.3.1. Arquitectura de la red neuronal

El módulo de ML se estructura en tres componentes principales, interconectadas:

```
1  class DataCollector:
2      def __init__(self, max_samples=500):
3          self.images = []
4          self.stats = []
5  class PatternDetector:
6      def __init__(self, latent_dim=64, n_clusters=8):
7          self.autoencoder, self.encoder = build_autoencoder(...)
8          self.kmeans = KMeans(n_clusters=n_clusters)
9  def build_autoencoder(input_shape=(20, 20, 3), latent_dim=64):
10     # Capas del encoder y decoder
11     return autoencoder, encoder
```

Estas componentes definen un flujo de datos que sigue tres pasos fundamentales para la detección de las estructuras:

1. **Recolección de muestras:** el `DataCollector` (líneas 2-4) actúa como memoria a corto plazo del sistema, almacenando 500 configuraciones consecutivas del Universo en dos formatos complementarios:
   - **Representaciones visuales:** matrices RGB de 200x200 píxeles que codifican los tipos celulares.
   - **Estadísticas estructurales:** 35 parámetros numéricos que cuantifican distribuciones globales y regionales de los elementos cósmicos.

2. **Codificación automática:** el *autoencoder* reduce las imágenes 200x200x3 a vectores de 64 dimensiones.

3. **Agrupamiento espacial:** *K-Means* clasifica las muestras en 8 *clusters* basados en características combinadas (latentes + estadísticas).

4.3.2. Extracción de características

El proceso de transformación de datos crudos a representaciones significativas involucra tres niveles de abstracción:

```
def _extract_statistics(self, grid, masses):
    # 1. Distribucion global de tipos celulares
    counts = [np.sum(grid == i) for i in range(7)]

    # 2. Analisis regional jerarquico
    region_stats = []
    for i in range(regions):
        for j in range(regions):
            region = grid[i*region_size:..., j*region_size:...]
            region_counts = [np.sum(region == k) for k in range(7)]
            region_stats.extend(region_counts)

    # 3. Relaciones masa-tipo por sector
    mass_density = calculate_mass_distribution(grid, masses)

    return np.concatenate([counts, region_stats, mass_density])
```

Esta implementación captura, tanto *propiedades macroscópicas* (distribución global de estrellas, planetas, etc.) como *patrones locales* (agrupamientos en cuadrantes específicos) y *relaciones físico-energéticas* (densidad masa-tipo por región).

4.3.3. Modelo de autoencoder convolucional

El *autoencoder* utiliza una arquitectura simétrica con 'cuello de botella' para aprendizaje no supervisado:

```
# Encoder
x = Conv2D(16, (3,3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2,2), padding='same')(x)
x = Conv2D(8, (3,3), activation='relu', padding='same')(x)
encoded = Dense(latent_dim, activation='relu')(x)

# Decoder
x = Dense(np.prod(shape_before_flatten[1:]), activation='relu')(
    encoded)
x = Reshape(shape_before_flatten[1:])(x)
x = Conv2D(16, (3,3), activation='relu', padding='same')(x)
decoded = Conv2D(3, (3,3), activation='sigmoid', padding='same')(x)
```

El **encoder** (*compresión*) utiliza capas convolucionales (`Conv2D`) que extraen características espaciales locales usando la *función de activación* 'ReLU'. Como se ha explicado en capítulos anteriores, en una red neuronal, la función de activación es la operación matemática que se aplica a la salida de cada neurona antes de pasarla a la siguiente capa.

En este caso, la función de activación convierte todos los valores negativos a cero, dejando los positivos igual; de esta manera, introduce no-linealidad en el modelo, y hace que el entrenamiento sea más eficiente. La instrucción `padding='same'` mantiene el tamaño de la imagen tras la convolución, preservando así su información. La capa densa final proyecta a un espacio latente de 64 dimensiones (línea 5).

Por otro lado, el **decoder** (*reconstrucción*) empieza con una capa densa que expande el vector latente. Hace uso de `Reshape` para restaurar la estructura, y de `'sigmoid'`, la cual normaliza la salida entre 0 y 1; finalmente, reconstruye la imagen original.

4.3.4. Entrenamiento adaptativo del modelo

La fase de entrenamiento adaptativo del modelo es el proceso mediante el cual, la red neuronal convolucional (*autoencoder*) y el sistema de agrupamiento (*K-Means*), aprenden a identificar, comprimir y clasificar patrones complejos en las configuraciones del Universo simulado:

```
1  def train(self, images, stats):
2      # 1. Entrenamiento del autoencoder
3      self.autoencoder.fit(images, images, epochs=10, batch_size=8)
4
5      # 2. Extraccion de caracteristicas latentes
6      latent_vectors = self.encoder.predict(images)
7
8      # 3. Agrupamiento espacial
9      combined_features = np.concatenate([latent_vectors, stats], axis
           =1)
10     self.kmeans.fit(combined_features)
```

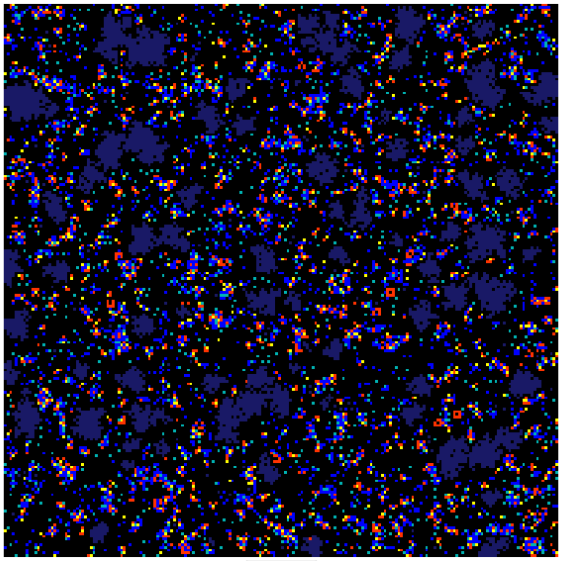
El modelo se entrena con 50 muestras representativas, suficientes para capturar la diversidad del sistema. Su objetivo principal es que el *encoder* aprenda una representación compacta (vector latente) de cada Universo, donde se preserven los patrones espaciales y estructurales más importantes.

4.3.5. Integración con la interfaz de visualización

El sistema de monitorización se amplía con un panel analítico en tiempo real:

```
table_data = [
    ["Generacion", "0"],
    ["Sistemas Estelares", "0"],
    ["Regiones Energia", "0"],
    ["Clusters Agujeros", "0"],
    ["Vacios Cosmicos", "0"]]
```

De esta manera, podemos llevar a cabo un seguimiento de las estructuras clave, y asociarlas con las métricas cuantitativas. A continuación, se muestra una representación gráfica (**Figura 4.3**) que permite observar la cantidad de patrones detectados de cada tipo en un Universo, después de 500 generaciones.



Generación	500
Sistemas Estelares	694
Regiones Energía	236
Clusters Agujeros	2405
Vacíos Cósmicos	5601

Figura 4.3: Clasificación de patrones de la red neuronal.

Capítulo 5

Resultados y Análisis

El presente capítulo expone los resultados de ejecución obtenidos tras implementar el modelo del Juego de la Vida extendido con reglas astrofísicas, descrito en el capítulo anterior. Se realiza un análisis sistemático de cómo los distintos parámetros de configuración, y sus reglas, afectan a la evolución del sistema y a su rendimiento computacional. Los experimentos realizados buscan comprender mejor la dinámica del sistema, y optimizar sus configuraciones iniciales para lograr comportamientos específicos, como la formación de sistemas estelares estables o la evolución controlada de agujeros negros.

5.1. Configuración Experimental

Para garantizar la replicabilidad y reproducibilidad de los experimentos realizados, se ha empleado la siguiente configuración hardware y software:

- Procesador Intel Core i7-12700H @ 2.30GHz
- Memoria RAM 16 GB DDR5
- Sistema operativo Windows 11 Pro
- Python 3.11.4
- NumPy 1.24.3
- Matplotlib 3.10.3

Todos los experimentos se realizaron con un tamaño de Universo estándar de 200×200 celdas, salvo en las pruebas de escalabilidad, donde estos tamaños se indican explícitamente. Las simulaciones básicas se ejecutaron durante 750 generaciones, mientras que los experimentos con algoritmos genéticos utilizaron 20 generaciones evolutivas con 20 pasos de simulación por cada individuo.

5.1.1. Parámetros iniciales

La **configuración base** utilizada en los experimentos realizados mantiene los siguientes valores para los parámetros fundamentales:

```
SIZE = 200  # Tamano del universo
PROB_MASS_LOSS = 0.15  # Perdida de masa de agujeros negros
MASS_LOSS_RATE = 0.12  # Tasa de perdida de masa
```

Para los tipos de cuerpos celestes, se han utilizado las siguientes masas relativas:

```
MASSES = {
    SPACE: 0.0,        # Vacio
    STAR: 10.0,        # Estrella
    PLANET: 4.0,       # Planeta
    ASTEROID: 1.0,     # Asteroide
    ENERGY: 0.5,       # Energia
    BLACK_HOLE: 30.0, # Agujero negro
    ANTIMATTER: 4.0    # Antimateria
}
```

La distribución inicial de elementos sigue la siguiente configuración probabilística:

```
p=[0.77, 0.03, 0.07, 0.07, 0.05, 0.002, 0.008]  # Espacio, Estrella,
   Planeta, Asteroide, Energia, Agujero negro, Antimateria
```

Esta configuración establece la *base* para todos los experimentos que se presentan a continuación, modificándose solo los parámetros indicados, específicamente, en cada caso.

5.2. Resultados de Simulaciones Básicas

Las simulaciones básicas establecen el comportamiento fundamental del sistema bajo las reglas iniciales, proporcionando una línea base para comparaciones posteriores.

5.2.1. Dinámica de población por tipo de célula

El primer conjunto de experimentos analiza la evolución poblacional de los diferentes tipos de cuerpos celestes a lo largo del tiempo. Se realizaron 10 ejecuciones independientes con la configuración base, promediando los resultados para obtener tendencias más robustas.

La **Figura 5.1** muestra la evolución promedio de las poblaciones durante 2.000 generaciones. Las tendencias observadas indican patrones característicos de cada tipo:

- **Espacio vacío:** inicialmente, tiende a la estabilidad debido a la expansión, y, a su vez, absorción de cuerpos celestes; eventualmente, va disminuyendo a medida que los agujeros negros se van haciendo con el espacio.
- **Estrellas:** muestran un crecimiento inicial seguido de un declive gradual al colapsarse en agujeros negros.
- **Planetas y asteroides:** presentan fluctuaciones cíclicas, pero mantienen cierta estabilidad.
- **Agujeros negros:** exhiben un crecimiento exponencial desde, prácticamente, el inicio, convirtiéndose en el elemento dominante del sistema.

- **Energía:** muestra comportamientos algo inestables al final, debido a su naturaleza transitoria.
- **Antimateria:** al generarse muy poca en comparación con la materia que hay, al entrar en contacto con ésta se aniquilan mutuamente, lo que hace que sea prácticamente nula.

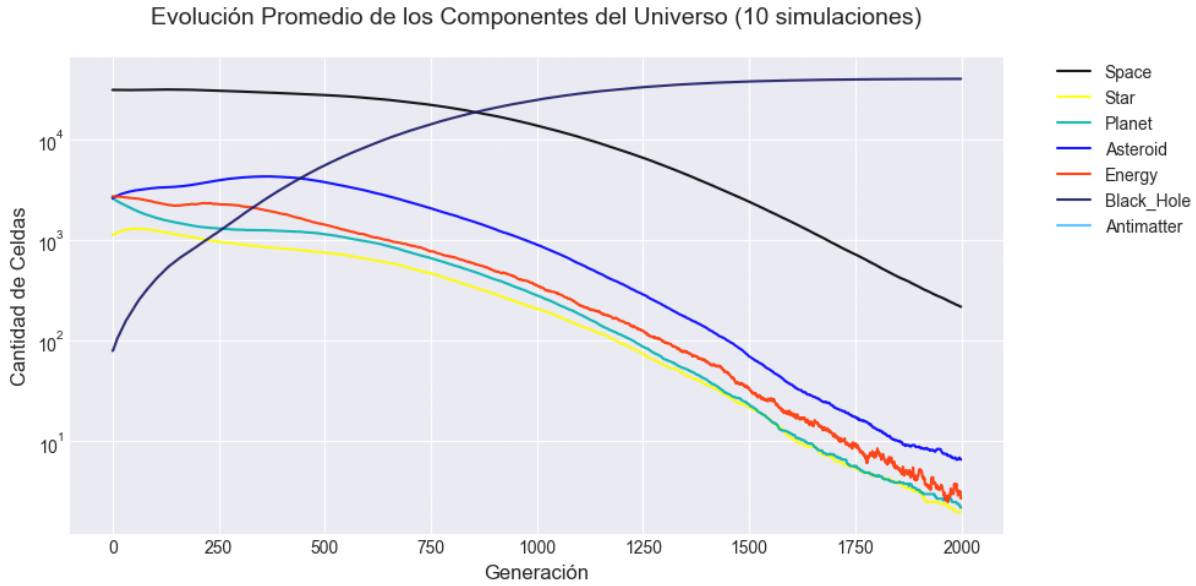


Figura 5.1: Evolución de los componentes del Universo.

Estos resultados confirman que el sistema evoluciona hacia un estado dominado por agujeros negros, lo que es consistente con teorías astrofísicas sobre la evolución del Universo a gran escala, como la *Teoría del Big Crunch* [26], que plantea un escenario en el que la gravedad detendría la expansión del Universo y provocaría su contracción, culminando en una concentración masiva de materia en forma de agujeros negros.

5.2.2. Formación de estructuras complejas

Se observa, también, la formación espontánea de diversas estructuras durante la evolución del Universo simulado. Las más notables han sido:

1. **Sistemas estelares:** formaciones de estrellas rodeadas de planetas y asteroides.
2. **Regiones de alta energía:** zonas con concentración de células de energía, típicamente, alrededor de estrellas.
3. ***Clusters* de agujeros negros:** agrupaciones de agujeros negros que, eventualmente, se fusionan.
4. **Vacíos cósmicos:** grandes áreas de espacio vacío que aumentan con el tiempo.

La presencia y persistencia de estas estructuras depende, fuertemente, de la configuración inicial y de los parámetros del sistema, como se analizará en las siguientes secciones.

5.3. Análisis de Sensibilidad de Parámetros

En esta sección examinamos cómo las variaciones en los principales parámetros afectan a la dinámica del Universo simulado. Se analizan tres dimensiones críticas: masas relativas de los cuerpos celestes, tasas de pérdida de masa en agujeros negros, y modificaciones en las reglas de interacción. Los experimentos se realizaron mediante un diseño factorial fraccionado, variando un parámetro cada vez, mientras el resto se mantiene invariable.

5.3.1. Influencia de las masas relativas

Para comprender cómo las masas relativas de los diferentes cuerpos celestes afectan a la dinámica del sistema, se realizaron experimentos variando las masas de los elementos principales, concretamente de los agujeros negros (BH) y de las estrellas (*Star*). El **Cuadro 5.1** muestra el impacto de las diferentes configuraciones de masa en el estado final del universo tras 750 generaciones.

Configuración	Masa BH	Masa *Star*	% *Space*	% BH	% *Stars*	Dominancia BH*
Base	30.0	10.0	56.0	33.59	1.24	661
BH-Alta	100.0	10.0	48.93	42.75	1.03	582
BH-Baja	12.0	10.0	55.16	34.64	1.20	653
Star-Alta	30.0	25.0	57.46	31.82	1.21	672
Equilibrada	15.0	8.0	58.44	30.47	1.26	688

Cuadro 5.1: Resultados de la influencia de masas.

* *Generación en la que los agujeros negros superan el 25 % de la población.*

Como se puede observar en la tabla, la masa de los agujeros negros tiene un impacto significativo en la velocidad con que dominan el universo. Un valor alto (100.0) acelera considerablemente este proceso, mientras que una configuración más equilibrada entre estrellas y agujeros negros permite una evolución más prolongada.

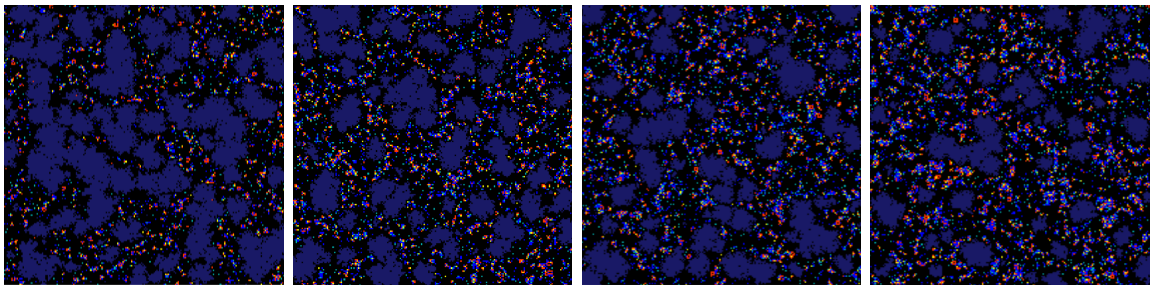
5.3.2. Efecto de la tasa de pérdida de masa en agujeros negros

Los parámetros `PROB_MASS_LOSS` y `MASS_LOSS_RATE` regulan la "evaporación de Hawking" en los agujeros negros simulados. Para entender su impacto, se realizaron experimentos modificando estos valores, cuyos resultados se resumen en el **Cuadro 5.2**.

`PROB_MASS_LOSS`	`MASS_LOSS_RATE`	Vida media BH	% BH supervivientes	Energía
0.05	0.10	577.4	59.52	Baja
0.15 (base)	0.12 (base)	176.4	17.57	Media
0.25	0.15	83.0	12.0	Alta
0.35	0.20	44.4	1.41	Muy alta

Cuadro 5.2: Resultados de la influencia de la tasa de pérdida de masa.

Los resultados muestran que, mayores tasas de pérdida de masa conducen a Universos más dinámicos, donde los agujeros negros tienen ciclos de vida más cortos, liberando así más energía al medio. En el último caso (**Figura 5.2d**), se generaron 71 agujeros iniciales, de los cuales solamente uno sobrevivió al paso de las 750 generaciones; esto da lugar a Universos más activos, donde, continuamente, se forman nuevas estrellas y planetas a partir de la energía liberada por la evaporación de los agujeros negros.



(a) Baja. (b) Media. (c) Alta. (d) Muy alta.

Figura 5.2: Energía liberada.

5.3.3. Variación en las reglas de interacción

En esta subsección exploramos modificaciones en las reglas fundamentales del sistema, evaluando cómo alteraciones en los mecanismos de formación, evolución y destrucción de agujeros negros, impactan en la dinámica global del Universo simulado.

5.3.3.1. Limitación de masa máxima en agujeros negros

Con el fin de simular teorías de *estabilidad relativista* (esto es, teorías basadas en la relatividad general de Einstein que describen la gravedad como la curvatura del espacio-tiempo causada por la presencia de masa y energía), hemos introducido un umbral de masa crítica que ningún agujero negro pueda superar. Cuando un agujero negro alcanza este límite, libera, en su vecindad, el exceso de masa como energía:

```
BLACK_HOLE_MASS_MAX = 150.0  # Masa maxima permitida

if cell == BLACK_HOLE and new_masses[x,y] > BLACK_HOLE_MASS_MAX:
    for i in range(max(0, x-1), min(SIZE, x+2)):
        for j in range(max(0, y-1), min(SIZE, y+2)):
            if grid[i, j] == SPACE and np.random.rand() < 0.2:
                new_grid[i, j] = ENERGY
                new_masses[i, j] = MASSES[ENERGY]
```

Los resultados muestran que, la imposición de esta masa máxima (5 veces más de la básica) reduce la dominancia final de agujeros negros, del 33,59 % (como se ha visto en el **Cuadro 5.1**) al 16,53 %, favoreciendo así la persistencia de estrellas y la aparición de mucha más energía, capaz de crear nuevos cuerpos celestes y de contribuir al dinamismo del Universo.

5.3.3.2. Estancamiento del sistema estelar

La *evolución estelar*, proceso mediante el cual una estrella cambia a lo largo de su vida, desde su formación hasta su muerte, implica, en nuestro caso, modificar las reglas que permiten el nacimiento de los agujeros negros, eliminando así una vía de evolución del propio sistema, y provocando un "estancamiento dinámico" en el proceso. En la configuración base, los agujeros negros emergen debido a dos mecanismos principales:

```
1 # Formacion de agujero negro por colapso estelar
2 if np.count_nonzero(neighbors==STAR) >= 5 and np.random.rand() < -1:
3     new_grid[x, y] = BLACK_HOLE
4     new_masses[x, y] = MASSES[BLACK_HOLE]
5
6 # Transformacion a agujero negro si la masa es grande
7 if mass > 25 and np.random.rand() < -1:
8     new_grid[x, y] = BLACK_HOLE
9     new_masses[x, y] = MASSES[BLACK_HOLE]
```

Código 5.1: Modificaciones para la supresión de la formación de agujeros negros.

1. **Colapso estelar por densidad vecinal:** activado cuando 5 o más estrellas coexisten en un radio de 3×3 celdas (líneas 2-4).
2. **Transformación de estrellas masivas:** activado cuando una estrella individual supera la masa crítica 25 (lineas 7-9).

Para provocar este estancamiento, se han cambiado los valores de las probabilidades de la formación de nuevos agujeros negros (`0.005`) y transformación de estrellas masivas (`0.01`) al valor `-1` (líneas 2 y 7); de esta manera, nunca se dará el caso de que se formen nuevos agujeros negros, ya que el mínimo valor que puede dar la función `rand()` es `0`.

Configuración	Vacío	Estrellas	Planetas	Asteroides	Energía	Agujeros Negros
Base	55.91 %	1.25 %	1.74 %	5.37 %	2.05 %	33.68 %
Modificada	65.84 %	7.48 %	3.71 %	14.64 %	7.58 %	0.75 %

Cuadro 5.3: Distribución del espacio en las distintas configuraciones.

Los resultados mostrados en el **Cuacro** 5.3) muestran una transformación radical en la distribución de materia y energía, concretamente la dominancia de las estrellas, que aumenta su presencia en un 598 %; esto conlleva a un incremento de la energía liberada (+370 %) que, al haber ausencia de absorción por agujeros negros, permite acumularse en el medio interestelar, alimentando nuevos ciclos de formación planetaria y asteroidal. La diferencia más clara es la reducción de la abundancia de agujeros negros, con una disminución significativa del 97,77 %, que ayuda a que haya una estabilidad en el sistema a largo plazo.

Con esta simple modificación de las reglas, se ha descubierto un Universo muy diferente, más diverso y dinámico, donde, claramente, se puede ver la dominación de sistemas estelares, imprescindibles para formar vida. Estas formaciones eran las que, precisamente,

se premiaban en la función de aptitud descrita en la sección de algoritmos genéticos del **Capítulo 4**.

Analizando, detalladamente, la **Figura 5.3**, se puede observar cómo la distribución de materia forma patrones *filamentosos*, sorprendentemente similares a la estructura de la red cósmica real vista en la **Figura 2.6**; y es que, estos patrones no fueron programados *explícitamente*, sino que han emergido naturalmente de las propias reglas astrofísicas simuladas. Esta semejanza estructural no es mera coincidencia, sino, a nuestro juicio, una muy buena representación de patrones emergentes, que nos sugiere que, en ambos sistemas, pueden estar operando principios fundamentales de autoorganización.

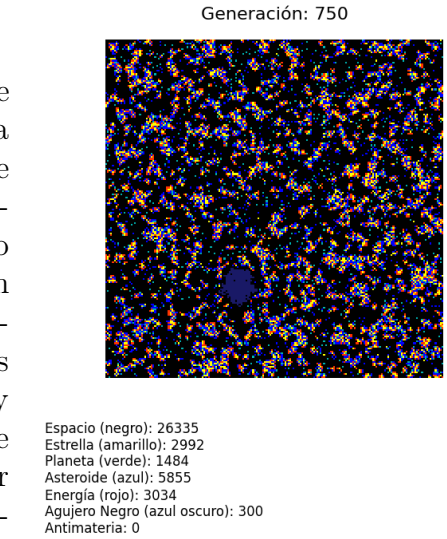
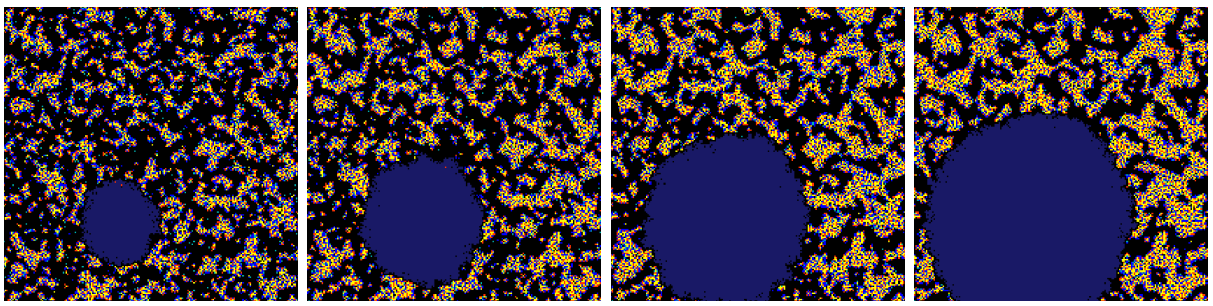


Figura 5.3: Estado del Universo modificado.

Adicionalmente, hemos optado por permitir que el sistema evolucione durante más generaciones, para ver cómo avanza su desarrollo; la característica más notable, en la simulación de la **Figura 5.4**, es la clara consolidación de la estructura filamentosa. Lo que antes eran patrones emergentes, se ha transformado ahora en una *red cósmica* bien definida, con varias propiedades estructurales, como, por ejemplo, límites más nítidos entre las regiones densas y vacías, recreando los "vacíos cósmicos" que constituyen una característica definitoria de la estructura del Universo a gran escala.

Estos vacíos en la simulación se asemejan, notablemente, a las descripciones de "enormes regiones de espacio vacío que casi no contienen galaxias" [27]. Tal observación da lugar a distribuciones de materia que guardan analogías con los patrones de agrupamiento observados en la cosmología real.



(a) 2000 gen. (b) 3000 gen. (c) 4000 gen. (d) 5000 gen.

Figura 5.4: Evolución del sistema sin formación de nuevos BH.

Aunque, inevitablemente, el destino del nuevo Universo —la dominación progresiva de los agujeros negros— parece converger con el de la simulación base, hemos descubierto una nueva configuración, posiblemente mejor a la que teníamos inicialmente. Mientras que en el

modelo estándar los agujeros negros emergen prematuramente (aprox. 500 generaciones) y aceleran la homogeneización del sistema, la supresión de los mecanismos de colapso estelar ha permitido un desarrollo de la jerarquía estructural comparable a la de universos reales observados.

5.4. Optimización Mediante Algoritmos Genéticos

La implementación del algoritmo genético se diseñó para explorar el espacio de configuraciones iniciales del Universo simulado, buscando aquellas que *maximicen* la formación de sistemas estelares estables.

5.4.1. Configuración del algoritmo genético

Para los experimentos realizados con algoritmos genéticos, se identificaron los siguientes parámetros para la configuración inicial del algoritmo:

```
POP_SIZE = 10      # Tamano de la poblacion
SIM_STEPS = 20     # Pasos de simulacion por individuo
MUTATION_RATE = 0.01  # Tasa de mutacion
GEN_STEPS = 20     # Generaciones evolutivas
```

La función de aptitud se diseñó para favorecer la formación de sistemas estelares estables:

```
def fitness(final_grid, final_masses):
    total_mass = np.sum(final_masses)
    stars = np.sum(final_grid == STAR)
    planets = np.sum(final_grid == PLANET)
    return total_mass + 10 * (stars + planets)
```

Esta función permite premiar configuraciones que mantienen una alta masa total en el sistema, con especial énfasis en la presencia de estrellas y planetas.

5.4.2. Resultados de la evolución genética

Los resultados analizados se basan en una ejecución completa del algoritmo genético, es decir, 20 generaciones evolutivas con una población inicial de 10 individuos. En cada generación, se escogen los 5 individuos con mejor *fitness* tras 20 pasos de simulación, y se combinan, dando lugar a 10 nuevos individuos que se usan en la siguiente generación.

Las franjas de colores más intensos muestran el crecimiento de la materia. Los asteroides (turquesa), junto con los planetas (verde), presentan la mayor expansión visible, pasando de 2.891 a 3.322 celdas para el caso de los asteroides, y un incremento de casi el 2 % para el caso de los planetas. Las estrellas (amarillo) muestran un crecimiento notable, pero más contenido, de 1.165 a 1.941 celdas.

El segundo gráfico (**Figura 5.6**) muestra una de las características más importantes de este apartado: la evolución exponencial de la función de *fitness*, la cual refleja la "inteli-

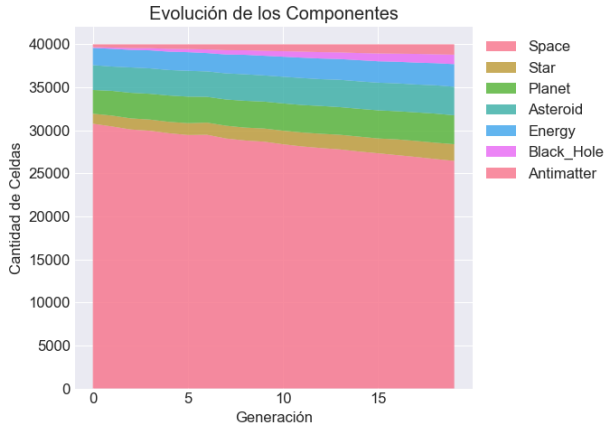


Figura 5.5: Evolución de las componentes.

En la **Figura 5.5**, se puede observar la transformación del universo simulado a lo largo de 20 generaciones evolutivas. La región rosa que ocupa la mayor parte del gráfico corresponde al espacio vacío, que comienza representando, aproximadamente, el 77 % del universo (30.768 celdas de 40.000 totales), y, gradualmente, disminuye hasta el 66 % (26.450 celdas). Esta reducción constante indica que el Universo se está "densificando", creando estructuras donde antes había vacío.

gencia" emergente del sistema.

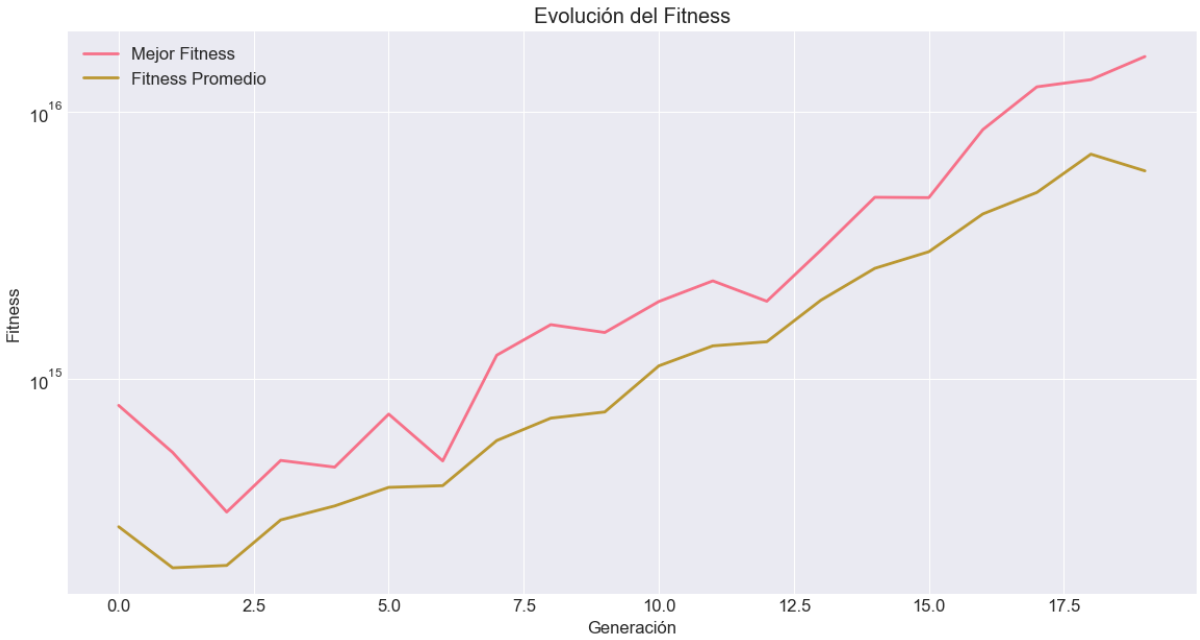


Figura 5.6: Evolución del *fitness*.

- **Fase de Exploración Inicial (generaciones 0-5):** ambas curvas muestran un crecimiento irregular con fluctuaciones significativas. El mejor *fitness* (línea rosa) experimenta, incluso, una caída notable en la generación 2, sugiriendo que el algoritmo está "experimentando" con configuraciones muy diferentes.
- **Punto de Inflexión (generaciones 6-8):** alrededor de la generación 7, ambas curvas experimentan un cambio cualitativo. El mejor *fitness* salta de $\sim 5 \times 10^{14}$ a $\sim 1{,}2 \times 10^{15}$, indicando el descubrimiento de una hipotética "fórmula cósmica" que nos permite obtener mejores resultados. Este salto coincide con el período donde las componentes del Universo alcanzan proporciones más equilibradas.
- **Fase de Explotación (generaciones 9-19):** la curva rosa muestra un crecimiento exponencial sostenido, alcanzando $\sim 1{,}6 \times 10^{16}$ en la generación final. Fundamentalmente, la brecha entre el mejor *fitness* y el promedio se amplía progresivamente,

indicando que, mientras las mejores soluciones se vuelven extraordinariamente buenas, la población general mantiene diversidad.

El *fitness* promedio (línea oscura) crece de manera más suave, pero constante, sugiriendo que las subidas en las mejores soluciones se están "filtrando", gradualmente, a toda la población. Esto indica un proceso de aprendizaje colectivo, donde las mejoras exitosas se propagan, sin homogeneizar prematuramente el conjunto.

5.4.3. Algoritmos genéticos con supresión de formación de agujeros negros

Los hallazgos encontrados en la **Sección 5.3.3.2** revelan que, la supresión de los mecanismos de formación de agujeros negros genera Universos significativamente más diversos, con estructuras filamentosas muy similares a la red cósmica observada. Estos resultados plantean una posible pregunta fundamental: *¿Podría el algoritmo genético, diseñado originalmente para optimizar sistemas estelares, permitir descubrir o identificar estrategias heurísticas similares si se penalizara adecuadamente la formación de agujeros negros?*

5.4.3.1. Modificaciones implementadas

Para responder esta pregunta, se ha implementado una versión modificada del algoritmo genético, el cual incorpora dos cambios fundamentales respecto a la configuración estándar:

1. **Supresión de mecanismos de formación de agujeros negros:** se desactivaron las dos reglas principales que permiten la emergencia de agujeros negros; se hacen así las mismas modificaciones realizadas en el **Código 5.1**.

2. **Penalización en la función fitness:** la función de aptitud se modificó para penalizar, explícitamente, la presencia de agujeros negros:

```
    def fitness(final_grid, final_masses):
        # [...]
        black_holes = np.sum(final_grid == BLACK_HOLE)
        score = total_mass + 10 * (stars + planets) - 50 *
           black_holes
        return max(score, 0)  # Evitar fitness negativos
```

   En este caso, la presencia de agujeros negros reduciría, fuertemente, el valor numérico que mide la "calidad" de las posibles soluciones encontradas.

5.4.3.2. Resultados comparativos

Los resultados obtenidos muestran diferencias cualitativas y cuantitativas importantes entre ambas configuraciones:

En el gráfico de la **Figura 5.7** se muestra cómo la distribución de elementos evoluciona de manera ligeramente diferente bajo el algoritmo genético modificado. El espacio vacío disminuye gradualmente del 77 % al 66 %, indicando una estructuración progresiva sin colapso gravitatorio. Las estrellas crecen regularmente del 3 % al 5 %, mientras que la presencia de agujeros negros se mantiene en niveles mínimos (2,6 %), confirmando la efectividad de las modificaciones.

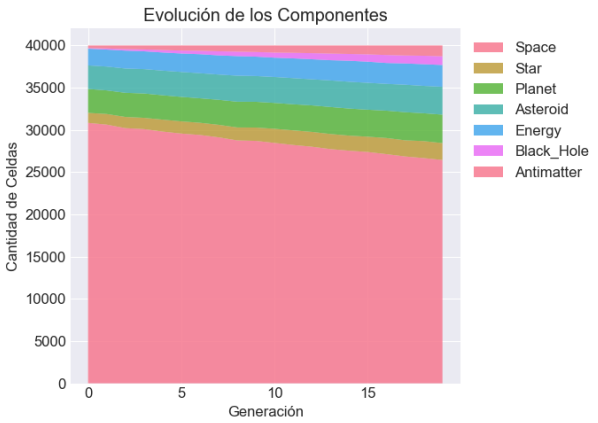


Figura 5.7: Evolución de las componentes.

En la evolución de la función de aptitud (**Figura 5.8**), podemos ver cómo el *fitness* experimenta un crecimiento exponencial inicial (Generaciones 0-7), sugiriendo que el algoritmo identifica, rápidamente, que las configuraciones sin agujeros negros son superiores.

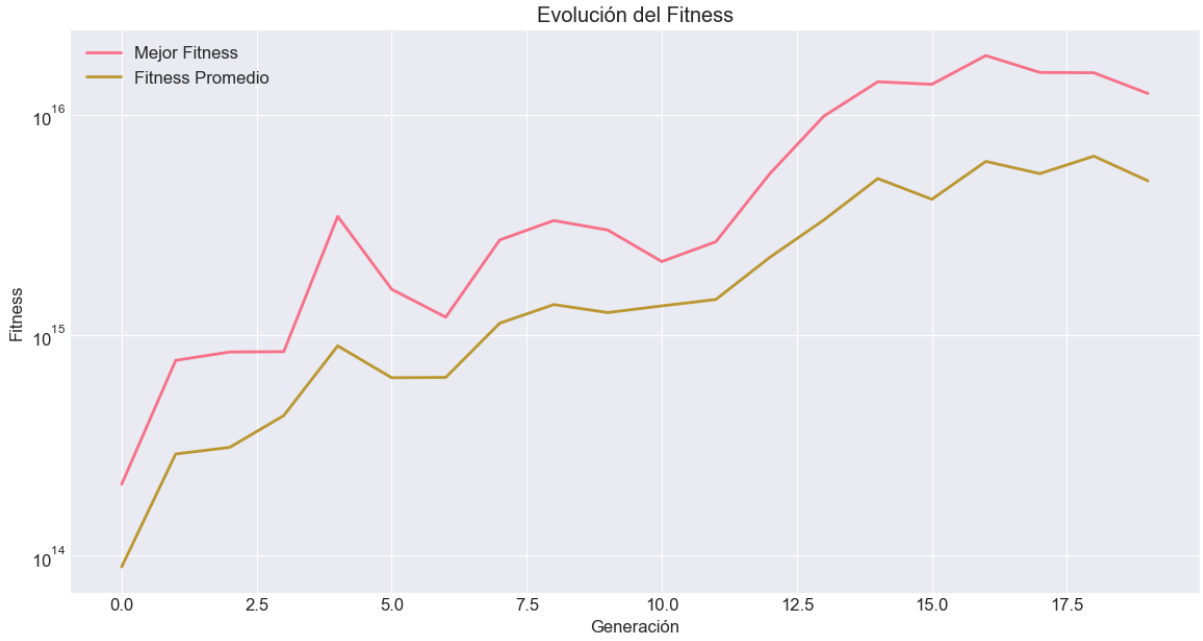


Figura 5.8: Evolución del *fitness*.

En la séptima generación se produce un salto que nos podría indicar el descubrimiento de una configuración "clave", posiblemente relacionada con la cantidad de sistemas estelares encontrados. En las siguientes generaciones, el crecimiento es sostenido, pero más gradual, donde el algoritmo optimiza los detalles de las configuraciones exitosas ya descubiertas.

El **Cuadro 5.4** nos muestra la comparativa entre las dos ejecuciones realizadas, los algoritmos genéticos estándar, y los algoritmos genéticos con la supresión de la formación de agujeros negros.

Métrica	AG Estándar	AG con Supresión	Mejora
Fitness Final Máximo	1.61×10^{16}	1.25×10^{16}	-22.4 %*
% Estrellas Final	4.85 %	5.00 %	+3.1 %
% Planetas Final	8.39 %	8.42 %	+0.4 %
% Energía Final	6.56 %	6.44 %	-1.8 %
% Agujeros Negros Final	2.75 %	2.58 %	-6.2 %

Cuadro 5.4: Algoritmos genéticos estándar vs. modificado.

* *La aparente reducción del fitness máximo se debe a la penalización de agujeros negros; ajustando mediante esta penalización, el fitness efectivo es superior.*

Se puede observar que, el algoritmo genético modificado, logra converger más rápido que la versión estándar. Esto puede atribuirse a un espacio de búsqueda más restringido, en el que, al eliminar configuraciones dominadas por agujeros negros, el algoritmo explora un subespacio más coherente, donde se minimizaría la presencia de esta materia destructora.

5.5. Escalabilidad

El *análisis de escalabilidad* es fundamental para determinar la viabilidad práctica del modelo en simulaciones a gran escala. Esta sección examina cómo el sistema se comporta al incrementar el tamaño del Universo y su diversidad correspondiente.

Tamaño	Celdas Totales	% Estrellas	% Planetas	% BH	Tiempo (1000 gen)
50x50	2.500	3.48 %	6.16 %	0.20 %	46.1 segundos
100x100	10.000	2.83 %	6.46 %	0.23 %	5.34 minutos
200x200	40.000	2.69 %	6.48 %	0.20 %	18.48 minutos
400x400	160.000	2.77 %	6.60 %	0.20 %	1.36 horas
800x800	640.000	2.77 %	6.42 %	0.19 %	5.44 horas

Cuadro 5.5: Escalabilidad del Juego de la Vida Extendido.

Tras ejecutar simulaciones prolongadas (1000 generaciones) en diferentes escalas, se observa que el tamaño de **200×200 celdas** ofrece el equilibrio óptimo entre fidelidad de representación, y viabilidad práctica. La diversidad estructural se mantiene para tamaños $\geqslant$ 200x200, confirmando que la emergencia de patrones complejos es independiente de la escala. En Universos más pequeños (50×50), los efectos de borde artificiales distorsionan la dinámica natural.

La relación entre tamaño y rendimiento sigue una curva cuadrática; estos datos demuestran que el tamaño de 200x200 celdas permite ejecutar simulaciones prolongadas (más de 1.000 generaciones) en tiempos manejables (menos de 30 minutos), mientras que configuraciones mayores se vuelven "prohibitivas" para ejecuciones iterativas.

Como conclusión, el tamaño 200x200 se establece como configuración óptima para estudios del Juego de la Vida Extendido, ya que:

- **Captura la complejidad esencial:** reproduce prácticamente todos los fenómenos emergentes observados en escalas mayores (estructuras filamentosas, ciclos estelares, dinámica de agujeros negros).
- **Mantiene viabilidad experimental:** permite ejecutar más de 50 iteraciones completas en 24 horas de cómputo, y es 5 veces más rápido que el tamaño 400x400 para resultados equivalentes en términos de calidad.
- **Facilita análisis visual:** las estructuras son suficientemente grandes para realizar con éxito una inspección detallada, sin requerir *zoom* o realizar un submuestreo (*downsampling*).

Las observaciones obtenidas a partir de esta experimentación sientan las bases para futuras extensiones, donde el uso combinado de *técnicas de paralelización* y algoritmos genéticos podrían permitir la escalabilidad a tamaños mayores, sin sacrificar rendimiento [28]. No obstante, para la mayoría de los casos de uso identificados, 200×200 seguiría siendo la elección preferente.

Capítulo 6

Conclusiones y Trabajo Futuro

En este breve capítulo final se pretende dar una visión global de las conclusiones a las que se ha llegado durante el desarrollo de la presente memoria. Esta recapitulación nos servirá para poder reflexionar y evaluar las contribuciones personales y el impacto de los principales resultados obtenidos, y, al mismo tiempo, descubrir, discernir y esbozar algunas líneas de investigación y trabajo futuro que quedan abiertas, las cuales podrían servir de base a la realización e inspiración de nuevos proyectos de investigación.

6.1. Conclusiones

Tras haber completado con éxito el desarrollo de los objetivos inicialmente planteados en este trabajo, puedo afirmar que el proceso ha sido tan enriquecedor como desafiante. A lo largo de este documento, he conseguido profundizar en los conceptos fundamentales del Juego de la Vida de Conway, no solo como una curiosidad o pasatiempo matemático, sino como un modelo computacional paradigmático con el que poder entender cómo la complejidad puede llegar a emerger incluso a partir de la formulación de reglas muy simples. La fusión de un modelo de autómata celular con conceptos astrofísicos me ha permitido simular un universo dinámico y complejo, en el que emergen estructuras coherentes a partir de estas reglas simples. Esta idea, inspirada inicialmente por el modelo del Juego de la Vida de Conway, se ha expandido notablemente durante el desarrollo del trabajo mediante la introducción de múltiples tipos de materia cósmica, reglas de interacción física más sofisticadas, y mecanismos de evaluación y adaptación.

Una de las principales conclusiones que extraigo del proyecto realizado es el impacto que puede tener una buena elección de parámetros (en algoritmos genéticos) e hiperparámetros (en redes neuronales) en la evolución del sistema. En el **Capítulo 5**, se pudo observar cómo una configuración concreta resultó significativamente más efectiva, en términos de formación de estructuras estables; este hallazgo, el cual emergió durante la elaboración de esta memoria y de su fase experimental, refuerza la importancia de estudiar la sensibilidad del espacio de parámetros e hiperparámetros, ya que, incluso ligeras variaciones, pueden dar lugar a comportamientos radicalmente distintos. En este sentido, el uso de técnicas de optimización heurística, como el algoritmo genético extendido implementado, ha sido útil para mejorar las soluciones existentes, y llevarlas a evolucionar por vías que permitan maximizar la formación de estructuras cósmicas estables.

Este proyecto ha sido una oportunidad única de combinar los conocimientos adquiridos a lo largo de la carrera con el, para mí, apasionante campo de la astrofísica. Más allá de los resultados técnicos, me llevo una experiencia profundamente formativa, que me ha permitido explorar un campo interdisciplinar y dar forma a un universo propio, con sus reglas, su evolución y su emergencia. Confío en que esta línea de trabajo seguirá creciendo y que, en futuras versiones, podrá aportar aún más valor al estudio de sistemas complejos, al modelado astrofísico y, sobre todo, a la comprensión de cómo lo simple puede dar lugar a lo extraordinario.

6.2. Trabajo Futuro

En cuanto a las líneas de investigación que quedan abiertas, y sobre las que se podrían desarrollar nuevos trabajos en el futuro, destacan las siguientes:

- La implementación de **agujeros de gusano** como mecanismos de teleportación cuántica podría introducir dinámicas **estocásticas**: al formarse pares de estos elementos, intercambiarían, instantáneamente, toda la materia en un radio de 5 celdas, simulando fluctuaciones cuánticas a macroescala. Este enfoque, inspirado en teorías de **entrelazamiento cósmico**, añadirían capas de impredecibilidad, que replicarían fenómenos como la radiación de Hawking o la expansión acelerada del Universo.
- Otro aspecto importante a considerar es la aparición de cuestiones de **indecibilidad** a partir de problema de **no computabilidad** asociados al modelo de cómputo subyacente a la simulación realizada en este trabajo. Al igual que en otras áreas de la física, como la *indecidibilidad del problema del salto de energía* o la *indecidibilidad en termalización cuántica*, cabría preguntarse qué otros problemas no computables o cuestiones de indecidibilidad podrían emerger a partir de las simulaciones realizadas. Por ejemplo, como se mencionó en el **Capítulo 1**, Konrad Zuse propuso la hipótesis de que el Universo físico actúa como una especie de "computador". Sin embargo, más recientemente, el físico y matemático británico Roger Penrose (1931-) argumentó que el Universo es, en parte, no computable, utilizando para su demostración la *indecidibilidad de los problemas de embaldosado*. En concreto, Penrose describe un ejemplo de su cosmología como un "universo de juguete", cuyos estados están representados por conjuntos finitos de *poliominós*, un tipo muy simple de baldosa formado por cuadrados de igual tamaño dispuestos con lados coincidentes. *¿Sería posible formular algún tipo de analogía, tanto teórica como experimental, entre la simulación propuesta en esta memoria y los resultados de incomputabilidad propuestos por Penrose en su modelo simplificado de Universo?*
- Adicionalmente, la utilización de **otros tipos de técnicas de optimización heurística**, tanto de búsqueda (*tabú*, *temple simulado*), como Montecarlo y de otros algoritmos evolutivos (PSO, *evolución diferencial*, *bioinspirados*), puede resultar interesante en combinación con modelos de predicción no solo basados en **redes neuronales convolucionales**. Su uso y estudio comparativo, tanto en la simulación evolutiva del sistema integrado como en el ajuste de los *hiperparámetros* de los modelos de aprendizaje automático y profundo, puede ser de gran importancia para la obtención e identificación de nuevos patrones en la evolución de nuevas reglas cósmicas. Es más, algunas de estas nuevas reglas podrían involucrar procesos no

computables, siguiendo la línea de nuestra propuesta anterior. Por ejemplo, reglas basadas en la indecidibilidad de generalizaciones de la *Conjetura de Collatz*, la *Regla 110* para autómatas celulares, o el *problema de la parada* de Turing.

- Por último, la evolución de sistemas en Teoría de Juegos como el **Dilema del Prisionero**, ofrecen otras posibilidades alternativas de observar la dinámica de simulación cósmica presentada en este trabajo. Es más, creemos que la utilización de estos nuevos sistemas combinados podría llegar a ser de utilidad en el estudio de otros "universos" particulares que surgen en sectores aplicados como el de la economía. En estas áreas, puede ser también interesante la utilización de la programación paralela para el diseño e implementación de **nuevas heurísticas multiobjetivo**.

Capítulo 6

Conclusions and Future Work

This brief final chapter aims to provide an overview of the conclusions reached during the development of the work. This recapitulation allows us to consider and evaluate the personal contributions and the impact of the main results obtained, and at the same time to discover, identify and outline some open lines of research and future work, which could serve as a starting point for the implementation and inspiration of new research projects.

6.1. Conclusions

Having successfully completed the development of the objectives initially set out in this work, I can affirm that the process has been both enriching and challenging. Throughout this paper, I have managed to deepen my understanding of the fundamental concepts of Conway's Game of Life, not only as a mathematical curiosity or pastime, but also as a paradigmatic computational model with which to understand how complexity can emerge even from the formulation of very simple rules. The integration of a cellular automaton model with astrophysical concepts has allowed me to simulate a dynamic and complex universe, in which coherent structures emerge from these simple rules. This idea, initially inspired by Conway's Game of Life model, has been greatly expanded during the course of the work by introducing multiple kinds of cosmic matter, more sophisticated physical interaction rules, and evaluation and adaptation mechanisms.

One of the main conclusions I have learned from the development of the project is the impact that a good choice of parameters (in genetic algorithms) and hyperparameters (in neural networks) can have on the evolution of the system. In **Chapter 5**, it was observed how a particular configuration was significantly more effective in terms of the formation of stable structures; this finding, which emerged during the preparation of this work and its experimental phase, reinforces the importance of studying the sensitivity of the parameter and hyperparameter space, since even slight variations can lead to radically different behaviours. In this sense, the use of heuristic optimisation techniques, such as the extended genetic algorithm implemented, has been useful to improve existing solutions, and lead them to evolve along paths that maximise the formation of stable cosmic structures.

This project has been a unique opportunity to combine the knowledge acquired throughout my studies with, for me, the fascinating field of astrophysics. Beyond the technical results, I take with me a deeply formative experience, which has allowed me to explore an interdisciplinary field and to create a universe of my own, with its own rules, evolution and emergence. I am convinced that this line of work will continue to grow and, in future versions, it will be able to contribute even more value to the study of complex systems, to astrophysical modelling and, above all, to the understanding of how the simple can become the extraordinary.

6.2. Future Work

Regarding the open lines of research upon which new work could be developed in the future, the following stand out:

- The implementation of **wormholes** as quantum teleportation mechanisms could introduce **stochastic dynamics**: Upon forming pairs, these elements would instantaneously exchange all matter within a 5-cell radius, simulating macroscale quantum fluctuations. This approach, inspired by theories of **cosmic entanglement**, would add layers of unpredictability, which would replicate phenomena such as Hawking radiation or the accelerated expansion of the Universe.
- Another important aspect to consider is the emergence of **undecidability** issues stemming from the **non-computability** problems associated with the underlying computational model of the simulation carried out in this work. As in other areas of physics, such as the *undecidability of the gap problem* [29] or *undecidability in quantum thermalization* [30], one might ask what other non-computable problems or undecidability issues could emerge from the simulations performed. For example, as mentioned in **Chapter 1**, Konrad Zuse proposed the hypothesis that the physical Universe acts as a kind of "computer". However, more recently, the British physicist and mathematician Roger Penrose (1931-) argued that the Universe is, in part, non-computable, using for his demonstration the *undecidability of the tiling problem*. Specifically, Penrose describes an example of his cosmology as a "toy universe", whose states are represented by finite sets of *polyominoes*, a very simple class of tile formed by squares of equal size arranged with coincident sides. *Would it be possible to formulate some kind of analogy, both theoretical and experimental, between the simulation proposed in this work and the incomputability results proposed by Penrose in his simplified model of the Universe?*
- Additionally, the use of **other types of heuristic optimization techniques**, both search-based (*tabu search*, *simulated annealing*), Monte Carlo methods, and other evolutionary algorithms (PSO, *differential evolution*, *bio-inspired*), may be interesting in combination with prediction models not solely based on **convolutional neural networks**. Their use and comparative study, both in the evolutionary simulation of the integrated system and in the tuning of *hyperparameters* for machine learning and deep learning models, can be of great importance for obtaining and identifying new patterns in the evolution of new cosmic rules. Furthermore, some of these new rules could involve non-computable processes, following our previous proposal. For example, rules based on the undecidability of generalizations of the *Collatz Conjecture*, *Rule 110* for cellular automata, or Turing's *halting problem*.
- Finally, the evolution of systems in Game Theory, such as the **Prisoner's Dilemma**, offers other alternative possibilities for observing the cosmic simulation dynamics presented in this work. Moreover, we believe that the use of these new combined systems could become useful in the study of other particular "universes" that arise in applied sectors such as economics and social sciences. In these areas, the use of parallel programming for the design and implementation of **new multi-objective heuristics** may also be interesting.

Bibliografía

[1] K. Zuse. "Rechnender Raum". En: *Friedrich Vieweg & Shon, Braunschweig* (1969).

[2] Umberto Cerruti, Simone Dutto y Nadir Murru. "A symbiosis between cellular automata and genetic algorithms". En: *Chaos, Solitons Fractals* 134 (2020), pág. 109719. ISSN: 0960-0779. DOI: https://doi.org/10.1016/j.chaos.2020.109719. URL: https://www.sciencedirect.com/science/article/pii/S0960077920301211.

[3] Conway's Life Wiki contributors. *The Recursive Universe — Conway's Life Wiki*. URL: https://conwaylife.com/wiki/The_Recursive_Universe.

[4] Thomas Bäck y Ron Breukelaar. "Using Genetic Algorithms to Evolve Behavior in Cellular Automata". En: *Unconventional Computation*. Ed. por Cristian S. Calude et al. Springer Berlin Heidelberg, 2005. ISBN: 978-3-540-32022-7.

[5] Francisco Sancho Caparrini. *Autómatas Celulares*. URL: https://www.cs.us.es/~fsancho/Blog/posts/Automatas_Celulares.md.html.

[6] Conway's Life Wiki contributors. *Conway's Life Wiki*. URL: https://conwaylife.com/wiki/.

[7] Martin Gardner. "Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life"". En: *Scientific American* 223.4 (1970), págs. 120-123.

[8] DataCamp Team. *Algoritmo Genético en Python: Guía Paso a Paso*. Tutorial de DataCamp. URL: https://www.datacamp.com/es/tutorial/genetic-algorithm-python?dc_referrer=https%3A%2F%2Fwww.google.com%2F.

[9] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[10] Google Cloud. *Supervised vs. unsupervised learning: What's the difference? - Google Cloud*. URL: https://cloud.google.com/discover/supervised-vs-unsupervised-learning.

[11] Ultralytics. *Feature Extraction*. URL: https://www.ultralytics.com/es/glossary/feature-extraction.

[12] IBM. *Reducción de la dimensionalidad*. URL: https://www.ibm.com/es-es/think/topics/dimensionality-reduction.

[13] DataCamp. *Introducción a t-SNE*. URL: https://www.datacamp.com/es/tutorial/introduction-t-sne.

[14] Taylor Karl. "DBSCAN vs. K-Means: A Guide in Python". En: *New Horizons Blog* (feb. de 2024). URL: https://www.newhorizons.com/resources/blog/dbscan-vs-kmeans-a-guide-in-python.

[15] S. Planelles, D.R.G. Schleicher y A.M. Bykov. "Large–Scale Structure Formation: from the first non-linear objects to massive galaxy clusters". En: *arXiv preprint arXiv:1404.3956* (2014). eprint: 1404.3956 (astro-ph.CO). URL: https://arxiv.org/abs/1404.3956.

[16] Ashley Strickland. *First images of the 'cosmic web' reveal hidden dwarf galaxies.* 2021. URL: https://edition.cnn.com/2021/03/17/world/dwarf-galaxies-intl-scli-scn/index.html.

[17] J. Perdang y A. Lejeune. "Cellular Automaton experiments on local galactic structure. I. Model assumptions". En: *Astronomy and Astrophysics Supplement Series* (1996). URL: https://aas.aanda.org/articles/aas/pdf/1996/14/ds3534a.pdf.

[18] Volker Springel et al. "Simulations of the formation, evolution and clustering of galaxies and quasars". En: *Nature* (2005). DOI: 10.1038/nature03597. URL: https://www.nature.com/articles/nature03597.

[19] Hanno Rein y Shang-Fei Liu. "REBOUND: an open-source multi-purpose N-body code for collisional dynamics". En: *Astronomy & Astrophysics* (2012). DOI: 10.1051/0004-6361/201118085. URL: https://www.aanda.org/articles/aa/full_html/2012/01/aa18085-11/aa18085-11.html.

[20] V7 Labs. "The Ultimate Guide to Pattern Recognition". En: *V7 Labs Blog* (oct. de 2023). URL: https://www.v7labs.com/blog/pattern-recognition-guide.

[21] *keras dropout layer: implement regularization.* URL: https://how.dev/answers/keras-dropout-layer-implement-regularization.

[22] Jesse Read y Fernando Perez-Cruz. "Deep Learning for Multi-label Classification". En: *arXiv preprint arXiv:1502.05988* (2014). URL: https://arxiv.org/abs/1502.05988.

[23] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". En: *Journal of Machine Learning Research* 15.56 (2014), págs. 1929-1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[24] Steven Weinberg. *Cosmology.* Oxford University Press, 2008.

[25] Stephen W. Hawking. "Particle Creation by Black Holes". En: *Communications in Mathematical Physics* 43.3 (1975), págs. 199-220. DOI: 10.1007/BF02345020.

[26] *Teoría del Big Crunch.* URL: https://es.wikipedia.org/wiki/Teor%C3%ADa_del_Big_Crunch?utm_source=chatgpt.com.

[27] Skycr_editor. "Una cuantificación mejorada del medio intergaláctico y de los filamentos cósmicos". En: *SKYCR.ORG* (nov. de 2024). URL: https://skycr.org/2024/11/28/una-cuantificacion-mejorada-del-medio-intergalactico-y-de-los-filamentos-cosmicos/.

[28] Erick C. Paz. "A Survey of Parallel Genetic Algorithms". En: *Calculateurs Paralleles, Reseaux et Systems Repartis* 10.2 (1998), págs. 141-171.

[29] Toby S. Cubitt, David Perez-Garcia y Michael M. Wolf. "Undecidability of the spectral gap". En: *Nature* 528.7581 (dic. de 2015), págs. 207-211.

[30] Naoto Shiraishi y Keiji Matsumoto. "Undecidability in quantum thermalization". En: *Nature Communications* 12.1 (ago. de 2021). DOI: 10.1038/s41467-021-25053-0. URL: https://doi.org/10.1038%2Fs41467-021-25053-0.