ULPGC
Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería Informática

TRABAJO DE FIN DE GRADO - 2022/2023

Grado en Ingeniería Informática

Lenia+: Framework para experimentación con autómatas celulares en Vida Artificial

ESTUDIANTE: JITEN RAJESH PARWANI

TUTOR: FRANCISCO MARIO HERNÁNDEZ TEJERA

Las Palmas de Gran Canaria, julio de 2023

ÍNDICE

GLOSARIO

AUTÓMATA CELULAR: modelo de un sistema formado por celdas individuales que viven en una cuadrícula. Cada celda puede estar en un estado determinado e interactuar con sus celdas vecinas según reglas específicas. Con el paso del tiempo, estas reglas determinan cómo cambian los estados de las celdas.

ESPACIO CELULAR: entorno en el que se desarrolla el autómata. Es el espacio que se divide en celdas, donde cada celda puede tener un estado determinado.

CELDA: unidades básicas que componen la estructura del espacio celular. Representa un elemento discreto dentro del espacio celular y puede tener un estado específico.

ESTADO DE CELDA: valor binario, como vivo/muerto. También puede ser estado discreto, con múltiples valores posibles (colores, números, etc).

EJEMPLAR: autómatas celulares que presentan patrones autónomos complejos, caracterizados por ser geométricos, metaméricos, difusos, resilientes, adaptables.

SIMULACIÓN: técnica que permite llevar a cabo experimentos en entornos controlados (computadora) con el fin de comprender el comportamiento y la estructura de sistemas complejos del mundo real a lo largo del tiempo.

METAMÉRICO: fenómeno biológico en el cual un organismo está compuesto por segmentos repetitivos que son similares, pero no idénticos entre sí.

MOTILIDAD: término en la ciencia de la biología para expresar la habilidad de moverse independientemente.

HOMEOSTASIS: capacidad que posee un organismo de mantener una condición interna estable, equilibrando los cambios en su entorno mediante el intercambio regulado de materia y energía con el exterior.

RESILIENTE: capacidad de una persona o entidad para sobreponerse y adaptarse a situaciones difíciles, adversidades o momentos críticos.

CANAL RGB: componente primario de color en un modelo de imagen RGB (Red, Green, Blue). En este modelo, una imagen se descompone en tres canales: rojo, verde y azul. Cada canal representa la intensidad o contribución de ese color específico en cada píxel de la imagen.

IMAGEN DIGITAL: representación bidimensional de una imagen utilizando bits (1 y 0). En el contexto de la informática y los dispositivos digitales, la imagen digital se almacena como un archivo informático que contiene una serie de matrices numéricas que definen las características de la imagen.

IMAGEN MULTIDIMENSIONAL: en el contexto de las imágenes digitales, una imagen multidimensional puede referirse a una imagen que tiene información adicional más allá de los canales de color estándar (rojo, verde, azul en el modelo RGB). Esto permite una representación más completa y detallada de la imagen en comparación con una imagen bidimensional.

CORRELACIÓN CRUZADA: técnica utilizada para determinar la similitud o correspondencia entre dos matrices. En el contexto de dos matrices 2D, la correlación cruzada bidimensional calcula la similitud entre ellas mediante la superposición y comparación de sus elementos. Esto puede ser útil en diversas aplicaciones de procesamiento de imágenes y señales.

REINTEGRATION TRACKING: algoritmo que usa una técnica eficiente y precisa para realizar simulaciones de fluidos. Utiliza una cuadrícula de autómatas celulares para realizar el seguimiento y promedio de las propiedades de las partículas en cada célula.

CAPÍTULO 1. INTRODUCCIÓN

1.1. MOTIVACIÓN

La motivación principal de este proyecto es contribuir al campo de la Vida Artificial. Desde la década pasada, muchos investigadores se han interesado por los fundamentos de la vida y el uso de la informática para realizar estudios. Se plantean preguntas sobre la naturaleza de la vida, la conciencia y la relación entre lo natural y lo artificial. ¿Cómo emerge la vida de lo no-vivo? ¿Cuáles son los potenciales y límites de los sistemas vivos? ¿Cómo se relaciona la vida con la mente, las máquinas y la cultura?

Con la motivación de realizar una pequeña contribución al entendimiento de la vida, en este proyecto se desarrolla una herramienta que permita experimentar con uno de los modelos más recientes y prometedores para el estudio de Vida Artificial.

1.2. OBJETIVOS CUBIERTOS

Los objetivos que se pretenden alcanzar con este trabajo son varios. Estos se pueden separar en dos grupos. El primero son los objetivos académicos, que se tratan de los propios de un Trabajo de Fin De Grado del área de la Ingeniería Informática. El segundo grupo son los objetivos de la herramienta, que tienen el propósito de ofrecer un software que facilite el estudio en Vida Artificial.

1.2.1. OBJETIVOS ACADÉMICOS

En primer lugar, están los objetivos relacionados con el desarrollo de un proyecto software:

- Desarrollar un marco software (framework) que permita realizar experimentación en entornos con autómatas celulares de la familia Lenia
- Abordar y desarrollar un proyecto software completo que permita cimentar los conocimientos adquiridos en los estudios del grado y adquirir la formación y experiencia ligada a la ejecución de un proyecto software en todas sus fases, desde el principio al final
- Enfrentarse al proceso de experimentación y evaluación de resultados de un desarrollo software
- Desarrollar las actividades de comunicación propias de todo trabajo de esta naturaleza que implican la redacción de la correspondiente memoria final y la preparación de la presentación de la defensa del trabajo.

1.2.2. OBJETIVOS DE LA HERRAMIENTA

En segundo lugar, se pretende alcanzar los objetivos relacionados con el área de la Vida Artificial. Es decir, conocer aquellos métodos y técnicas relacionados con la aplicación de conceptos de biología y la vida en un entorno software que permita estudiar esos conceptos. Por un lado, se pretende entender las características asociadas a la vida, de interés en las ciencias biológicas. Por otro lado, está el objetivo de extraer conocimiento y conclusiones que puedan ser útiles en el desarrollo de sistemas artificiales.

1.3. ORGANIZACIÓN DEL DOCUMENTO

El proyecto se ha dividido en varias partes. A continuación, se pondrá en contexto el tema del que se trata el actual trabajo, para lo cual se introducirán los conceptos de la Vida Artificial y el estado del arte de sus investigaciones. Luego, se introducirán algunas métricas del software que se emplean para el análisis del sistema de autómata celular a tratar. Tras este capítulo, se explicarán la metodología y planificación con la que se procedió. Seguidamente, se explicará el desarrollo del proyecto, que se divide en varias etapas, que incluye desde la arquitectura del programa, hasta su interfaz y herramientas. Respecto a las librerías y herramientas empleadas, se destacarán especialmente las razones por las que se decidió emplearlas, aprovechando todas sus ventajas. Tras este capítulo, se mostrarán los resultados y experimentos del trabajo realizado. Por último, se incluye una sección de conclusiones y trabajos futuros que podrían ser interesantes para el tema del proyecto. Por último, se encontrará la bibliografía seguidas de varios anexos.

CAPÍTULO 2. CONTEXTO

En este capítulo, se pondrá en contexto el ámbito de investigación en el que se trabaja en este proyecto, que es la Vida Artificial. Por otra parte, se introducirá el modelo Lenia (Chan, 2018), el cual es el punto de apoyo para este proyecto, así como una de sus variantes.

2.1. LA VIDA ARTIFICIAL

El término “Vida Artificial” fue utilizado por primera vez por el científico Christopher Langton a finales de la década de 1980 durante la "Primera Conferencia Internacional de la Síntesis y Simulación de Sistemas Vivientes". En esta conferencia se discutieron enfoques teóricos y computacionales relacionados con la vida artificial (Grand, 2016).

Se trata de un campo que probablemente pueda confundirse con la Inteligencia Artificial. Sin embargo, es un ámbito científico que trata de entender y experimentar artificialmente con la vida, a diferencia de la Inteligencia Artificial, que trata de entender la naturaleza de la inteligencia y desarrollar las capacidades tecnológicas de su síntesis y desarrollo artificial.

Además, la Vida Artificial no solo se preocupa por entender la vida “como la conocemos” (“life-as-we-know-it”), sino también de “cómo podría ser” (“life-as-it-might-be”) (Langton, 1997). En este sentido, a diferencia de la biología, que se basa (mayoritariamente) en la vida tal como la conocemos en la Tierra, la Vida Artificial busca crear sistemas artificiales que exhiban propiedades de los seres vivos sin necesariamente depender de la biología tradicional.

Este campo se basa en varias disciplinas para crear los sistemas artificiales con las características mencionadas. Combina las ciencias de computación, la biología, la química, la física y otras disciplinas.

Se distinguen tres tipos principales de vida artificial (Bedau, 2003): *soft* (software), *hard* (hardware) y *wet* (bioquímica). El de tipo *soft* es el que se tratará en este proyecto. Este tipo consiste en simulaciones que exhiban comportamientos semejantes a los que se podrían encontrar en los sistemas vivos. Entiéndase por “simulación” los experimentos que imitan fenómenos realizados mediante computadoras.

La computación tiene un papel fundamental en este sentido, y especialmente en la actualidad, cuando la potencia computacional es más asequible. A través de las simulaciones, es posible realizar estudios que permiten experimentar y comprender los comportamientos “vivos”.

Antes de entender el pasado y estado del arte de este campo, es necesario dar una breve explicación de los autómatas celulares. Se trata de modelos matemáticos y computacionales con un conjunto finito de reglas. Sus componentes básicos son: el espacio de celdas n-dimensional; los instantes de tiempo; el conjunto de estados posibles de cada celda; la vecindad de cada celda y las reglas de actualización de cada celda. Con estos componentes, es posible simplificar la simulación de fenómenos complejos que emergen de reglas simples. Debido a sus propiedades sencillas y simples, desde el siglo pasado han sido objeto de interés en diversas áreas de investigación, donde han contribuido al modelado de sistemas complejos.

Por las razones mencionadas anteriormente, la Vida Artificial es otra de las áreas donde los autómatas celulares aportan una gran contribución. Ahora, se explicará brevemente el pasado del modelo Lenia, para facilitar su entendimiento.

2.2. EL PASADO: GAME OF LIFE

Uno de los primeros inicios de los autómatas celulares es Game of Life, o Juego de la Vida de John Conway (Gardner, 1970). Se trata de un juego simulado sobre una cuadrícula de celdas, que representa el espacio celular o mundo. En esta, cada celda tiene 2 posibles estados: vivo o muerto. En la Ilustración 1 se muestra los componentes principales del juego.

Ilustración 1. Cuadrícula de Game Of Life

Se inicia un mundo con estados inicial para sus celdas, que es la primera generación. A partir de esta, se genera una nueva generación. Para ello, se actualizan las nuevas celdas en base a ciertas reglas que dependen del estado anterior de las respectivas celdas y de su vecindad. La vecindad de una celda en este juego se define como las celdas horizontal, vertical y diagonalmente adyacentes a la misma, sumando un total de 8 celdas.

Las reglas que Conway estableció en el juego son las siguientes:

1. La celda se actualizará al estado vivo solo si en la anterior generación tenía una vecindad de 3 celdas vivas (nacimiento)

2. Se mantendrá en el estado vivo si en la anterior generación tenía una vecindad de entre 3 o 4 celdas vivas (supervivencia)

3. En cualquier otro caso, la celda se mantendrá o actualizará al estado “muerto”

Este modelo es el punto de partida para el estado del arte del que se hablará en este proyecto. A continuación, se explicará la reciente familia de autómatas celulares Lenia.

2.3. EL ESTADO DEL ARTE: LENIA

Tras entender la dinámica de Game of Life, en esta sección se explicará el modelo de Lenia. Se trata de una familia de autómatas celulares ideada por Bert Chan en 2018 (Chan, 2018). Lenia generaliza el comentado Game of Life, con la intención de ser simulado en espacio, tiempo y estados continuos (de ahí el nombre en latín "lenis", que significa "suave").

Ha atraído la atención por su capacidad para generar una amplia diversidad de patrones de autoorganización, incluidos los patrones espacialmente localizados (SLP) que se asemejan a criaturas artificiales parecidas a la vida con comportamientos complejos.

En primer lugar, el autor define el espacio celular de Lenia. De forma similar a Game of Life, en el nuevo modelo el mundo está representado por una matriz *A*, con la diferencia de que cada elemento es un número entre 0 y 1. Es esta aproximación la que lo caracteriza como “estados continuos”. En la Ilustración 2 se muestra un ejemplo, obtenido del notebook original de Python de Lenia.

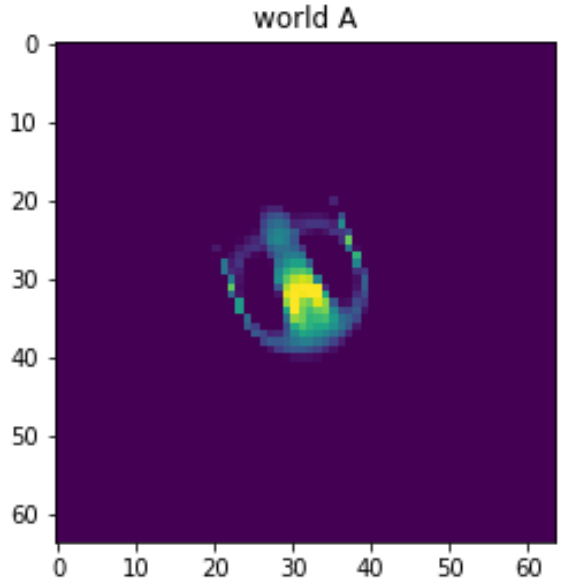


Ilustración 2. Ejemplo de mundo con estados iniciales. Obtenida del notebook original de Lenia

En segundo lugar, se definen las reglas de actualización para cada celda. Estas reglas no están definidas explícitamente, como ocurre en Game of Life. El autor ha dividido el proceso de actualización en dos etapas: distribución de potencial; y distribución de crecimiento.

En la primera etapa, la distribución de potencial se puede entender como una matriz que indica para cada celda la influencia de los vecinos en el estado actual. Para calcularla, se emplean kernels de convolución que, como se mencionó anteriormente, llevan implícitas en sus características las reglas de actualización. Así, se calcula la distribución de potencial, $U^t(x)$ mediante la convolución del kernel K con el mundo A, como se muestra en la Ecuación 1. Se destaca que el autor define el kernel radialmente, con funciones unimodales, de forma que solo sea importante la distancia al centro del kernel, y sea invariante a la orientación de este. Esta aproximación caracteriza al modelo como "espacio continuo".

$$U^t(x) = K * A^t(x)$$

Ecuación 1. Distribución de potencial.

En la segunda etapa, la distribución de potencial se utiliza luego para calcular la distribución final de crecimiento, que determina el estado siguiente del autómata celular. La distribución de crecimiento $G^t(x)$ se calcula utilizando una función de crecimiento $C(x)$, como se ve en la Ecuación 2, y que devuelve valores en el intervalo [-1, 1]. Esta función es también parte de las reglas implícitas de actualización.

$$G^t(x) = C(U^t(x))$$

Ecuación 2. Distribución de crecimiento.

Finalmente, la distribución de crecimiento se agrega al estado original del mundo A para obtener la nueva generación. Sin embargo, no se agrega completamente, sino que se escala por la resolución de tiempo elegida, Δt. De esta forma, cuanto más pequeña sea Δt, más "continuo" serán los cambios, mientras que cuanto más grande sea, más rápido se producirán los cambios. Esto le daría la propiedad de "tiempo continuo". Tras esta suma, se ajustan los valores para que estén dentro del rango [0, 1].

Como resultado, cada actualización genera una nueva matriz con valores entre 0 y 1, representables en imágenes digitales (o vídeos) de un único canal RGB. La Ecuación 3 representa la ecuación de actualización del mundo.

$$A^{t\,+\Delta t}(x) \;=\; [A^t(x) \;+\; \Delta t\; G^t(x)\,]_0^1$$

Ecuación 3. Ecuación de actualización del mundo

El modelo no está limitado a un espacio de un solo canal, o a un único kernel que contenga parte de las reglas de actualización. El autor ha expandido el modelo, de forma que el espacio celular sea de múltiples canales (matrices), conectados a sus correspondientes kernel. El resultado para cada canal sería la suma de todas las convoluciones ponderadas para cada canal, mediante una matriz *h* que contenga el peso de cada kernel sobre cada canal. De esta forma, la representación visual final es interpretable a una imagen multidimensional, donde el número de canales pueden ser distintos de los 3 clásicos (RGB). La Ilustración 3 representa un esquema del proceso de actualización del mundo, con la extensión de múltiples canales y kernels.

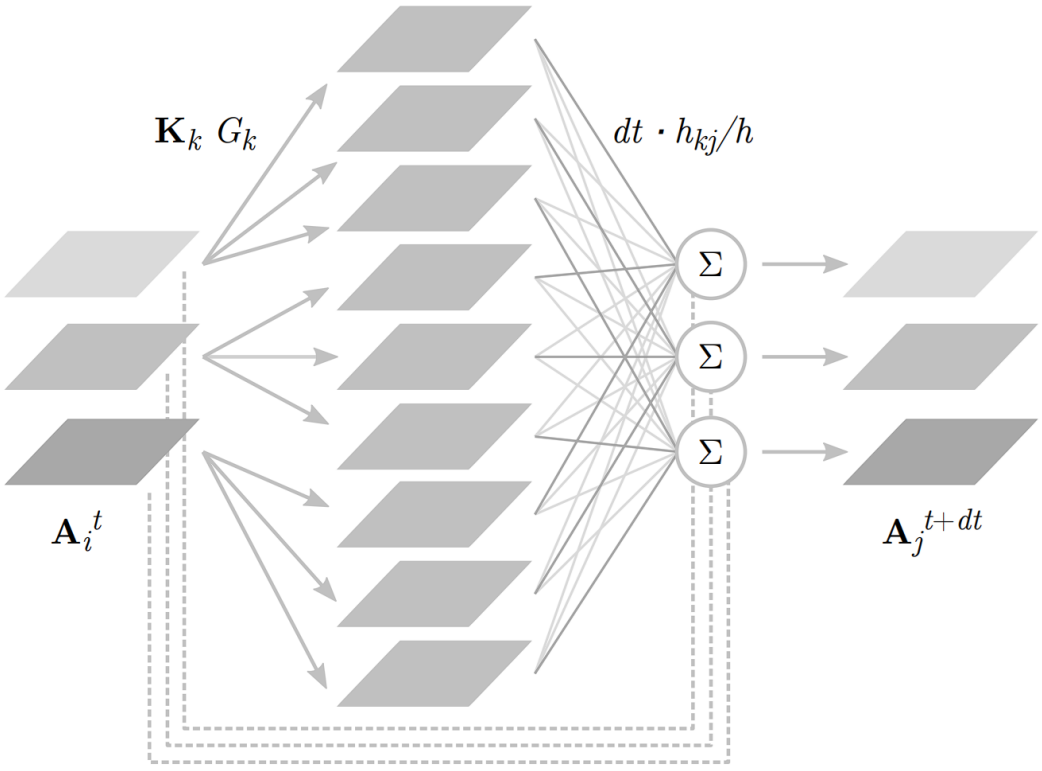


Ilustración 3. Esquema del proceso de actualización del mundo, extendido a múltiples canales y kernels. Obtenido del paper de Lenia.

Por tanto, la formulación final que generaliza Lenia para múltiples canales y kernels se puede ver en la Ecuación 4:

$$A_j^{t\ +dt} = \left[A_j^t + \Delta t \sum_{i,k} \frac{h_k}{h} G_k \left(K_k * A_i^t \right) \right]_0^1$$

Ecuación 4. Actualización del mundo, con extensión de múltiples canales y kernels

2.4. VARIANTE: FLOW LENIA

Aunque Lenia puede generar patrones complejos, las criaturas que produce solo se encuentran en un pequeño subespacio del espacio de parámetros y no son fáciles de descubrir sin algoritmos de búsqueda avanzados.

Flow Lenia (Plantec, et al., 2022) es una extensión de Lenia que introduce una restricción de conservación de masa, lo que significa que la masa total dentro del sistema se mantiene constante a lo largo del tiempo. Esta restricción facilita la aparición de patrones localizados espacialmente que se asemejan a criaturas artificiales similares a la vida. Al conservar la masa, el sistema puede exhibir comportamientos e interacciones más realistas.

En Flow Lenia, el comportamiento del sistema está determinado por dos componentes principales: el mapa de afinidad y el flujo.

El mapa de afinidad (U) es similar al mapa de crecimiento en el modelo original de Lenia. Representa la "afinidad" o atractividad de cada celda. La materia en Flow Lenia tiende a fluir hacia regiones con una afinidad más alta siguiendo el gradiente en el espacio de U. Esto significa que la materia se moverá hacia áreas donde la afinidad es más fuerte.

$$U^t = G(K * A^t)$$

Ecuación 5. Mapa de afinidad, equivalente a la distribución de crecimiento en Lenia.

El flujo (F) representa el movimiento de la materia en el sistema. La materia en cada celda se mueve según el flujo utilizando un algoritmo de seguimiento de reintegración. El algoritmo de seguimiento de reintegración es una técnica utilizada en simulaciones de autómatas celulares para rastrear partículas y conservar su masa total. Este algoritmo fue ideado y se describe en el blog de Michael Moroz (Moroz, 2020). Sin embargo, para evitar que toda la materia se acumule en una sola ubicación, el gradiente de afinidad se combina con el gradiente de concentración negativa. Esta combinación está controlada por un parámetro alfa, que determina la importancia del gradiente de concentración en relación con el gradiente de afinidad. En regiones de alta concentración, el gradiente de concentración domina, mientras que, en regiones de baja concentración, el gradiente de afinidad domina.

$$F^t = (1 - \alpha^t)\,\nabla U^t - \alpha^t \nabla A^t$$

Ecuación 6. Flujo de crecimiento, para evitar la concentración de masa.

El mecanismo de flujo en Flow Lenia permite la creación de patrones y comportamientos emergentes en el sistema. La interacción entre el mapa de afinidad y el flujo permite que la materia se mueva de manera coherente y forme estructuras complejas a lo largo del tiempo. En la Ilustración 4 se muestra una representación visual del proceso de actualización.

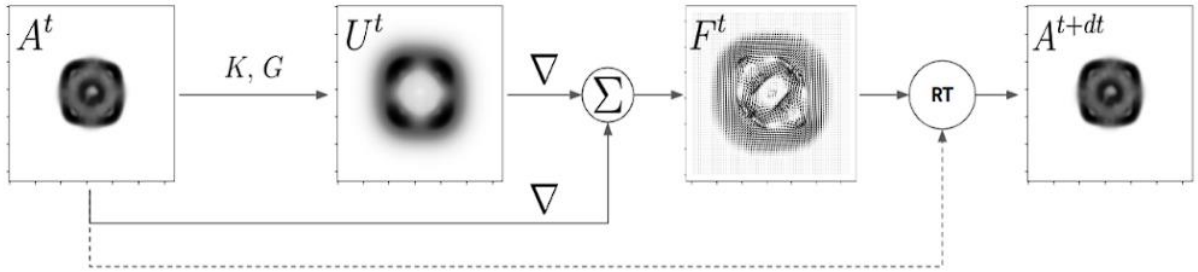


Ilustración 4. Esquema del proceso de actualización del mundo en Flow Lenia.

2.5. EJEMPLARES

Tras entender el funcionamiento de Lenia y Flow Lenia, se mostrará en esta sección algunos *ejemplares*.

El alcance que se pretende abarcar con el término “ejemplares” es el de “formas de vida” o “patrones emergentes” que se asemejan a organismos microscópicos del mundo real. En el sistema de Lenia, estos se generan como patrones autónomos complejos caracterizados por ser geométricos, metaméricos, difusos, resilientes, adaptables. Se han identificado más de 400 especies en 18 familias dentro de Lenia, muchas de las cuales se descubrieron a través de la computación evolutiva interactiva.

En primer lugar, se encuentra el ORB, uno de los primeros descubiertos por el creador. Este individuo vive en un espacio de una única matriz, es decir, se puede visualizar mediante un canal, con valores del 0 al 255. Como se ve en la Ilustración 5, en este caso es el canal rojo. El comportamiento que muestra es que tiene la capacidad de desplazarse, además de “esquivar” otros individuos cuando colisionan ligeramente.

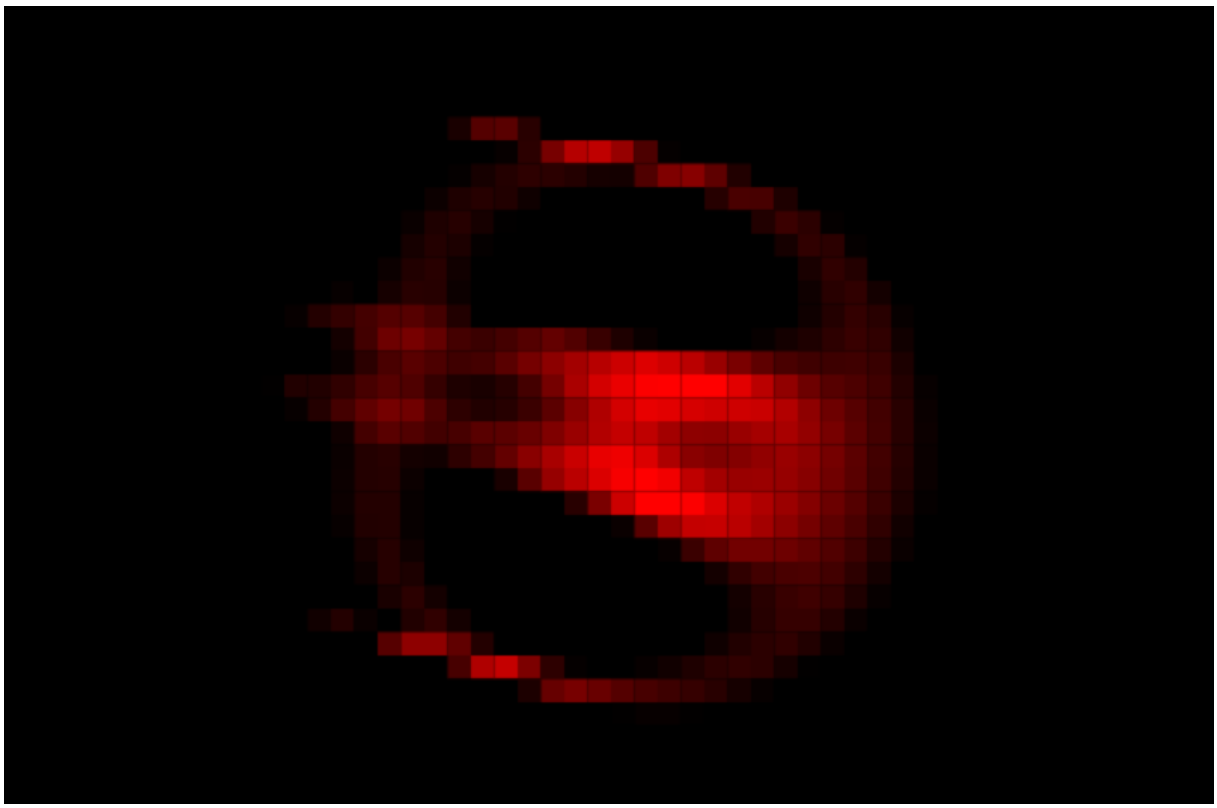


Ilustración 5. Visualización del ejemplar ORB.

El siguiente ejemplar es Aquarium. Este individuo vive en un espacio de 3 matrices, es decir, visibles mediante 3 canales. Por ello, se puede visualizar en la Ilustración 6 empleando todos los canales RGB. El comportamiento que posee es que se desplaza siguiendo una trayectoria semicircular, además de que se clona cada vez que recorre su trayectoria.

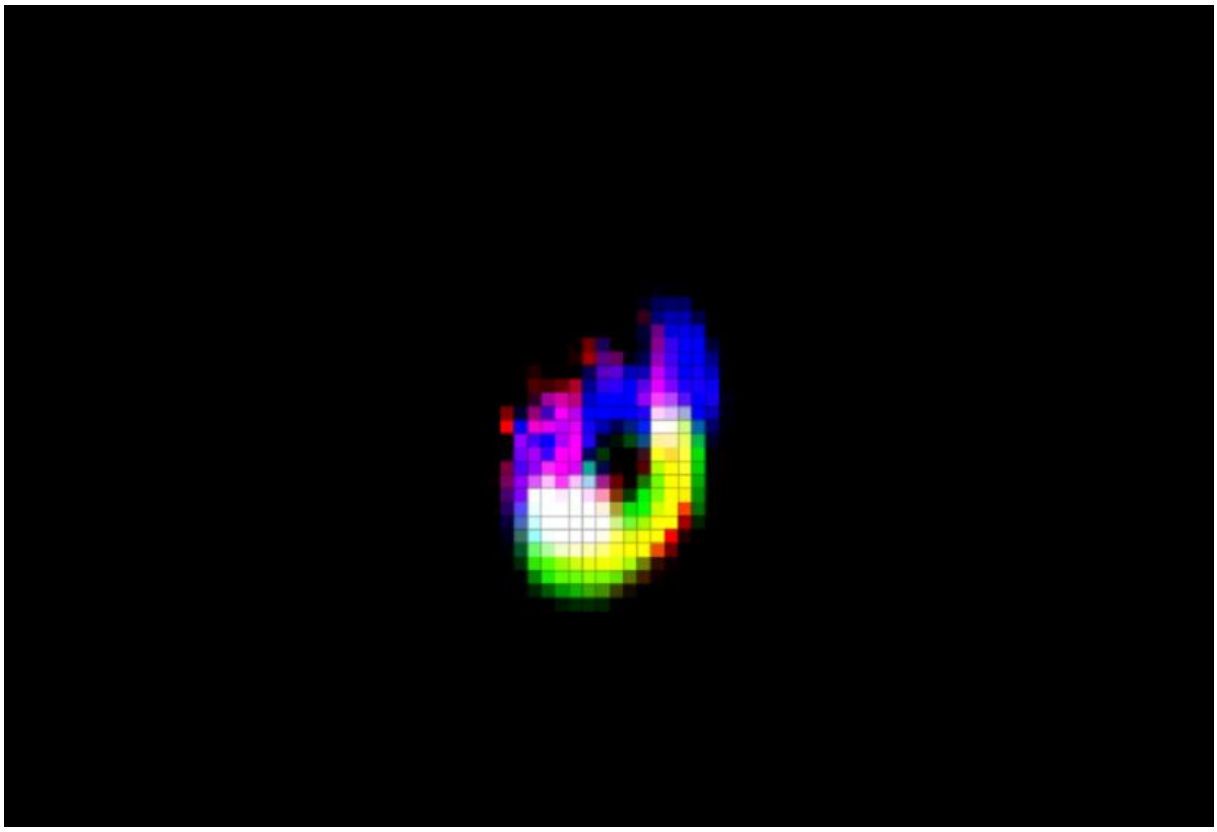


Ilustración 6. Visualización del ejemplar Aquarium.

El tercer ejemplar tiene el nombre de Emitter. Al igual que el anterior, es simulado sobre 3 canales (RGB), como se puede observar en la Ilustración 7. Su comportamiento destacable es que, además de tener la capacidad de desplazarse, emite pequeños individuos mientras lo hace. Esto es, individuos nuevos se originan del Emitter, alejándose unos pocos píxeles para luego evaporarse.

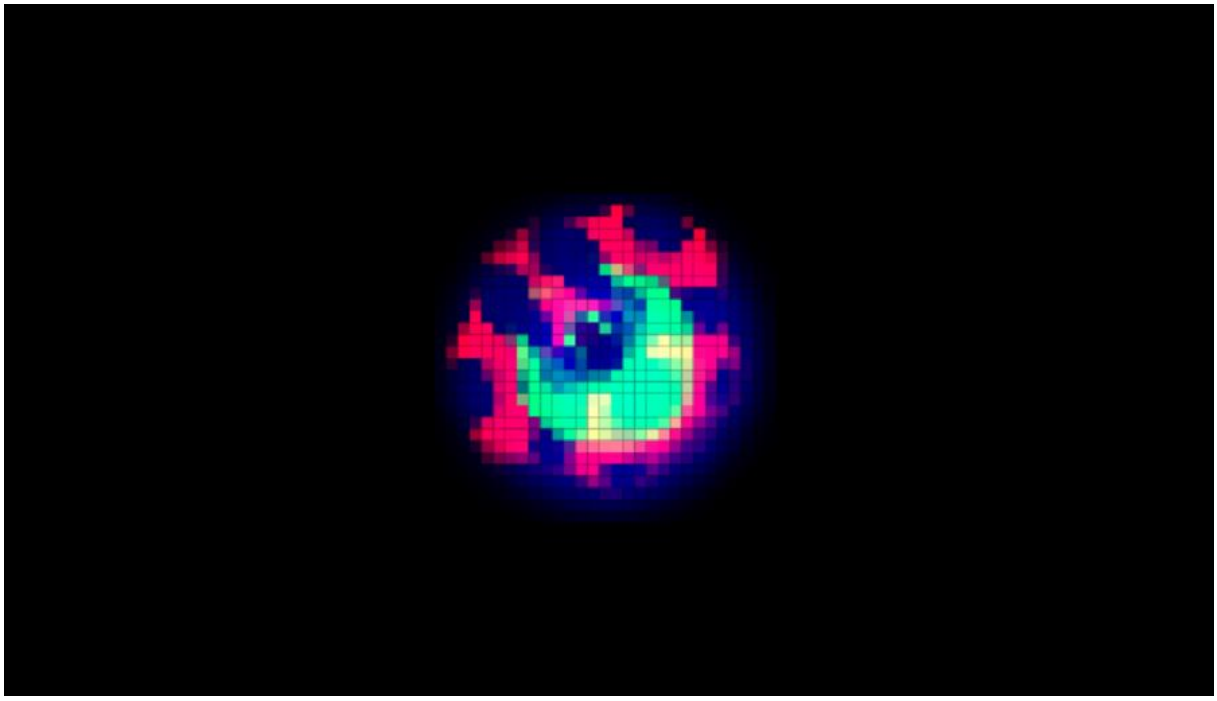
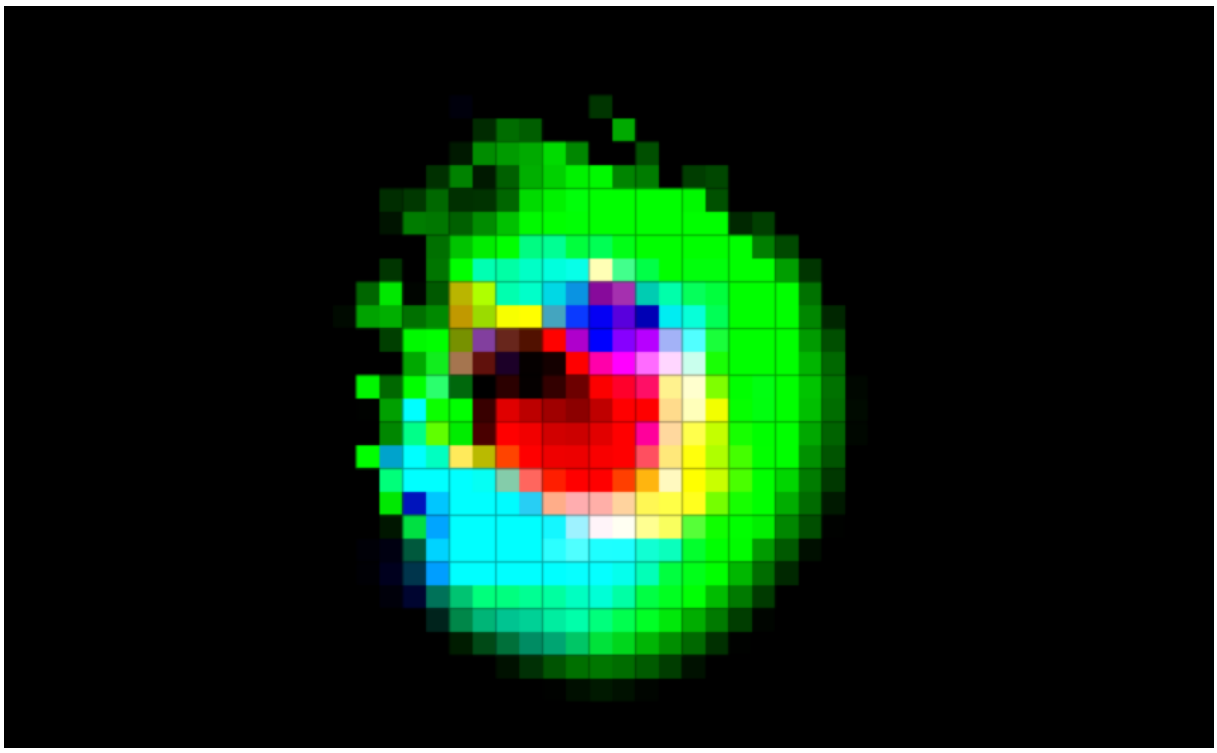


Ilustración 7. Visualización del ejemplar Emitter.

Este último ejemplar descubierto por el autor tiene el nombre de Self-Replicator. Al igual que los anteriores dos ejemplares, vive en un espacio de 3 canales. Su comportamiento característico es que tiene la capacidad de replicarse.



2.6. PROBLEMA EXISTENTE

Lenia representa un avance significativo en la investigación de Vida Artificial, principalmente debido a su capacidad única para generar patrones complejos y parecidos a la vida a través de reglas e interacciones locales simples. Su impacto se extiende más allá de los estudios teóricos. Sus aplicaciones prácticas abarcan desde gráficos por computadora y arte generativo hasta la comprensión de la dinámica cerebral. La versatilidad de Lenia lo convierte en una herramienta invaluable para científicos de diversas disciplinas, permitiéndoles explorar fenómenos complejos y formular nuevas hipótesis.

A pesar de su inmenso potencial, la falta de herramientas exhaustivas de investigación para Lenia plantea un obstáculo significativo para los investigadores. A pesar de que se encuentra disponible la implementación básica de Lenia, la ausencia de visualizadores y herramientas de análisis con interfaces sencillas limita la facilidad y eficiencia con la que los investigadores pueden explorar su dinámica. Para desatar todo el potencial de Lenia y permitir que una amplia comunidad científica aproveche su potencial, se vuelven imprescindibles herramientas exhaustivas de investigación.

Interfaces de usuario intuitivas y herramientas de visualización son vitales para facilitar la exploración intuitiva del comportamiento de Lenia. La retroalimentación visual en tiempo real y los controles interactivos pueden ayudar a los investigadores a ajustar las simulaciones, observar patrones emergentes y experimentar con diferentes parámetros. Además, las herramientas de análisis de datos eficientes capaces de cuantificar y extraer información significativa de las simulaciones de Lenia son esenciales para investigaciones científicas rigurosas.

Teniendo en cuenta lo comentado anteriormente, el software que se desarrolla en este proyecto poseerá diversas funcionalidades para poder experimentar con el sistema de Lenia. La visualización de la simulación en tiempo real es una de las principales funcionalidades. Se tendría la posibilidad configurar varios parámetros del espacio celular, como: el tamaño del incremento del tiempo; la semilla para la generación aleatoria del mundo; el sistema a emplear (Lenia o Flow Lenia). Por otro lado, también se puede configurar las características de los kernels, con la posibilidad de personalizar los parámetros de cada uno.

Luego, se podrá seleccionar ejemplares precargados de una galería para poder experimentar con estos, pudiendo acceder a sus parámetros. Además, se da la opción de guardar los

parámetros y estados de cualquier simulación en esta galería, de forma que se puedan cargar cuando se desee.

Por último, se proporcionan algunos datos que se registran de la simulación, para poder ser empleadas como herramientas de análisis. Se tendría la posibilidad de acceder a los datos de los ejemplares de manera individual.

CAPÍTULO 3. EVALUACIÓN Y MEDIDAS

Hasta esta sección, ya habría un buen entendimiento del autómata celular Lenia. Ahora, se proponen tres aspectos para evaluar comportamientos interesantes de la simulación.

3.1. MEDIDAS

En primer lugar, está el aspecto de la supervivencia. Esto es, la capacidad que tiene un individuo de seguir "vivo" en el siguiente instante de tiempo. Este comportamiento sería interesante en un autómata como Lenia, en el que no se puede garantizar cuándo un individuo morirá. Con "morir", lo que se pretende expresar es que se disuelve hasta desaparecer.

En segundo lugar, se eligió el aspecto de la reproducción. Cuando aparece (nace) un nuevo individuo en el mundo, podría tener un aspecto distinto al resto. En este sentido, se hace referencia al grado de similitud que tienen los nuevos individuos con otros individuos.

Por último, está el aspecto de la morfología. Los individuos podrían cambiar de estructura a medida que pasa el tiempo. Lo que se pretende con esto es medir la evolución morfológica en el tiempo. Es decir, el grado de similitud de un individuo en un estado determinado, con su estado inicial.

3.2. SUPERVIVENCIA

Para medir la supervivencia de un individuo, se decidió usar la variación de densidad como indicador. Es decir, la diferencia de densidad entre un estado determinado y el anterior. Para ello, es necesario tener la densidad del individuo. Se podría entender que si el individuo se está muriendo (evaporando o desapareciendo), con bastante probabilidad su densidad estaría disminuyendo.

Teniendo en cuenta los aspectos matemáticos de Lenia explicados en el CAPÍTULO 2: CONTEXTO, se obtiene densidad como la razón entre la masa total y el volumen que ocupa. Como se ve en la Ecuación 7, la masa total es la suma de todos los estados del individuo:

$$m = \int A_i(x)dx$$

Ecuación 7. Masa total

donde A_i es la matriz que describe el individuo i, accesible a las posiciones representadas por x. Por otra parte, el volumen es el número de estados positivos, como se muestra en la Ecuación 8.

$$V_i = \int_{A_i > 0} dx$$

Ecuación 8. Volumen del individuo i

donde V_i es el volumen del individuo i. Por tanto, la densidad del individuo i se calcula como su masa dividida por su volumen, como se ve en la Ecuación 9.

$$\rho_i = m \,/\, V_i$$

Ecuación 9. Densidad del individuo i

Donde ρ_i es la densidad del individuo i. Así, solo queda calcular la diferencia con la anterior densidad, como se muestra en la Ecuación 10.

$$\Delta\rho_i = \rho_i^t - \rho_i^{t-1}$$

Ecuación 10. Variación de densidad del individuo i

Así, se puede entender que, si la variación de densidad $\Delta\rho_i$ es negativa, el individuo está muriendo (desapareciendo). Si es nula (0) se mantiene constante. Y si es positiva, está creciendo.

En el CAPÍTULO 6: EXPERIMENTOS Y RESULTADOS se entrará más en detalle de cómo se muestran estos datos para cada individuo.

3.3. REPRODUCCIÓN

Para medir el aspecto de la reproducción mencionado anteriormente, se decidió realizar una comparación entre el estado de un individuo y otro del resto de individuos (tomando una media). De esta forma, cuando aparecen nuevos individuos, se tendría una métrica que indica el grado de similitud que este tiene con el resto, a lo largo de su vida.

La correlación cruzada se puede utilizar para medir la similitud entre dos imágenes (ProgrammerClick.com, s.f.). Se comparan las intensidades de los píxeles en las dos imágenes y se calcula la correlación cruzada entre ellas. Un valor alto de correlación cruzada indica una alta similitud entre las imágenes. Se calcula la correlación cruzada entre las características de las dos imágenes. La correlación cruzada más alta indica la mejor coincidencia entre las características.

Por ello, esta es la técnica que se decidió usar en el proyecto para la comparación de similitud. Como se verá a continuación, esta técnica es también la que se empleará con la siguiente métrica.

3.4. MORFOLOGÍA

Para medir la evolución morfológica del individuo, se procede de una manera similar al de la reproducción. En este caso, se calcula el grado de similitud entre el estado actual del individuo y su estado inicial, es decir, cuando apareció o fue detectado por primera vez.

Si el grado de similitud entre el individuo actual y su estado inicial cuando se detectó es alta (muy similares) se podría concluir que no ha evolucionado mucho morfológicamente. Si el fenómeno es contrario, es decir, si es muy diferente a su estado inicial, se podría entender que el individuo ha ido cambiando a lo largo del tiempo de su existencia.

Por otra parte, se puede llegar a más conclusiones si se tiene en cuenta el historial de esta métrica para un individuo. Si la similitud del individuo con su estado inicial es alta en algunos instantes de tiempo, y bajas en otras, se trataría de un comportamiento interesante para la configuración de ese modelo. Si, además, fuese periódica, sería un patrón de comportamiento aún más interesante.

Como se mencionó en la métrica del CAPÍTULO 3.3. REPRODUCCIÓN, se emplea la misma técnica de correlación cruzada para realizar la comparación entre las imágenes de los individuos.

CAPÍTULO 4. METODOLOGÍA Y PLANIFICACIÓN DEL PROYECTO

4.1. METODOLOGÍA

En este capítulo se explicará la metodología seguida. Este proyecto se desarrolló siguiendo una metodología de desarrollo incremental. Específicamente, se ha seguido la metodología de KanBan.

Kanban (ProjectManger.com, s.f.) es una metodología de gestión de proyectos que se basa en la visualización y seguimiento de las tareas a través de un tablero. Se basa en algunos principios clave que la diferencian de otras metodologías ágiles. Estos principios incluyen la garantía de calidad, la reducción del desperdicio, la mejora continua y la flexibilidad. En el contexto de proyectos software individuales, es especialmente útil debido a su enfoque en la visualización clara del estado de los proyectos, la gestión efectiva de las tareas, la adaptabilidad a cambios y la mejora continua.

Esta metodología se llevó a cabo sobre la herramienta Trello. Como se explica más adelante (CAPÍTULO 5.5. HERRAMIENTAS), Trello es una aplicación de gestión de proyectos basada en el método Kanban. Permite organizar tareas y colaborar de forma eficiente en proyectos tanto a nivel profesional como personal. A través de tableros virtuales compuestos de listas de tareas en forma de columnas, Trello facilita la visualización y seguimiento del progreso de las tareas.

Los estados empleados son 5:

1. To Do: En este estado se encuentran las tareas que aún no han sido abordadas. Representa las tareas pendientes que deben ser realizadas en el proyecto.

2. Sprint: Se utiliza para representar las tareas que están planificadas para un sprint o ciclo de desarrollo específico. Un sprint es un período de tiempo definido en el que se

trabajan un conjunto de tareas. En esta columna se encuentran las tareas que se abordarán en el sprint actual.

3. In Progress: En este estado se encuentran las tareas que están actualmente en progreso. Representa las tareas en las que se está trabajando activamente.

4. To Verify: Este estado se utiliza para representar las tareas que han sido completadas y están listas para ser revisadas o verificadas. Representa las tareas que han sido terminadas y están esperando una revisión de calidad o aprobación.

5. Done: En este estado se encuentran las tareas que han sido completamente finalizadas y aprobadas. Representa las tareas que han pasado la revisión y están listas para ser entregadas o marcadas como completadas.

En la Ilustración 8 se muestra las columnas empleadas con algunas de las tareas del proyecto.

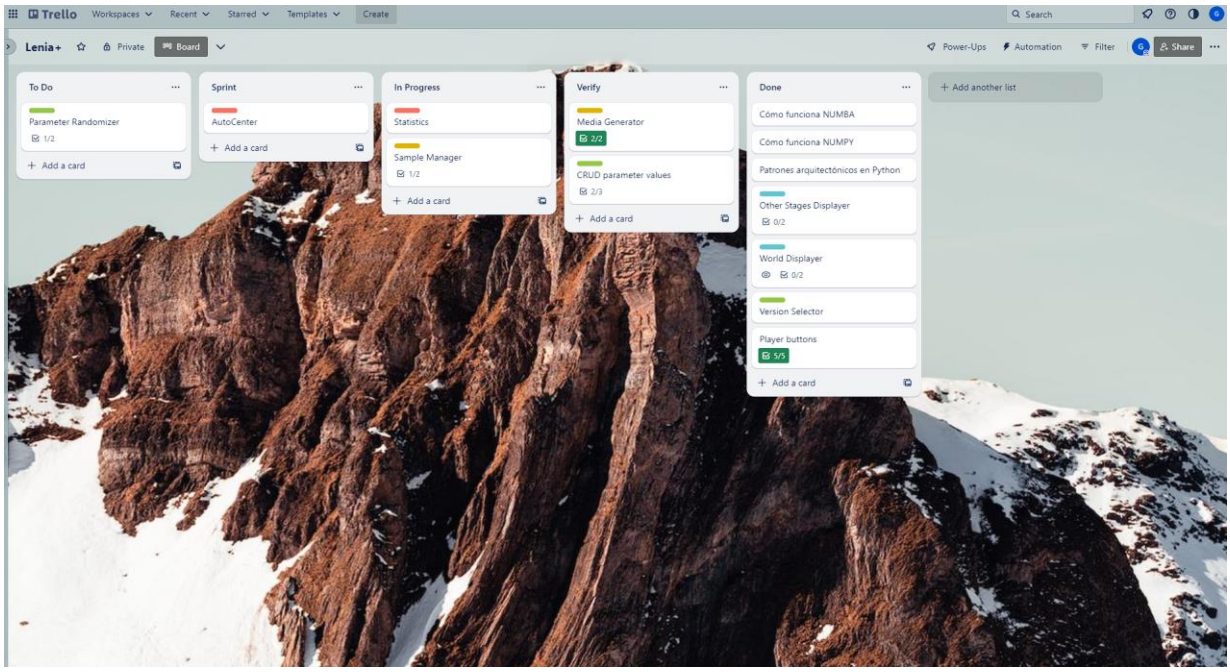


Ilustración 8. Captura de tablas de Trello para el desarrollo del proyecto.

4.2. PLANIFICACIÓN

En este capítulo se explicará la planificación seguida. Los *sprints* se fijaron a un tamaño de 3 semanas, en el que se seleccionaban las tareas con mayor prioridad. Por otra parte, se realizaban reuniones entre el alumno y el tutor cada 1 o 2 semanas, de forma que se estuviera informando al tutor respecto los avances y problemas encontrados. Además, se indicaban en estas reuniones las nuevas funcionalidades a añadir.

Las primeras tareas consistían en tener un entendimiento con suficiente profundidad en cuanto al modelo de Lenia. Para ello, se necesitaba realizar una exploración intensiva de su funcionamiento. Por tanto, era necesario que las primeras tareas fueran comprender el *paper* de Lenia (Chan, 2018). Además de estos, estaban publicados notebooks de Python en los que se llevaba a cabo algunas demostraciones funcionales del modelo, por lo que también era importante entender la implementación.

Seguidamente, se procedió a implementar el software. Las primeras horas de implementación se dedicaron a programar un prototipo del software. Para esta fase, también se debía tener en cuenta de cara al futuro los aspectos de la arquitectura. Tras esto, se empezó a desarrollar las tareas para el resto del prototipo, enfocándose en los cálculos que se debían programar.

Las siguientes tareas de la fase de implementación se centraban en la interfaz gráfica para el usuario. Una pequeña parte de este tiempo se requirió para explorar las formas más sencillas y flexibles de implementar la interfaz, seleccionando finalmente las tecnologías frontend clásicas (HTML, CSS, JavaScript).

Luego, se procedió a realizar las experimentaciones. En esta fase, se tomaron diferentes ejemplares y se simularon en tiempo real, pudiéndose observar los datos estadísticos que se obtenían. Con estos datos registrados se pretendía interpretar las características que mostraban los individuos de la simulación.

En la última fase, se dedicó el tiempo restante a redactar la memoria que contendría el proceso de desarrollo de este Proyecto de Fin de Grado.

CAPÍTULO 5. DESARROLLO DEL PROYECTO

Tras entender el autómata Lenia y el propósito de esta solución software, en este capítulo se detallará el desarrollo de este proyecto, destacando los aspectos más relevantes.

5.1. TECNOLOGÍAS DE PROGRAMACIÓN

Al tratarse de un simulador en tiempo real, y teniendo en cuenta el alcance de un Trabajo de Fin de Grado, se decidió emplear Python como principal lenguaje de programación en el backend. Por otra parte, se decidió no emplear librerías de Python para las interfaces del frontend, ya que estaban bastante limitadas en cuanto a diseño y flexibilidad. Aunque las librerías de Python también pueden ser utilizadas para crear interfaces gráficas, se decidió hacer uso de las tecnologías frontend clásicas, es decir, HTML, CSS y JavaScript. Estas ofrecen ventajas adicionales en términos de compatibilidad, flexibilidad y evolución tecnológica.

La comunicación entre frontend y backend se realiza mediante una reciente librería que permite el intercambio de datos entre Python y JavaScript. Más adelante se explicará esta conexión. A continuación, se describirán las ventajas de estas decisiones.

5.1.1. BACKEND

Python es una opción sólida para proyectos de simulaciones en tiempo real debido a su facilidad de aprendizaje, rendimiento, bibliotecas especializadas, integración con tecnologías de computación distribuida, facilidad para la colaboración y portabilidad. Algunas ventajas de utilizar Python en este tipo de proyectos incluyen:

1. Facilidad de aprendizaje: es conocido por su sintaxis clara y fácil de leer, lo que facilita su adopción por parte de los desarrolladores.

2. Rendimiento: Aunque Python es un lenguaje interpretado, su rendimiento es comparable al de otros lenguajes compilados como C++ o Java. Además, Python cuenta con bibliotecas y herramientas de optimización que pueden mejorar aún más su rendimiento en proyectos de simulaciones.

3. Bibliotecas especializadas: cuenta con una amplia gama de bibliotecas especializadas en cálculos numéricos y simulaciones, como Ñampí, SciPy y Pandas. Estas bibliotecas facilitan el desarrollo de proyectos de simulaciones y ofrecen funcionalidades avanzadas para el análisis de datos y la resolución de ecuaciones matemáticas.

4. Integración con tecnologías de computación distribuida: puede ser utilizado en combinación con tecnologías de computación distribuida, como la computación en malla o la computación en cuadrícula, para dividir problemas complejos en fragmentos más pequeños y realizar cálculos en paralelo en múltiples ordenadores. Esto puede ser especialmente útil en proyectos de simulaciones que requieren una gran cantidad de recursos computacionales.

5. Facilidad para la colaboración: es popular en la comunidad de desarrolladores y científicos, lo que facilita la colaboración en proyectos de simulaciones y la búsqueda de ayuda y recursos en línea.

6. Portabilidad: es compatible con múltiples sistemas operativos y plataformas, lo que facilita la implementación de proyectos de simulaciones en diferentes entornos.

5.1.2. FRONTEND

Las tecnologías frontend clásicas (HTML, CSS y JS) ofrecen ventajas en términos de simplicidad, compatibilidad, control, flexibilidad, integración y rendimiento en comparación con otros enfoques de programación de frontend. Estas ventajas hacen que estas tecnologías sigan siendo una opción popular y efectiva para el desarrollo de interfaces de usuario en aplicaciones web. Las ventajas más destacables son varias:

1. Amplia adopción y compatibilidad: son tecnologías ampliamente utilizadas y compatibles con la mayoría de los navegadores web. Esto significa que una interfaz diseñada con estas tecnologías puede funcionar en diferentes plataformas y dispositivos sin necesidad de adaptaciones adicionales. Además, la comunidad de desarrollo web está en constante evolución, lo que implica que hay una gran cantidad de recursos, herramientas y frameworks disponibles para ayudar en el diseño y desarrollo de interfaces.

2. Separación de responsabilidades: Al utilizar HTML, CSS y JS, se sigue el principio de separación de responsabilidades en el diseño de la interfaz. HTML se utiliza para estructurar el contenido, CSS para definir los estilos y JS para agregar interactividad y funcionalidad. Esta separación permite un desarrollo modular y mantenible, ya que cada tecnología se enfoca en un aspecto específico de la interfaz.

3. Flexibilidad y personalización: ofrecen una amplia gama de opciones para personalizar la apariencia y comportamiento de una interfaz. CSS permite definir estilos detallados, como colores, fuentes, espaciado, alineación, entre otros. JS proporciona la capacidad de agregar interacciones dinámicas, como animaciones, validaciones de formularios y manipulación del DOM. Esta flexibilidad brinda a los diseñadores y desarrolladores la posibilidad de crear interfaces únicas y atractivas.

4. Integración con servicios web: Al utilizar HTML, CSS y JS en el diseño de una interfaz, se facilita la integración con servicios web y APIs. Esto es especialmente útil de cara al futuro, para cuando la aplicación requiera de la interacción con servicios externos.

5. Evolución constante: están en constante evolución y actualización. Se introducen nuevas características y mejoras regularmente, lo que permite aprovechar las últimas tendencias y avances en el diseño de interfaces. Además, la comunidad de desarrollo web está activa y colaborativa, lo que facilita el intercambio de conocimientos y la resolución de problemas.

5.2. ARQUITECTURA DE LA SOLUCIÓN

En términos simples, la arquitectura de software se refiere a la estructura y comunicación de las diferentes partes de un software. Es como los planos de un edificio, donde se define cómo se distribuyen las habitaciones, los materiales utilizados y cómo se conectan los diferentes elementos. De manera similar, en la arquitectura de software se definen los componentes, las interfaces y las interacciones entre ellos.

Cuando se trata de software de simulación en tiempo real, la arquitectura adecuada es esencial para garantizar un rendimiento óptimo y una respuesta rápida. La arquitectura en capas o la arquitectura orientada a servicios pueden ser enfoques comunes utilizados en este tipo de software. Estas arquitecturas permiten una separación clara de las responsabilidades y facilitan la escalabilidad y el mantenimiento del software, lo cual es fundamental para un software de simulación en tiempo real.

Como se podría intuir desde el principio del documento, se diferencian dos partes en la arquitectura de este proyecto: el backend, donde se computarán todos los cálculos para la simulación; y el frontend, a través del que el usuario puede interactuar, configurar y observar los resultados de la simulación. En la Ilustración 9 se muestra diagrama de clases UML realizado para el proyecto.

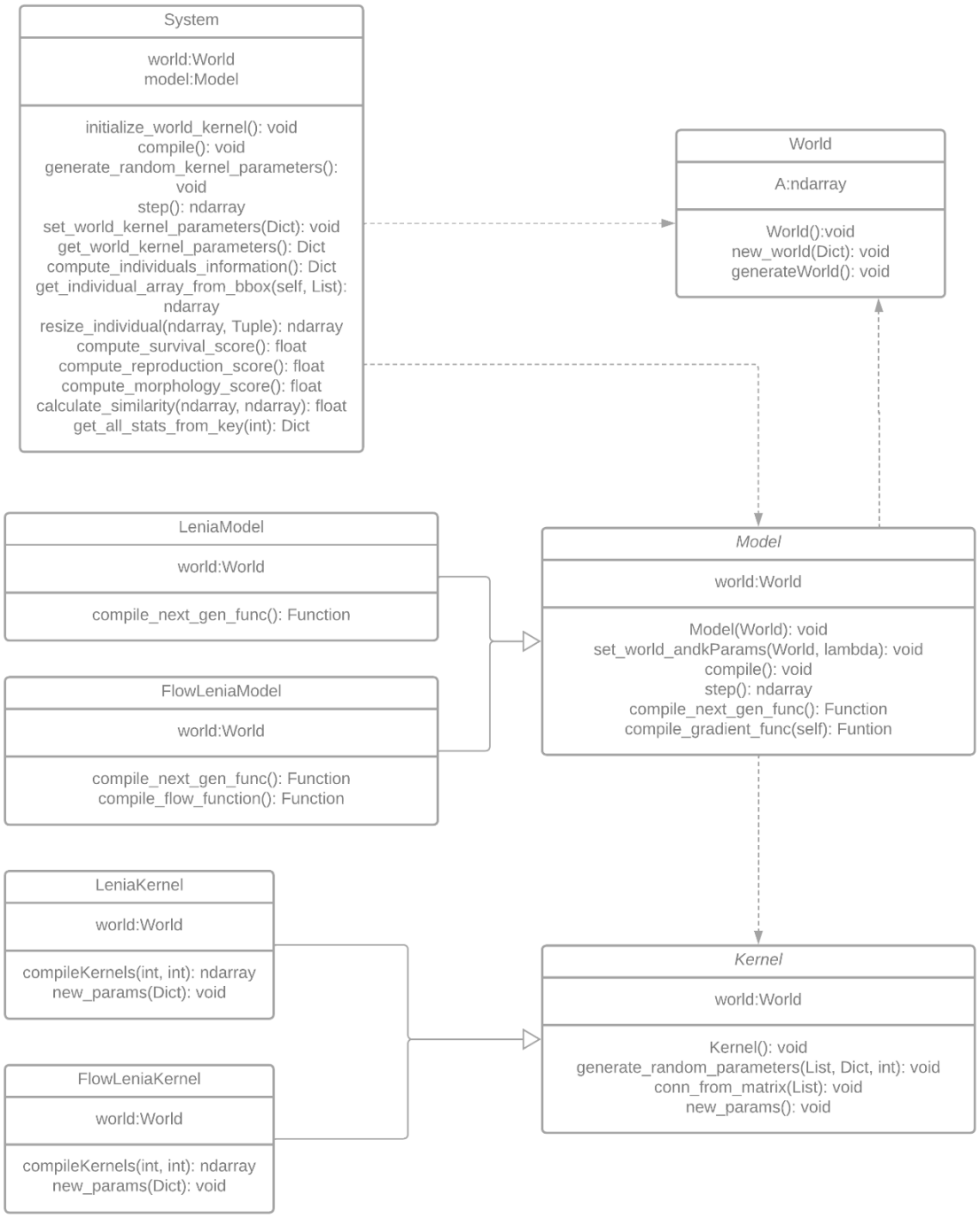


Ilustración 9. Esquema de clase de UML del proyecto.

Como se puede observar de la Ilustración 9, el patrón de diseño empleado es *Factory Method*, y se procede a explicar las razones. En primer lugar, se tendrá la clase principal “System”. Esta clase controlará y gestionará las operaciones del simulador.

Para gestionar estas operaciones, la clase principal se apoyará sobre otras dos clases llamadas “World” y “Kernel”. La clase World representa el espacio celular, donde se encuentra el estado

del mundo. Para llevar a cabo esto, se almacena el mundo en matrices de numpy, facilitando las operaciones sobre esta.

Luego, la clase Kernel sobre la que se apoya la clase principal representa un manejador de la configuración del kernel. En este sentido, como en este simulador los kernels son configurables, todos sus parámetros se almacenarán en estos objetos, para posteriormente generar las matrices correspondientes a los kernels. Como la generación de kernels para el sistema Lenia son diferentes que para las de Flow Lenia, se optó por crear dos clases hijas de Kernel. De esta forma, cada clase tiene la misma función, computando cada uno según su sistema.

Por otra parte, la clase Model maneja los cálculos necesarios para generar el nuevo mundo a partir de su estado actual. De la misma forma que ocurría en la clase Kernel, las operaciones para la generación del nuevo mundo son diferente para Lenia y Flow Lenia. Por ello, también se separan las clases en dos hijos, uno para cada sistema.

Es la clase Model la encargada de instanciar la clase de Kernel que corresponda, según el sistema seleccionado (Lenia o Flow Lenia), y es la clase System la que decide cuál se debe instanciar de la clase Model. De esta forma, quedan más claras las razones por las que se eligió el patrón de diseño Factory Method.

En segundo lugar, se describirá la arquitectura del frontend, que como se mencionó, emplea las tecnologías frontend clásicas (HTML, CSS y JavaScript). La arquitectura clásica de la tecnología frontend implica la separación de responsabilidades:

- HTML se encarga de definir la estructura y el contenido.
- CSS se encarga de definir el estilo y el diseño de los elementos HTML.
- JavaScript se encarga de agregar interactividad y funcionalidad.

Es desde JavaScript de donde se realiza la comunicación con Python. De esta forma, el frontend debe recopilar los parámetros de entrada necesarios para configurar los kernels. Además de la entrada de parámetros, desde el frontend se indica las operaciones que se deben hacer en cada momento, como el inicio de la simulación, su detención, detección de individuos, obtención de estadísticas, etc. Estos datos son procesados en el backend, y tras obtener los resultados, se devuelven al frontend para mostrarlos.

5.3. CÁLCULOS

En esta sección, se destacan los fragmentos de código Python que realizan los cálculos para la simulación, así como los datos que se extraen de esta. Se decidió destacar 3 tipos de códigos con diferentes propósitos. El primero consta de los fragmentos que se enfocan en generar los mundos y los kernels a partir de los parámetros de entrada. El segundo incluye las técnicas empleadas para la detección de individuos en el espacio celular. El tercero son los métodos que se usan para computar las medidas indicadas en anteriores apartados (supervivencia, reproducción y morfología).

En primer lugar, se describirá la generación de mundos y kernels. Como se mencionó en el CAPÍTULO 2.3. EL ESTADO DEL ARTE: LENIA, el mundo de una simulación está representada por un array de matrices. Para una pantalla, cada matriz de este array se interpreta como un canal de RGB. Así, se inicializa un array de numpy con las dimensiones deseadas en los dos primeros ejes, y el número de canales en el tercer eje. Como valores iniciales, todos los elementos son ceros, rellenando una subparte del centro con valores aleatorios o un ejemplar guardado que se haya seleccionado.

```
    def generateWorld(self):

        # Generate random world
        rand_gen = np.random.RandomState(self.seed)
        init_size = self.sX // 2
        self.A = np.zeros((self.sX, self.sY, self.numChannels))
        self.A[self.sX//2-init_size//2:self.sX//2+init_size//2,
self.sY//2-init_size //
               2:self.sY//2+init_size//2, :] = rand_gen.rand(init_size,
init_size, self.numChannels)
```

Código 1. Fragmento de código para la generación de un mundo inicial a partir de una semilla

Luego, se deben crear los kernels a partir de las características deseadas e introducidas por el usuario a través de la interfaz. Cada kernel se representa como una matriz, y se almacenan todos en un array de numpy, de forma que se facilita las operaciones con estas.

```
    def new_params(self, data):

        for k in 'rmsh':
            self.kernel_parameters[k] = np.array(data[k],
```

```
dtype=np.float64)

        for k in 'C':
            self.kernel_parameters[k] = np.array(data[k], dtype=np.int64)
        self.kernel_parameters['B'] = data['B']
        self.kernel_parameters['a'] = data['a']
        self.kernel_parameters['w'] = data['w']
        self.kernel_parameters['T'] = data['T']
        self.n_kernels = len(self.kernel_parameters['r'])
```

Código 2. Fragmento de código para establecer nuevos parámetros de los kernels

Teniendo el mundo y los parámetros, se procede a realizar la simulación. Como se dispone de dos modelos diferentes (Lenia y Flow Lenia), ambas necesitan de operaciones distintas. En el Código 3 se muestra la computación para Lenia, basada en el código proporcionado por el autor de Flow Lenia, que se encuentra en un notebook de Python, en la herramienta Colaboratory de Google.

```
        def next_gen_func(A : np.asarray):

            fourier_world = [ jnp.fft.fft2(A[:,:,c]) for c in
range(self.world.numChannels) ]

            potential_distribution = [
np.real(jnp.fft.ifft2(fourier_kernel * fourier_world[c0])) for
fourier_kernel, c0 in zip(self.fourier_kernels,
self.kernel_parameters.kernel_parameters["C"]) ]

            growth_distribution = [ self.growth_function(u,
self.kernel_parameters.kernel_parameters['m'][k],
self.kernel_parameters.kernel_parameters['s'][k]) * 2 - 1 for u, k in
zip(potential_distribution,
range(len(self.kernel_parameters.kernel_parameters['m']))) ]

            Hs = [sum(self.kernel_parameters.kernel_parameters['h'][k] *
g for g, k in zip(growth_distribution,
range(len(self.kernel_parameters.kernel_parameters['m'])))
                      if k in
self.kernel_parameters.kernel_parameters['T'][c1]) for c1 in
range(A.shape[2])]

            world_channels = [ jnp.clip(A[:,:,cA] + 1/self.world.dt * H,
0, 1) for cA,H in zip(range(A.shape[2]),Hs) ]
            new_world = jnp.dstack(world_channels)
            return new_world
```

Código 3. Fragmento de código para computar Lenia, basado en Notebook de demostración de Lenia

En el Código 4 se muestra la implementación para la computación de Flow Lenia, también basada en el código del mismo autor, en el notebook de Colaboratory.

```
        def next_gen_func(world : np.asarray):

            fourier_world = jnp.fft.fft2(world, axes=(0,1))

            fourier_world_kernel = fourier_world[:, :,
self.kernel_parameters.kernel_parameters['C']]

            potential_distribution =
jnp.real(jnp.fft.ifft2(self.fourier_kernels * fourier_world_kernel,
axes=(0,1)))

            affinity = self.growth_function(
                potential_distribution,
                self.kernel_parameters.kernel_parameters['m'],
                self.kernel_parameters.kernel_parameters['s']
            ) * self.kernel_parameters.kernel_parameters['h']

            H = jnp.dstack([ affinity[:, :,
self.kernel_parameters.kernel_parameters['T'][c]].sum(axis=-1)
                             for c in range(self.world.numChannels) ])

            flow_distribution = self.gradient_func(H)

            d_world = self.gradient_func(world.sum(axis = -1, keepdims =
True))

            alpha = jnp.clip((world[:, :, None, :] /
self.world.theta)**2, .0, 1.)

            flow_distribution = flow_distribution * (1 - alpha) - d_world
* alpha

            moved_coordinates = self.pos[..., None] + self.world.dt *
flow_distribution

            new_world = self.flow_func(
                self.rollxs,
                self.rollys,
                world,
                moved_coordinates
            ).sum(axis = 0)

            return new_world
```

Código 4. Fragmento de código para computar Flow Lenia, basado en Notebook de demostración de Flow Lenia

En segundo lugar, se describe cómo se realiza la detección de individuos en un mundo. La mayor parte del espacio tiene el valor cero o nulo (visible como el color negro), habiendo algunas zonas con estados no nulos. Se considerará estas zonas no nulas como un individuo cuando se forma una "isla". Es decir, un grupo de estados no nulos conectados se considerará como un único individuo. Por tanto, el objetivo es detectar estas "islas".

Con este objetivo, se emplean funciones integradas en las librerías de Numpy y Scipy para aprovechar su eficiencia. Por tanto, primero se realiza una búsqueda de estas "islas" (individuos) en la simulación mediante la función *find_objects()*, de la librería se Scipy, que devuelven los bounding box de los individuos detectados. Luego, para obtener los centros de masa de cada uno, se emplea la función *center_of_mass()*, también de Scipy.

```
    def compute_individuals_information(self):

        # Minimum size of individual to being detected
        min_size = 10 * 2

        # Sum the values along the last axis
        summed_world = np.sum(self.world.A, axis=-1)

        # Threshold the array
        mask = np.where(summed_world > 0, True, False)

        # Label the connected regions
        self.labels, num_features = ndimage.label(mask)

        # Count the number of non-zero values in each group
        counts = np.bincount(self.labels.ravel())

        # Find the bounding box of each region
        slices = ndimage.find_objects(self.labels)

        # Calculate length of each bounding box
        individuals_bbox = [tuple(slice_interval.stop -
slice_interval.start
                          for slice_interval in slice)
                          for slice in slices]

        # Filter out regions that don't meet the minimum size threshold
        individuals_bbox = [(length[0] / self.world.A.shape[0], length[1]
/ self.world.A.shape[1])
                    for i, length in enumerate(individuals_bbox)
                    if np.bincount(self.labels.ravel())[i+1] >= min_size]

        # Find the center of mass of each connected region
        new_centroids = np.array([ndimage.center_of_mass(summed_world,
labels=self.labels, index=i)
                          for i in range(1, num_features+1) if counts[i] >=
```

```
min_size])

        new_centroids = (new_centroids / self.world.A.shape[0]).tolist()
```

Código 5. Fragmento de código para computar bounding box y centroide de individuos.

Seguidamente, esta información de cada individuo (bounding box y centroide) se almacenan en diccionarios, siendo la clave de cada uno un ID que se les asocia y que es generado cuando se detecta.

Por último, se procederá a explicar cómo se obtienen las medidas mencionadas para las gráficas.

La primera medida es la *supervivencia*. Para ello, se recorre cada individuo detectado, y que están guardados en un diccionario. Como se vio en el CAPÍTULO 3.2. SUPERVIVENCIA, para calcular la variación de densidad, es necesario tener la densidad del instante anterior y el actual. La densidad anterior se guarda en otro diccionario, que se actualiza con las densidades nuevas densidades calculadas al final de este proceso. Luego, para calcular la densidad actual, se necesita la masa y volumen actual. Tratándose de individuos que ocupan un array de matrices, se considera la masa como la suma de todos los estados de ese espacio que ocupa el individuo. Para el volumen se considerará el número total de elementos que ocupa ese individuo. Así, solo quedaría dividir la masa y el volumen, y restando el resultado con la densidad anterior.

```
    def compute_survival_score(self):

        density_scores = {}

        for k, bbox in self.centroids.items():

            indiv = self.get_individual_array_from_bbox(bbox)

            total_mass = np.sum(indiv)
            total_volume = np.count_nonzero(indiv)

            if total_volume == 0: total_volume = jnp.asarray([1])
            density_scores[k] = (total_mass / total_volume).item()

            self.survival_scores[k] = density_scores[k] -
self.previous_survival_scores[k]
            self.previous_survival_scores[k] = density_scores[k]
```

```
        return density_scores
```

Código 6. Fragmento de código para computar puntuación de supervivencia

La segunda medida es la *reproducción*. Como se vio en el CAPÍTULO 3.3. REPRODUCCIÓN, se compara el estado actual del individuo con otro individuo del mundo. Se podría realizar una media de la comparación con todos los individuos, pero para optimizar los cálculos, se realiza una comparación con otro individuo aleatorio, y se realiza una media con la anterior acumulación de esta medida. De esta forma, a largo plazo se habría comparado con bastantes individuos, al mismo tiempo que las comparaciones más recientes tienen más peso en esta medida. Para realizar la comparación y obtener una puntuación de similitud se creó un método que se explicará al final de esta sección, al que se le pasan las matrices que contienen a los individuos, y devuelve esta puntuación de similitud. Antes de realizar la comparación, es necesario que las dos "imágenes" de individuos tengan las mismas dimensiones. Por ello, se implementó otro método para que realice la redimensión mediante interpolación lineal, proporcionada por métodos integrados en las librerías de Numpy y Scipy.

```
    def compute_reproduction_score(self):

        for k, bbox in self.centroids.items():

            indiv = self.get_individual_array_from_bbox(bbox)

            keys = [key for key in self.centroids.keys() if key != k]

            if len(keys) == 0: continue

            neighbour =
self.get_individual_array_from_bbox(self.centroids[random.choice(keys)])

            neighbour_resized = self.resize_individual(neighbour,
indiv.shape)

            similarity_score = self.calculate_similarity(indiv,
neighbour_resized)

            self.reproduction_scores[k] = (self.reproduction_scores[k] +
similarity_score) / 2

        common_keys = set(self.centroids.keys()) &
set(self.reproduction_scores.keys())

        self.reproduction_scores = {key: self.reproduction_scores[key]
```

```
for key in common_keys}

        return self.reproduction_scores
```

Código 7. Fragmento de código para computar puntuación de reproducción.

La tercera medida es la *morfología*. Como se vio en el CAPÍTULO 3.3. MORFOLOGÍA, se realiza una comparación entre el estado actual del individuo y su estado inicial. Se destaca que cuando un nuevo individuo es detectado, su estado inicial se guarda en un diccionario. Por tanto, teniendo en cuenta que también se tiene el estado actual, solo se necesitaría obtener una puntuación de similitud de la misma forma que se realiza en la medida de la *reproducción*, llamando al método creado para esta comparación.

```
    def compute_morphology_score(self):

        scores = {}

        for k, bbox in self.centroids.items():

            indiv = self.get_individual_array_from_bbox(bbox)

            original = self.individuals_initial_state[k]
            resized_target = self.resize_individual(indiv,
original.shape)

            similarity_score = self.calculate_similarity(original,
resized_target)

            scores[k] = similarity_score

        self.morphology_scores = scores
```

Código 8. Fragmento de código para computar puntuación de morfología

El método de comparación empleado se basa en la correlación cruzada, como se indicó en el CAPÍTULO 3.3. REPRODUCCIÓN. Antes de calcular la similitud, se normalizan las matrices de los dos individuos. Se toma ventaja del método *np.correlate()*, que computa la correlación cruzada entre los dos individuos, que se introducen como imágenes. Del resultado se toma el máximo valor como la puntuación de similitud.

```
    def calculate_similarity(self, image1, image2):

        # Convert arrays to JAX DeviceArrays
```

```
        image1 = jnp.asarray(image1)
        image2 = jnp.asarray(image2)

        # Check if the standard deviation is zero
        if jnp.std(image1) == 0 or jnp.std(image2) == 0:

            return 0.0  # Return a default similarity value when there is
zero variance

        # Normalize the arrays
        arr1_norm = (image1 - jnp.mean(image1)) / jnp.std(image1)
        arr2_norm = (image2 - jnp.mean(image2)) / jnp.std(image2)

        # Compute the cross-correlation
        cross_corr = jnp.correlate(arr1_norm.flatten(),
arr2_norm.flatten(), mode='same')

        # Compute the similarity
        similarity = jnp.max(cross_corr) / (jnp.linalg.norm(arr1_norm) *
jnp.linalg.norm(arr2_norm))

        score = similarity.item()
        if type(score) != float:
            score = 0.0

        return score
```

Código 9. Fragmento de código para computar similitud.

5.4. INTERFAZ

Es conocido que JavaScript y Python suelen desarrollarse en ámbitos distintos. JavaScript se centra en el desarrollo web y las aplicaciones móviles, mientras que Python tiene aplicaciones más diversas, incluyendo el desarrollo web y las aplicaciones científicas.

Como en este proyecto software se requiere comunicar JavaScript con Python, se decidió solucionarlo mediante la librería Eel. Eel es una librería de Python y JavaScript que permite crear aplicaciones de escritorio con interfaces gráficas utilizando tecnologías web como HTML, CSS y JavaScript.

La forma en que Eel funciona es estableciendo una comunicación bidireccional entre el código de Python y el código de JavaScript. Esto significa que se pueden llamar funciones de Python desde JavaScript y viceversa. Para lograr esto, Eel utiliza un servidor web local que se ejecuta en el equipo del usuario y permite la comunicación entre los dos lenguajes. Al utilizar Eel, se pueden crear interfaces gráficas interactivas utilizando tecnologías web familiares como HTML, CSS y JavaScript, y luego comunicarse con el código de Python para realizar las operaciones de cómputo de la simulación.

Eel utiliza promesas y callbacks para facilitar la comunicación entre JavaScript y Python. Esto permite pasar objetos JSON u otros tipos de datos de una función de Python a JavaScript y viceversa, a través de una promesa devuelta por la función de Python y manejada por Eel. Esta forma de comunicación asincrónica y eficiente es fundamental para lograr una integración exitosa de JavaScript y Python.

Una vez que se ha justificado la importancia de la comunicación entre JavaScript y Python, así como su solución, se procederá a destacar los aspectos más importantes de la interfaz gráfica y el proceso de su diseño.

En primer lugar, se destaca que los diseños se realizaron mediante la herramienta web Figma. El software consta principalmente de una única vista con tres paneles: un panel superior para las operaciones básicas del simulador; un gran panel izquierdo donde se puede observar el estado del espacio celular en tiempo real; y un panel derecho para la configuración de la simulación.

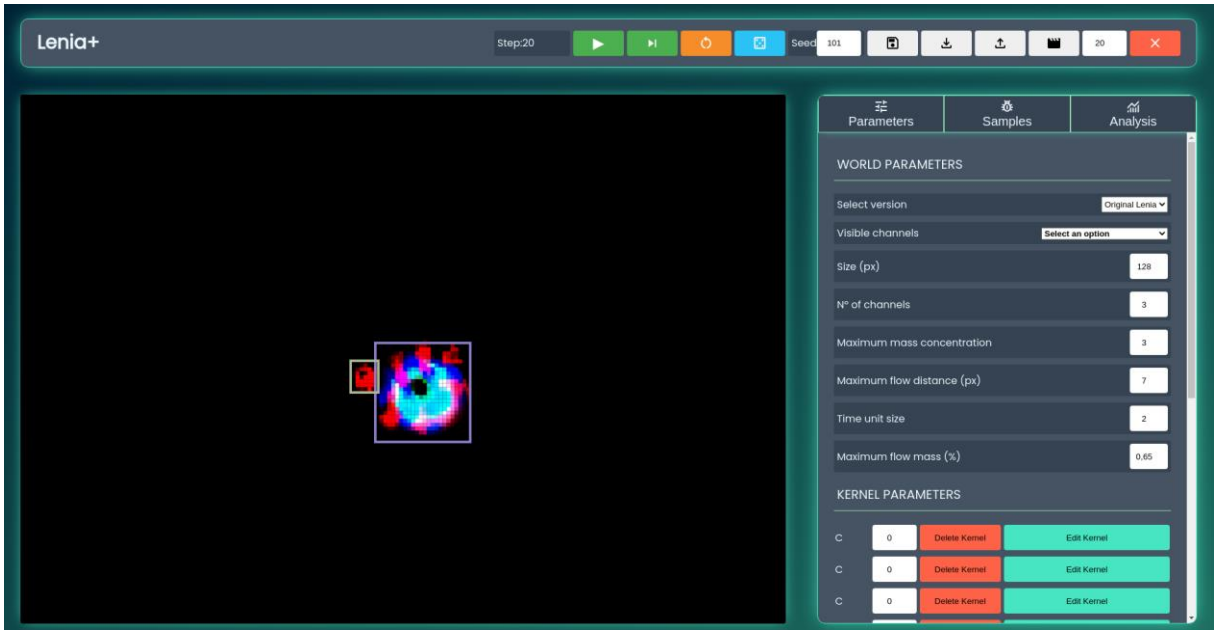


Ilustración 10. Interfaz gráfica, mostrando panel superior, visualizador y panel derecho

El primer panel contiene botones con las siguientes operaciones: reproducir; reproducir un paso; reiniciar simulación con la configuración establecida; generar una nueva configuración de kernels a partir de la semilla indicada; grabar vídeo de duración indicada; cerrar programa.

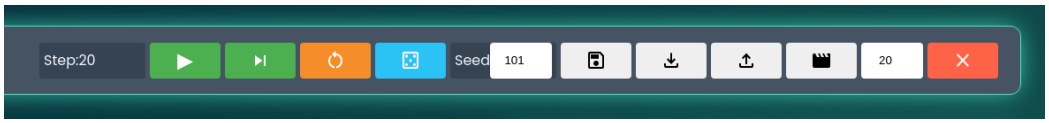


Ilustración 11. Interfaz gráfica de panel superior.

Seguidamente, está el panel del espacio celular. Esta se representa como una visualización RGB, es decir, se muestran 3 canales. Como se indicará más adelante, se podrá seleccionar los canales que se quieran mostrar, de forma que, si hay más de 3, solo se podrá visualizar un máximo de 3. Además, en esta visualización se muestra el *bounding box* correspondiente para cada individuo, que se obtiene de la forma indicada en el CAPÍTULO 5.3. CÁLCULOS. Para diferenciar los individuos, se asocia un color aleatorio a cada nuevo individuo que se detecte, y con el que quedará asociado hasta que muera.

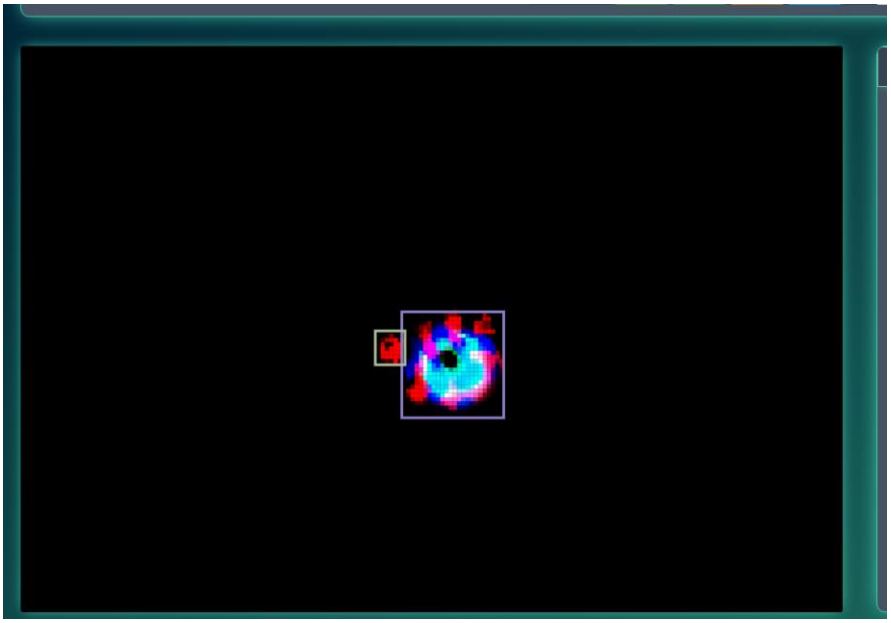


Ilustración 12. Interfaz gráfica de visualizador de simulador.

El panel de configuración contiene 3 posibles secciones a mostrar. La primera es donde se encuentran los parámetros que se pueden configurar de la simulación. Esta incluye una sección que lista los kernels configurados para la simulación.

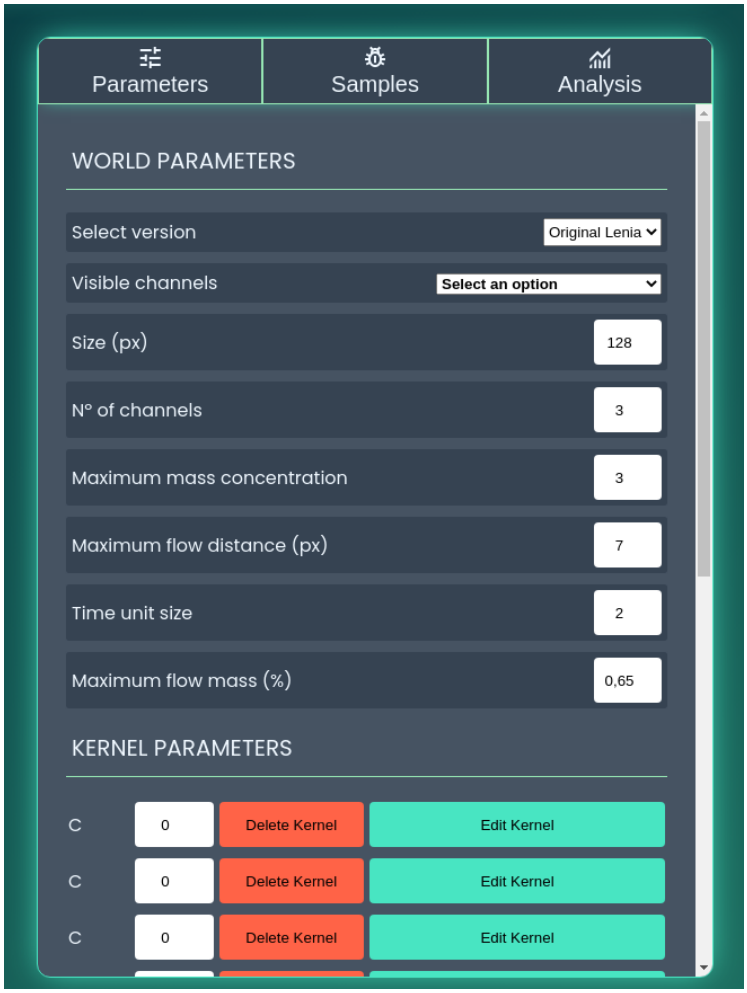


Ilustración 13. Interfaz gráfica de panel derecho, en la pestaña de configuración del mundo.

Desde la sección de kernels, se pueden añadir o eliminar tantos kernels como se necesite. Pulsando en el botón de editar, se acceden a los parámetros que caracterizan ese kernel, de forma que se pueden modificar, añadir o eliminar.

Ilustración 14. Interfaz gráfica de panel derecho, en la pestaña de configuración del mundo, en la sección de kernels.

La segunda sección es una galería de ejemplares almacenados. Desde esta, el usuario puede seleccionar uno para que sea cargado al simulador, con sus parámetros y estado inicial del mundo.

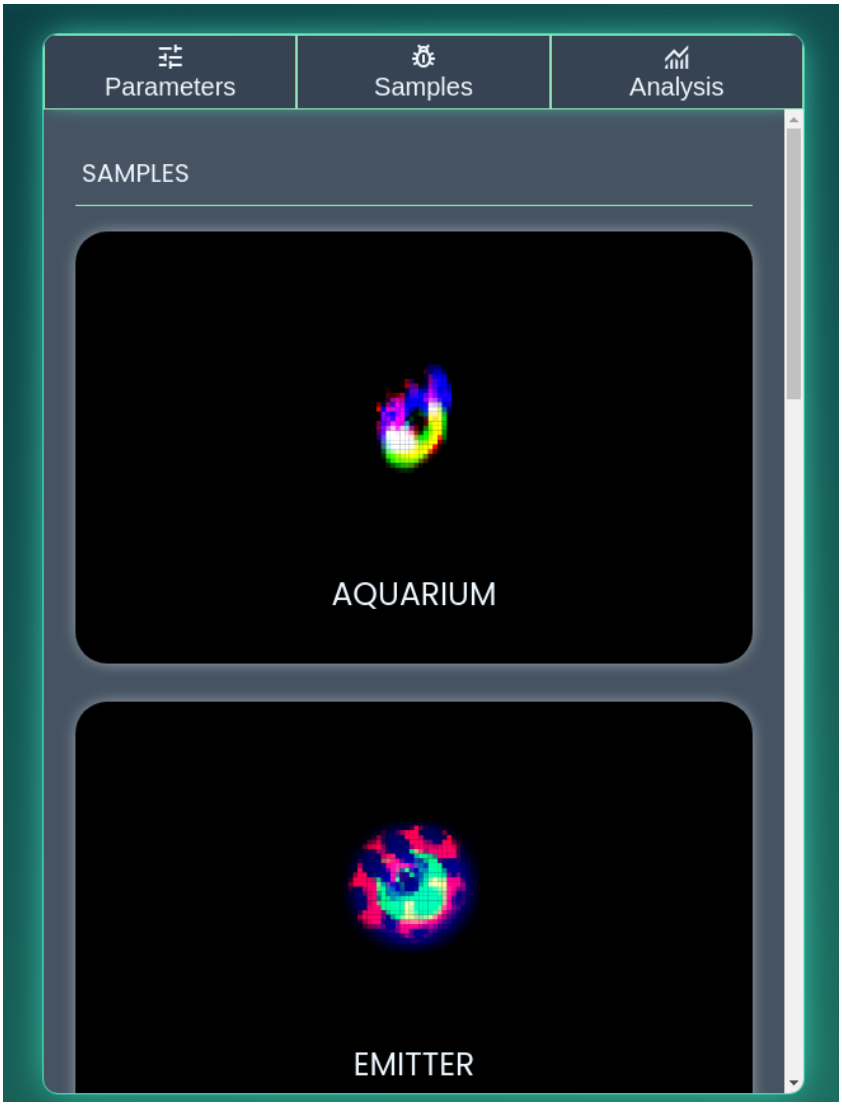


Ilustración 15. Interfaz gráfica de panel derecho, en la pestaña de galería de ejemplares.

Por último, está la sección para el análisis estadístico. En esta, se muestran varios datos de la simulación.

El primero de los datos es un cuadro que muestra la traza dibujada por el movimiento del centroide de cada individuo. Se puede diferenciar el individuo mediante el color asociado a su bounding box. Seguidamente, se muestran tres gráficas correspondientes a las 3 métricas mencionadas en el CAPÍTULO 3. EVALUACIÓN Y MEDIDAS, es decir, la supervivencia, reproducción y morfología. Se consideran estas gráficas como *globales*, ya que incluyen todos los individuos.

Ilustración 16. Interfaz gráfica de panel derecho, en la pestaña de estadísticas.

La primera gráfica contiene las estadísticas de la métrica de supervivencia, y que representa su medida de supervivencia para el instante actual. Lo mismo ocurre con las siguientes dos gráficas, que muestran las métricas de reproducción y morfología. Todas las gráficas muestran la medida de cada individuo representado en el eje horizontal por su correspondiente barra, y que se encuentran coloreada por el mismo color que tienen asociado en sus bounding box. El valor de la medida representadas por las barras puede ir desde -1 a 1.

Al final de la sección de Análisis se encuentra una lista de botones para cada individuo, asociados con los colores de su correspondiente bounding box.

Ilustración 17. Interfaz gráfica de panel derecho, en la pestaña de estadísticas, en la sección de gráficas de las medidas.

Seleccionando un individuo, se abre una ventana nueva, mostrando las gráficas de las mismas 3 medidas mencionadas, pero esta vez solo para el individuo seleccionado. Estas gráficas acumulan el historial de los últimos 10 instantes de tiempo. De esta forma, se tiene una visualización más sencilla de la evolución de ese individuo a lo largo del tiempo.

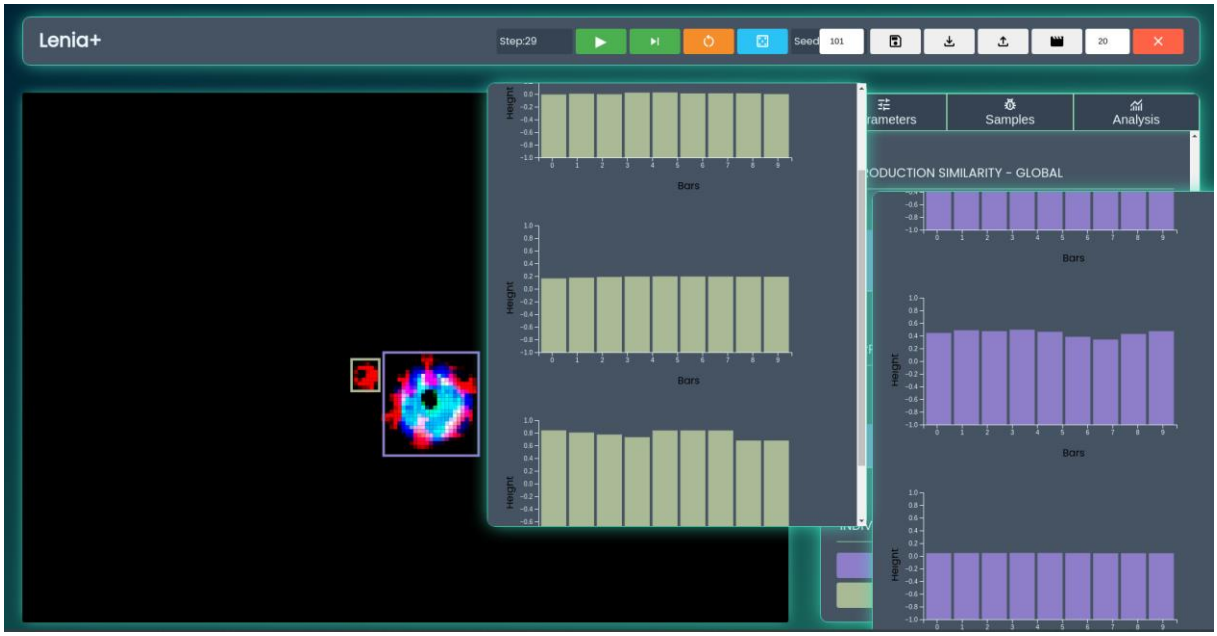


Ilustración 18. Interfaz gráfica de nuevas ventanas para mostrar mediciones individuales.

Se pueden abrir un máximo de dos individuos con estadísticas individuales, de manera que se pueda realizar una comparación entre estas.

5.5. HERRAMIENTAS

En este apartado, se nombrarán las distintas herramientas empleadas para llevar a cabo el desarrollo, así como las ventajas que proporcionan cada uno.

TRELLO

En primer lugar, está la herramienta Trello (Trello.com, 2011). Trello es una herramienta de gestión de proyectos y colaboración en equipo basada en tarjetas organizadas en listas. Permite arrastrar y soltar tarjetas para indicar el flujo de trabajo y el estado de las tareas. Se pueden adjuntar archivos, establecer fechas de vencimiento y asignar tareas a miembros del equipo. Trello también ofrece integraciones con otras aplicaciones y servicios, lo que aumenta su flexibilidad y productividad. Es una opción popular debido a su simplicidad y facilidad de uso. Aunque tiene algunas limitaciones en la versión gratuita, sigue siendo una herramienta eficaz para simplificar la organización y coordinación de actividades en proyectos individuales.

FIGMA

En segundo lugar, se empleó los servicios de Figma (Figma.com, 2016). Es una herramienta en línea para diseñar interfaces. La interfaz de Figma es similar a otras aplicaciones, y se puede utilizar desde el navegador o descargando una aplicación. Ofrece recursos predeterminados y una amplia variedad de plugin y assets disponibles para los usuarios. Además, cuenta con características avanzadas de seguridad para evitar cambios accidentales en los componentes.

ANACONDA

En tercer lugar, se usó Anaconda (Anaconda.com, 2012). Se trata de una distribución de Python y R diseñada para la ciencia de datos, aprendizaje automático, y análisis de datos en general. Anaconda simplifica la gestión de datos y el despliegue de los mismos, proporcionando un entorno fácil de gestionar.

La principal ventaja de Anaconda es su facilidad de uso y la inclusión del administrador de paquetes Conda, que instala, ejecuta y actualiza paquetes y sus dependencias. Conda es útil

para el manejo de paquetes en Python y otros lenguajes de programación, y ofrece una gestión de dependencias más eficiente en comparación con pip. Además, Anaconda incluye una gran cantidad de bibliotecas y paquetes preinstalados, como NumPy, SciPy, Pandas, Scikit-learn, nltk y Jupyter.

VISUAL STUDIO CODE

Por último, el código se desarrolló en Visual Studio Code (Visual Studio, 2015). Es un editor de código fuente ligero y altamente personalizable que ofrece una amplia variedad de características y extensiones para mejorar la experiencia de programación en Python. Entre las ventajas de Visual Studio Code para programar en Python, destacan su interfaz fácil de usar, la capacidad de integrarse con herramientas de control de versiones como Git, la extensibilidad mediante la instalación de extensiones como la extensión de Python, y la capacidad de ejecutar código Python directamente desde el editor.

GIT

La importancia de Git (Git, 2005) para un gran proyecto individual como este radica en varias razones clave. Git proporciona un sistema de control de versiones distribuido, lo que permite realizar un seguimiento de los cambios realizados en el proyecto a lo largo del tiempo. Permite revertir cambios, fusionar ramas y ramificar el proyecto de manera eficiente. También brinda seguridad y respaldo al almacenar todo el historial de cambios en el repositorio, lo que proporciona una capa adicional de protección para el proyecto.

5.6. LIBRERÍAS

En esta sección del capítulo, se explicarán las librerías empleadas en este proyecto, tanto para Python, como para JavaScript. Además, se indicarán algunas de las decisiones por las que se eligieron.

NUMPY

NumPy (Numpy, 1995) es una librería de Python muy importante en el ámbito de la computación científica (Cardellino, 2021). Sus principales ventajas para este proyecto (computación de simulaciones en tiempo real) se pueden resumir en los siguientes puntos:

1. Generación y manejo de datos extremadamente rápido: NumPy proporciona su propia estructura de datos llamada arreglo, que es similar a la lista normal de Python, pero puede almacenar y operar con datos de manera mucho más eficiente.

2. Operaciones matemáticas rápidas: NumPy facilita realizar operaciones aritméticas con arreglos, lo que permite un manejo eficiente de cálculos numéricos.

3. Operaciones complejas en arreglos: NumPy ofrece métodos y operaciones incorporadas para realizar operaciones matemáticas más complejas en arreglos, lo que permite una mayor flexibilidad y eficiencia en el manejo de datos.

SCIPY

SciPy (SciPy , 2001) es una librería de código abierto que se basa en NumPy y proporciona un amplio conjunto de herramientas y funciones para realizar cálculos científicos y matemáticos. Una de las principales ventajas de SciPy es su integración con NumPy, lo que permite aprovechar las capacidades de álgebra lineal y manipulación de matrices de NumPy para realizar cálculos eficientes y rápidos.

Además, SciPy ofrece una amplia gama de módulos especializados que cubren áreas como la optimización, la interpolación, la integración numérica, la estadística y mucho más. Estos módulos proporcionan funciones altamente optimizadas y eficientes que permiten realizar cálculos complejos de manera sencilla y rápida.

JAX

La librería Jax (JAX, 2022) de Python es altamente recomendada para realizar computaciones pesadas con NumPy debido a varias razones. Está diseñada para aprovechar al máximo las unidades de procesamiento gráfico (GPU) y las unidades de procesamiento tensorial (TPU), lo que permite acelerar las operaciones de álgebra lineal y realizar cálculos matemáticos de manera eficiente y rápida.

Otra ventaja de Jax es que adopta un enfoque funcional para la programación, lo que significa que se basa en funciones puras y evita el uso de estados mutables. Esto tiene ventajas como la eliminación de efectos secundarios y la facilidad de paralelización y optimización automática. Jax también proporciona una interfaz compatible con NumPy, lo que facilita la transición y el uso de las funciones existentes.

EEL

La librería Eel (Eel, 2017) de Python y JS es muy popular debido a varias razones. Primero, Eel permite crear aplicaciones de escritorio utilizando Python en el backend y JavaScript en el frontend. Esto facilita el desarrollo de aplicaciones web interactivas y de alto rendimiento.

Además, Eel proporciona una interfaz sencilla y fácil de usar para la comunicación entre Python y JavaScript. Esto permite la transferencia de datos y la ejecución de funciones de manera eficiente y sin problemas entre los dos lenguajes.

D3JS

La librería D3.js (D3js, 2011) de JavaScript es ampliamente reconocida por su capacidad para representar eficientemente grandes cantidades de datos y su sencillez de uso. Una de las principales ventajas de D3.js es su capacidad para manejar grandes conjuntos de datos y

representarlos de manera eficiente. D3.js utiliza enfoques como la manipulación del DOM y el uso de escalas y ejes para optimizar la representación de datos y garantizar un rendimiento óptimo.

Además, D3.js ofrece una amplia gama de gráficos y visualizaciones predefinidas, como gráficos de barras, gráficos de líneas, diagramas de dispersión y mucho más. Estas visualizaciones se pueden personalizar fácilmente para adaptarse a las necesidades específicas de cada proyecto.

CAPÍTULO 6. EXPERIMENTOS Y RESULTADOS

En este capítulo, se describen experimentos evaluados en la herramienta software desarrollada. Se especifican los pasos seguidos para poder ser reproducidos, y se tratará de dar una interpretarán a los resultados.

6.1. EXPERIMENTACIÓN 1: ORB

En primer lugar, se experimenta con el ejemplar ORB. Como se comentó en el CAPÍTULO 2.5. EJEMPLARES, es uno de los primeros descubiertos por el autor de Lenia, que se simula sobre un único canal RGB. En la simulación se puede apreciar su capacidad para desplazarse. En biología, se emplea el término *motilidad*, para expresar la habilidad de moverse espontánea e independientemente.

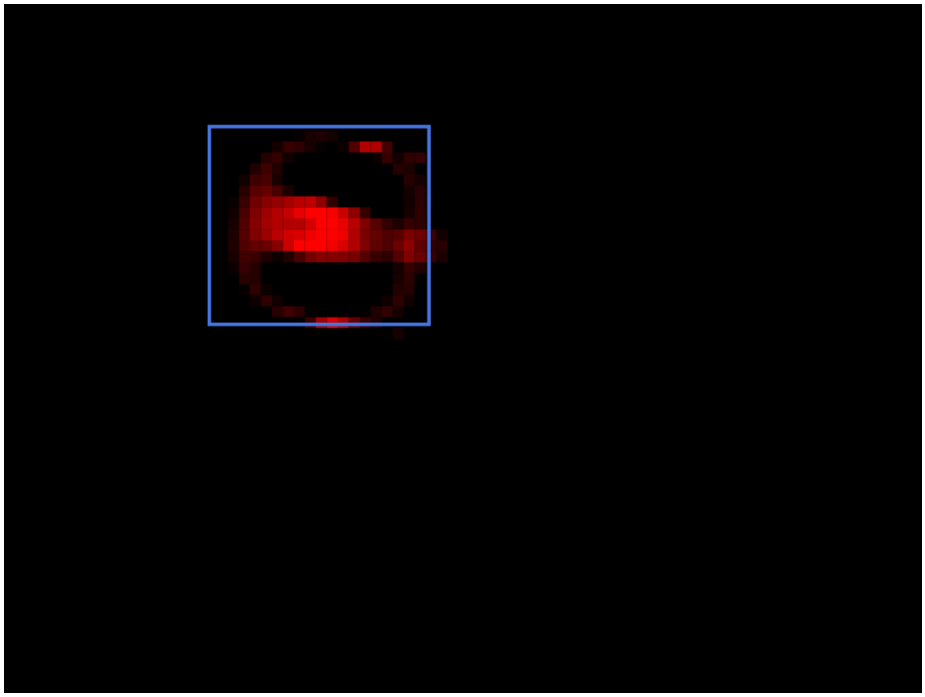


Ilustración 19. Visualización de simulación de ejemplar ORB.

Al tener la capacidad de moverse, sigue una trayectoria. Por ello, en el panel de análisis se proporciona un visor de un tamaño más reducido, en el que se puede observar la traza que va dejando la trayectoria seguida. Se puede observar claramente cómo sigue una trayectoria rectilínea.

Ilustración 20. Visualización de traza de movimiento de ORB.

Se puede destacar su capacidad de homeostasis. En biología se define esta propiedad como la capacidad de un organismo de mantener una condición interna estable compensando los cambios en su entorno mediante el intercambio regulado de materia y energía con el exterior. Este aspecto se puede comprobar viendo la gráfica de supervivencia, en el que se ve que no se está disolviendo o *muriendo*, pero tampoco creciendo.

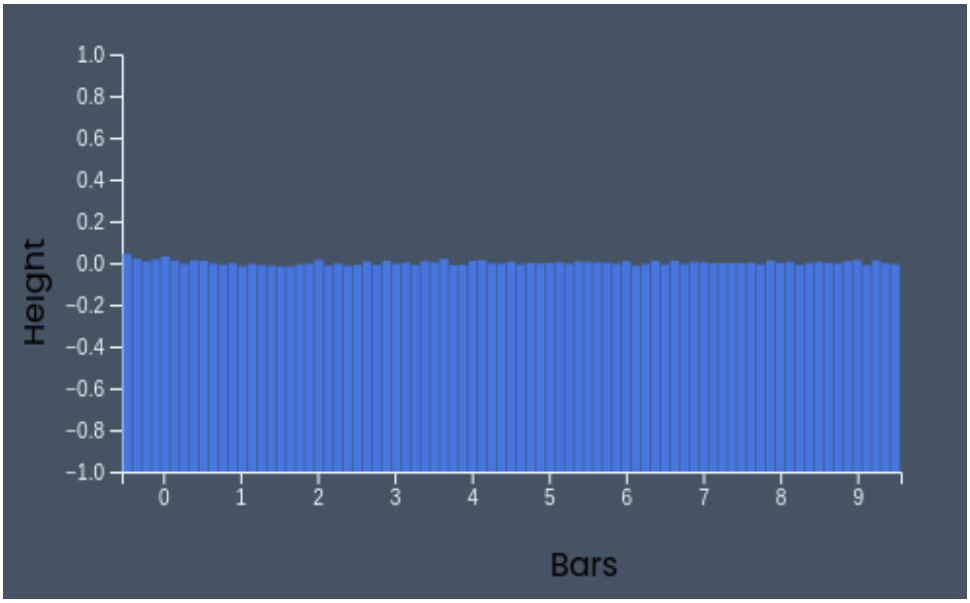


Ilustración 21. Gráfica de medición de supervivencia de ORB.

Por otro lado, durante su desplazamiento, el individuo también cambia su forma. Esto se puede apreciar mejor en la gráfica de morfología, ya que, como se menciona en el CAPÍTULO 3.4 MORFOLOGÍA, se computa la similitud entre el estado actual del individuo y su estado inicial. Se puede ver cómo la morfología del individuo es similar a su estado inicial cada cierto tiempo, con una determinada periodicidad.

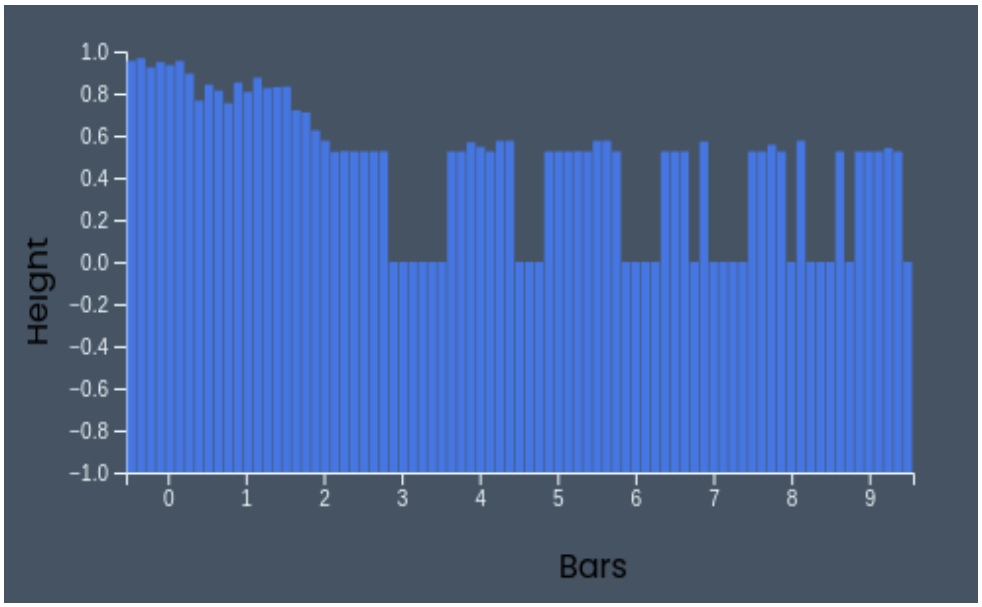


Ilustración 22. Gráfica de medición de morfología de ORB.

6.2. EXPERIMENTACIÓN 2: AQUARIUM

Ahora, se estudiará el ejemplar Aquarium, también descubierto por el autor y comentado en el CAPÍTULO 2.5. EJEMPLARES. En este caso, a diferencia del ejemplar anterior, este es simulado sobre los tres canales RGB. Por otro lado, es similar en que también tiene la propiedad de motilidad, teniendo la capacidad de desplazarse.

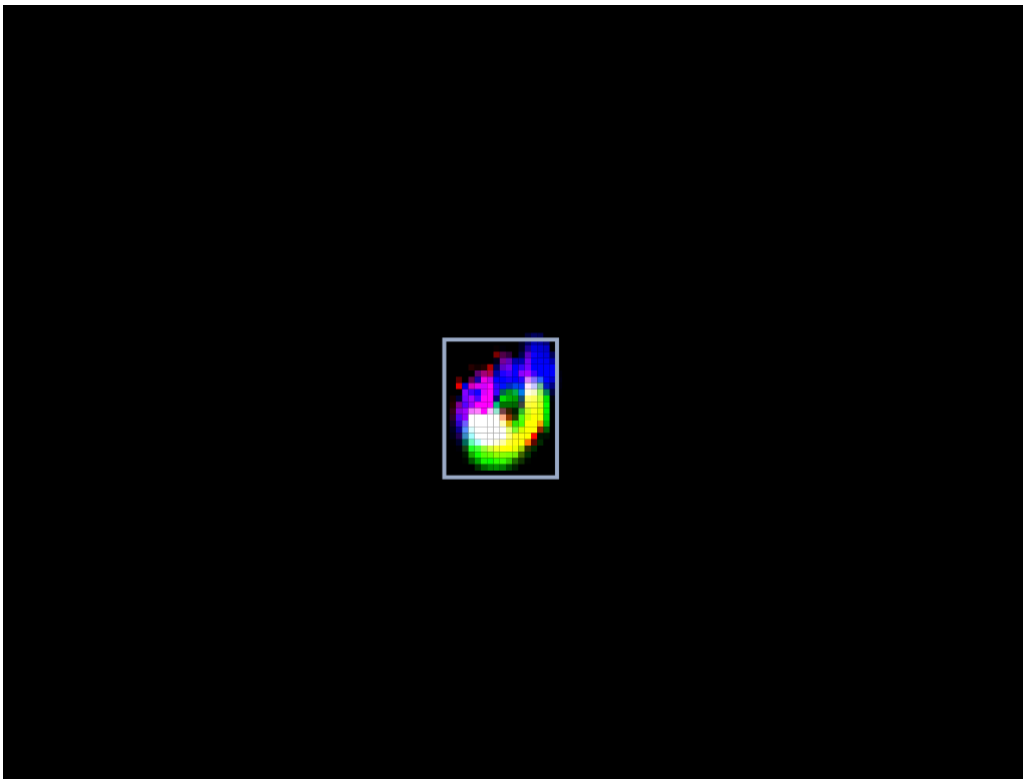


Ilustración 23. Visualización de simulación de ejemplar Aquarium.

A simple vista, parecería que la trayectoria que sigue no sigue un patrón muy distinguido, como la trayectoria rectilínea de la anterior especie. No obstante, en el visor se puede observar que la traza que deja la trayectoria es curvilínea, y retrocediendo en línea recta.

Ilustración 24. Visualización de traza de movimiento de Aquarium.

Tras recorrer media circunferencia, empieza a cambiar su morfología, alargándose para proceder a dividirse, originándose un nuevo individuo. Este comportamiento se puede interpretar como *reproducción*, concretamente una reproducción asexual, pues no necesita de otro individuo para llevarlo a cabo.

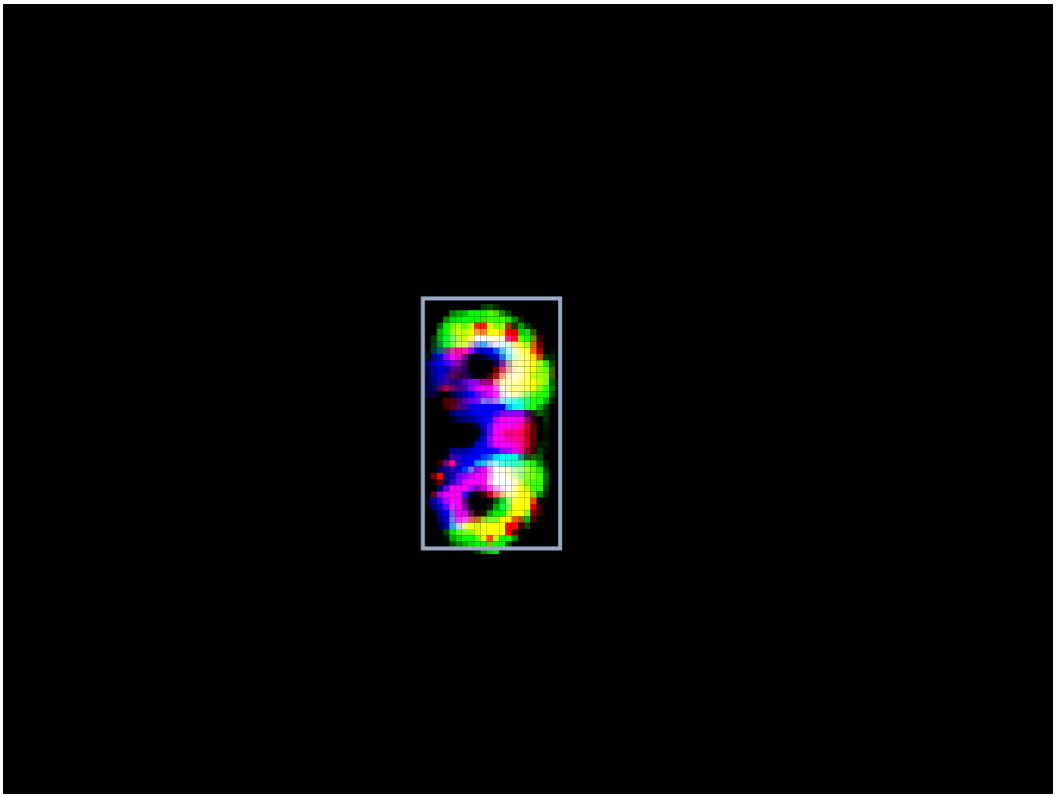


Ilustración 25. Visualización de simulación de ejemplar ORB en proceso de reproducirse.

Se puede observar este fenómeno cuando hay más individuos reproducidos.

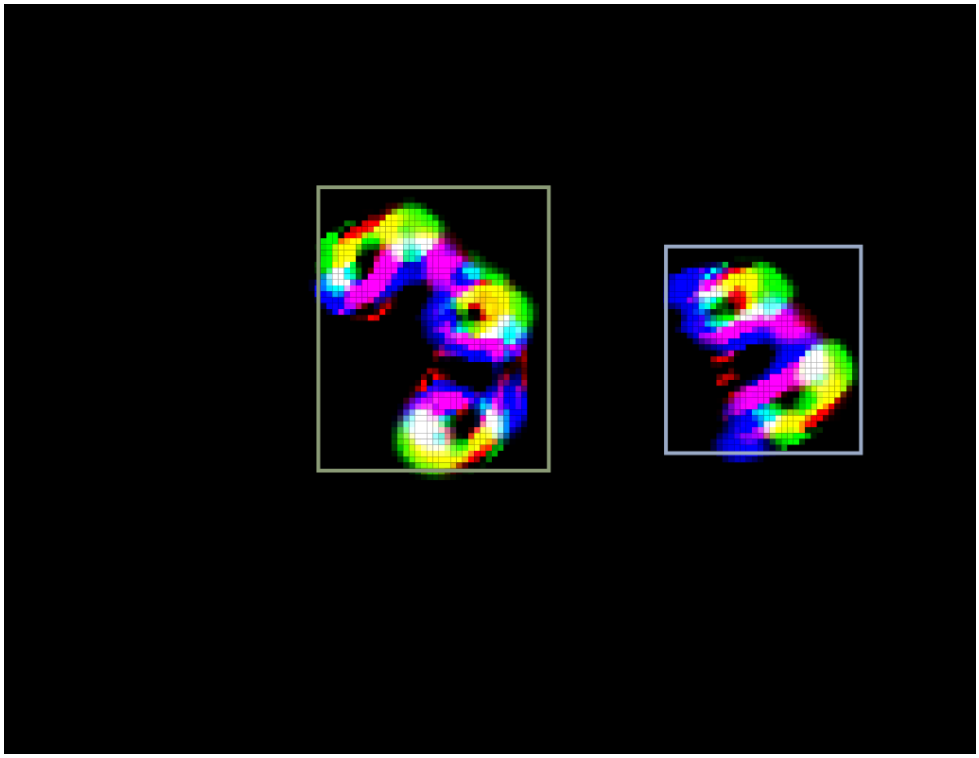


Ilustración 26. Visualización de simulación de dos ejemplares ORB.

Y su trayectoria.

Ilustración 27. Visualización de traza de movimiento de varios ejemplares Aquarium.

Observando la gráfica de morfología, se puede ver que es diferente a la anterior en que su morfología no periódicamente mientras se desplaza. Sin embargo, tampoco es constante, ya que se puede apreciar que durante el proceso de reproducción su morfología va cambiando y alejándose de su forma inicial.

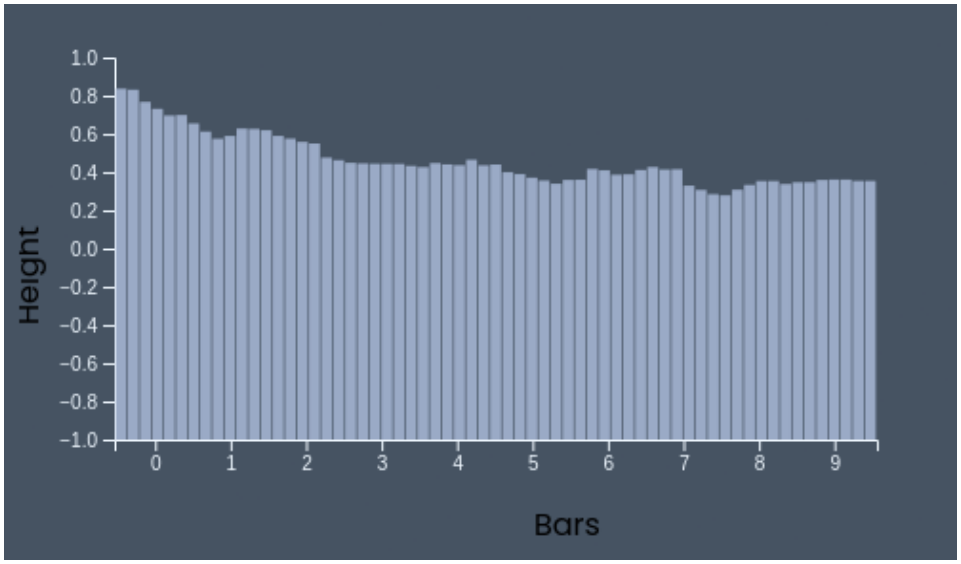


Ilustración 28. Gráfica de medición de morfología de Aquarium.

Como esta especie tiene la capacidad de reproducirse, a medida que pase el tiempo habrá más individuos.

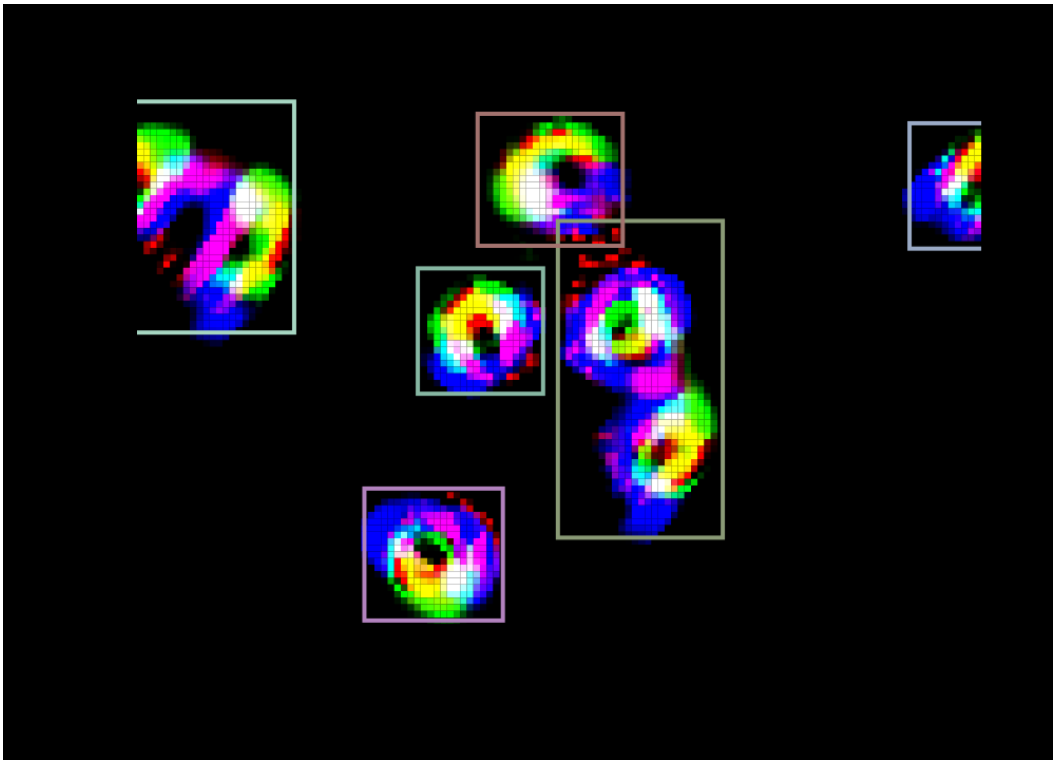


Ilustración 29. Visualización de simulación de varios ejemplares ORB tras reproducción.

Por eso, la gráfica de reproducción resulta más útil en este caso, especialmente la global, que incluye el de todos los individuos del espacio. Se puede ver que hay bastantes diferencias entre estas, lo que se podría interpretar como que la reproducción de un individuo nuevo provoca cambios en su morfología respecto al resto de individuos, para las siguientes generaciones. En la gráfica, el color de cada barra se corresponde al color del bounding box asociado a cada individuo en la figura anterior.

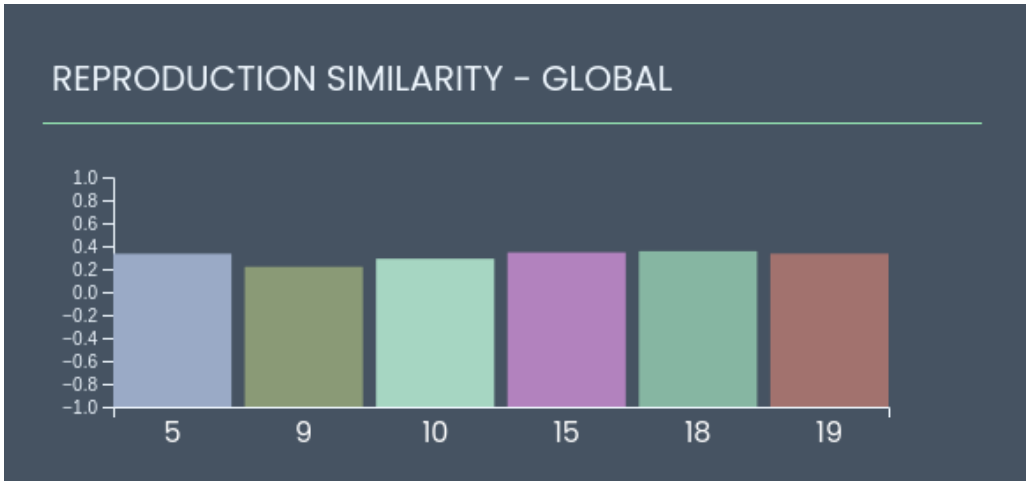


Ilustración 30. Gráfica de medición de reproducción global de Aquarium.

CAPÍTULO 7. CONCLUSIONES Y TRABAJO FUTURO

7.1. CONCLUSIONES

Con este proyecto, se ha logrado alcanzar los objetivos planteados inicialmente. A continuación, se detallan los logros obtenidos en relación a cada uno de ellos.

7.1.1. DEL TRABAJO

En primer lugar, se ha logrado implementar un marco software sólido y flexible que permite llevar a cabo experimentos en entornos con autómatas celulares de la familia Lenia. Este marco ofrece las herramientas necesarias para definir, simular y visualizar el comportamiento de los autómatas celulares en tiempo real. Además, se ha asegurado la escalabilidad del marco, permitiendo la incorporación de nuevos elementos y adaptándose a posibles extensiones futuras.

En segundo lugar, se ha llevado a cabo un proceso de experimentación y evaluación de resultados. Se han diseñado y ejecutado diversos experimentos utilizando diferentes configuraciones de autómatas celulares de la familia Lenia, y se ha analizado y comparado el comportamiento obtenido.

Por último, como parte de las actividades propias de este tipo de proyectos, se ha redactado una memoria final que documenta todos los aspectos relevantes del desarrollo del simulador. Esta memoria incluye la descripción de los objetivos, la metodología utilizada, los resultados obtenidos, así como las conclusiones y recomendaciones derivadas del proyecto.

7.1.2. DE CARÁCTER PERSONAL

Durante el desarrollo del proyecto, se ha adquirido un profundo conocimiento sobre los métodos y técnicas asociados a la Vida Artificial. Se han estudiado y aplicado conceptos de biología y vida en un entorno software, permitiendo comprender las características fundamentales relacionadas con la vida, tal como se exploran en las ciencias biológicas. Esto ha involucrado el análisis de la evolución, la autoorganización, la emergencia y la adaptación en sistemas vivos, y su aplicación en entornos artificiales.

En segundo lugar, se ha desarrollado un proyecto software completo y adquirido conocimientos y experiencia. El proyecto ha permitido consolidar los conocimientos adquiridos durante los estudios de grado y adquirir una valiosa experiencia en todas las fases de desarrollo de un proyecto software, desde su concepción inicial hasta su implementación y finalización. Durante el proceso, se ha aplicado de manera efectiva las mejores prácticas de desarrollo de software, como la planificación, el diseño modular, la codificación eficiente y la realización de pruebas exhaustivas.

Se ha logrado profundizar en los principios biológicos que definen los sistemas vivos. Esto ha implicado el estudio de procesos biológicos como la supervivencia, la reproducción, y la morfología, y su representación en el simulador de tiempo real. A través de esta comprensión, se ha logrado modelar de manera efectiva los fenómenos biológicos en el entorno software.

7.2. PRINCIPALES APORTACIONES

Con este proyecto software de fin de grado, se han realizado varias aportaciones. Se ha desarrollado una herramienta software que aborda de manera efectiva el problema comentado en el CAPÍTULO 2.6 PROBLEMA EXISTENTE, relacionado con la falta de herramientas exhaustivas con interfaces gráficas sencillas de investigación para Lenia.

Con esta herramienta, los usuarios tienen la capacidad de configurar y ajustar los parámetros de Lenia, lo que les permite explorar y observar patrones emergentes en tiempo real. Además, se ha incorporado una función de guardar ejemplares, lo que permite a los investigadores seleccionar y conservar configuraciones particulares de Lenia para su análisis posterior.

Una característica clave de esta herramienta es la visualización en tiempo real y los controles interactivos. Esto permite a los investigadores recibir retroalimentación visual instantánea sobre las simulaciones y ajustar los parámetros de manera interactiva para observar cambios en los patrones generados. Esta funcionalidad facilita la exploración del comportamiento de Lenia y proporciona una herramienta valiosa para la experimentación y la formulación de hipótesis.

Además de la capacidad de simulación en tiempo real y la configuración de parámetros, esta herramienta también ofrece estadísticas detalladas sobre varias mediciones de la simulación de Lenia. Esto permite a los investigadores realizar análisis cuantitativos y extraer información significativa de las simulaciones. Estas estadísticas proporcionan una base para investigaciones científicas y contribuyen a la comprensión de los fenómenos complejos generados por Lenia.

7.3. TRABAJO FUTURO

Como futuras ideas a implementar en una herramienta como la que se ha trabajado en este proyecto, se ha enfocado en aquellas mejoras que tienen más atracción de cara al futuro.

Una posible mejora sería la integración de bibliotecas y módulos adicionales que permitan a los investigadores ampliar las capacidades de simulación y análisis de Lenia. Esto podría incluir la incorporación de algoritmos de aprendizaje automático para el análisis y la predicción de patrones emergentes, o la integración de bibliotecas de visualización avanzadas para representar los resultados de las simulaciones de manera más inmersiva y comprensible.

Otra idea interesante sería explorar la compatibilidad con plataformas de computación distribuida. Esto facilitaría la realización de simulaciones a gran escala y de alto rendimiento al distribuir la carga de trabajo en múltiples nodos o máquinas. Con esta mejora, se aceleraría la velocidad de las simulaciones y facilita la exploración de sistemas Lenia aún más complejos y de mayor escala.

Por otra parte, se destaca la posibilidad del trabajo colectivo. Para fomentar la colaboración entre investigadores y facilitar la compartición de conocimientos y resultados, se podría añadir funcionalidades de colaboración y compartición. Esto permitiría compartir simulaciones en línea, colaborar en tiempo real en proyectos de investigación y tener acceso a una biblioteca centralizada de configuraciones y ejemplares de Lenia compartidos por la comunidad científica.

Sería interesante también incorporar herramientas automáticas de optimización de parámetros, de la misma forma que comenta el autor de Lenia. Estas herramientas podrían utilizar algoritmos evolutivos o técnicas de búsqueda para explorar de manera más eficiente el espacio de parámetros de Lenia y descubrir configuraciones óptimas. Esto agilizaría el proceso de ajuste de parámetros y maximizaría la eficacia de las simulaciones.

Por último, la integración con bases de datos y herramientas de análisis podrían proporcionar una visión más profunda y exhaustiva de los sistemas modelados por Lenia. Esto permitiría realizar análisis comparativos a gran escala, correlacionar los resultados de las simulaciones con datos externos y extraer conocimientos más sólidos y significativos.

CAPÍTULO 8. BIBLIOGRAFÍA

8.1. ARTÍCULOS

[BEDA-03] Bedau, M. A., 2003. *Artificial life: organization, adaptation and complexity from the bottom up*. [Online]
Available at: https://people.reed.edu/~mab/publications/papers/BedauTICS03.pdf
[Accessed 25 June 2023].

[CARD-21] Cardellino, F., 2021. *La guía definitiva del paquete NumPy para computación científica en Python*. [Online]
Available at: https://www.freecodecamp.org/espanol/news/la-guia-definitiva-del-paquete-numpy-para-computacion-cientifica-en-python/
[Accessed 29 June 2023].

[CHAN-18] Chan, B. W.-C., 2018. *Lenia - Biology of Artificial Life*. [Online]
Available at: https://arxiv.org/abs/1812.05433
[Accessed 25 June 2023].

[GARD-70] Gardner, M., 1970. *The fantastic combinations of John Conway's new solitaire game 'life'*. [Online]
Available at: https://web.stanford.edu/class/sts145/Library/life.pdf
[Accessed 9 7 2023].

[GRAN-16] Grand, S., 2016. *Artificial life | Computer Simulation & Robotics | Britannica*. [Online]
Available at: https://www.britannica.com/technology/artificial-life
[Accessed 17 June 2023].

[LANG-95] Langton, C. G. ed., 1997. *Artificial Life: An Overview*. s.l.:BRADFORD BOOK.
Numpy, 1995. *Numpy Website*. [Online]
Available at: https://numpy.org/
[Accessed 5 7 2023].

[PLAN-22] Plantec, E. et al., 2022. *Flow-Lenia: Towards open-ended evolution in cellular automata through mass conservation and parameter localization*. [Online]
Available at: https://arxiv.org/abs/2212.07906
[Accessed 25 June 2023].

8.2. PÁGINAS WEB

Anaconda.com, 2012. *Anaconda Website*. [En línea]
Available at: https://www.anaconda.com/
[Último acceso: 2 4 2023].

D3js, 2011. *D3js Website*. [En línea]
Available at: https://d3js.org/
[Último acceso: 5 7 2023].

Eel, 2017. *Eel Repository*. [En línea]
Available at: https://github.com/python-eel/Eel
[Último acceso: 5 7 2023].

Figma.com, 2016. *Figma*. [En línea]
Available at: https://www.figma.com/
[Último acceso: 1 4 2023].

Git, 2005. *Git Website*. [En línea]
Available at: https://git-scm.com/
[Último acceso: 5 7 2023].

JAX, 2022. *JAX Website*. [En línea]
Available at: https://jax.readthedocs.io/en/latest/
[Último acceso: 5 7 2023].

Numpy, 1995. *Numpy Website*. [En línea]
Available at: https://numpy.org/
[Último acceso: 5 7 2023].

ProgrammerClick.com, s.f. *Cálculo de correlación cruzada normalizada (NCC) del mapa de disparidad*. [En línea]
Available at: https://programmerclick.com/article/78991651504/
[Último acceso: 1 July 2023].

ProjectManger.com, s.f. *Kanban Methodology: The Ultimate Guide (Examples & Software Included)*. [En línea]
Available at: https://www.projectmanager.com/guides/kanban
[Último acceso: 1 July 2023].

SciPy , 2001. *SciPy Website*. [En línea]
Available at: https://scipy.org/
[Último acceso: 5 7 2023].

Trello.com, 2011. *Trello Website*. [En línea]
Available at: https://trello.com/
[Último acceso: 1 4 2023].

Visual Studio, 2015. *Visual Studio Code Website*. [En línea]
Available at: https://code.visualstudio.com/
[Último acceso: 10 7 2023].

ANEXOS

Repositorio Github del proyecto: https://github.com/G10on/LeniaPlus-Framework-Final-Project.git