eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Computación

Echoes of GaIA

Framework de modelado evolutivo basado en simulación de biomas artificiales enfocado en sistemas multiagente e Inteligencia Artificial
Anexo - Fundamentos, arquitectura y sistemas principales

Aingeru García Blas

Dirección
Iñaki Inza Cano
Departamento de Ciencias de la Computación e Inteligencia Artificial

Índice de contenidos

CAPÍTULO 1

Echoes of Gaia: fundamentos y diseño

1.1. ¿Qué es Echoes of GaIA?

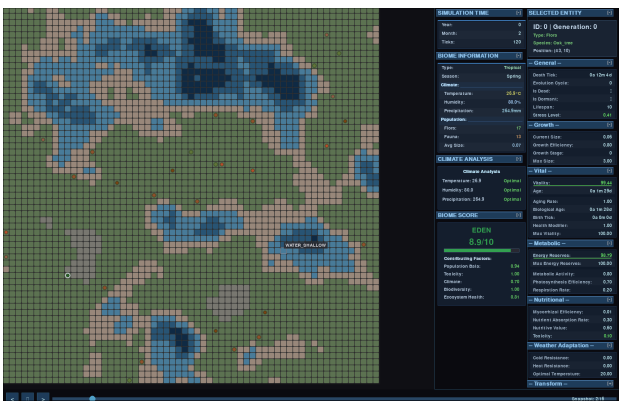
Echoes of GaIA es un framework híbrido que hace de plataforma o motor de simulación de biomas y ecosistemas. La idea principal es poder lanzar simulaciones donde el bioma *cobre vida* para que la fauna, flora y entorno interactúen. Intento simular una pequeña parte de la complejidad de la retroalimentación que existe en ecosistemas de diferentes biomas, pero en un entorno digital; cada especie tiene su propio ciclo evolutivo para adaptarse al entorno, y todo esto se ve afectado por el clima, que a su vez también cambia según la salud del propio bioma y su ecosistema. Hay muchas sinergias e interacciones que determinan el comportamiento general.

Como ya he mencionado, el proyecto posee forma de framework híbrido, de manera que otros usuarios/desarrolladores puedan utilizarlo y extenderlo. Es híbrido, esto es, se posee del core y diferentes módulos que pueden ser extendidos para agregar nuevas funcionalidades, en

lugar de un API para uso puramente externo. También dispone de soporte para *montar* diferentes escenarios mediante ficheros de configuración sencillos. No obstante, dada la arquitectura y la implementación de APIs internas, sería muy sencillo de empaquetar y exponer una API de uso externo.

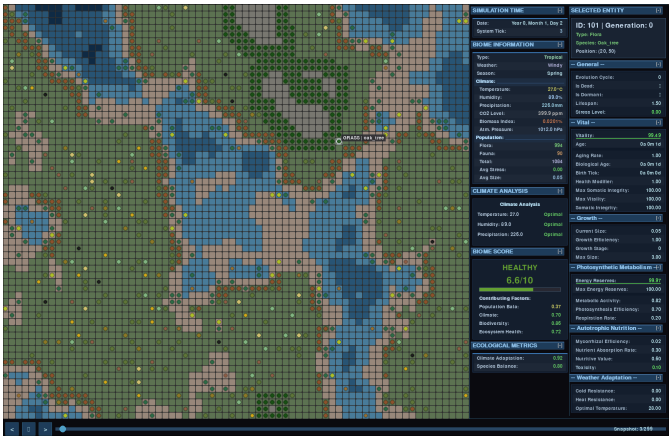
Todo esto que menciono sobre la simulación; son datos en memoria. Estos datos, algunas veces los materializo en 1) ficheros a modo de **snapshots** para utilizar en un visor, 2) **datapoints** para base de datos de **series temporales** (`InfluxDB`) y visualización de métricas a tiempo real (**Grafana)**) 3) e incluso a veces también produzco **datasets**, cosas que me han venido muy bien para analizar comportamientos y entrenar los modelos – pero esto ya lo veremos más adelante.

Con objeto de poder facilitar el desarrollo y además poder visualizar lo que ocurre en el mundo de manera más intuitiva, he desarrollado un visor simple que muestra simulaciones, estadísticas, etc. a lo largo del tiempo de la simulación. He terminado incluyéndola en la plataforma en sí.

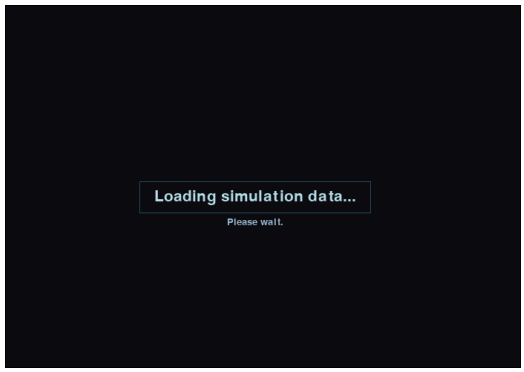


(a) Interfaz

(b) UI mostrando tundra



(c) UI con bioma tropical



(d) Pantalla de carga de datos

Figura 1.1: Interfaz

En esta interfaz veremos una fotografía del estado del bioma en un momento dado, por defecto cada 30 días, pero esto es configurable. Se pueden apreciar la distribución de entidades (flora y fauna), forma de los mapas y sus terrenos. Según avancemos en el tiempo, se podrá apreciar cómo se va actualizando.

Dado el tiempo del que se dispone para desarrollar un TFG, es muy complicado el implementar un proyecto que emule en gran detalle muchos procesos reales. Mi intención ha sido preparar en entorno biológico simplificado, pero que cubra las bases y a modo de framework modular, para que en un futuro otr@s desarrolladoras/es puedan extenderlo con facilidad y a su vez, disponer de unas bases estables basadas en un modelado matemático básico que me he aventurado a hacer.

SOBRE EL NOMBRE

Con el nombre de *Echoes of GaIA* intento evocar la idea de un eco lejano, una réplica virtual que simule los procesos y patrones de la Tierra, a la vez que hago un pequeño guiño al protagonismo que tiene la Inteligencia Artificial (*Ga-IA*). *Gaia* representa nuestro planeta como sistema vivo, y con *echoes* lo que quiero transmitir es que esta simulación refleja de manera simplificada pero más o menos reconocible, sus procesos naturales.

VIDEOJUEGO

Me he dedicado al desarrollo de videojuegos y servidores de juegos durante años y, con el tiempo, he ido pensando y guardando muchas ideas sueltas para posibles nuevos juegos. El factor común en todas ellas, es que siempre están presente el incorporar sistemas inteligentes, siempre me ha atraído mucho por algún motivo, muchos años atrás del *boom* actual. Con lo que en Echoes of GaIA he dado un poco de rienda suelta en ese aspecto e intentado hacer precisamente eso, pero en este caso orientado a simulación ecológica. No obstante, cuando empecé a diseñar el proyecto, tenía en mente la posibilidad de desarrollar una demo pequeña, que hiciera uso de los modelos entrenados y permitiera al jugador interactuar con los sistemas para tomar decisiones en cuanto si intervenir o no en el ecosistema - en función de lo que los sistemas predijeran.

Antes de comenzar *oficialmente* con el TFG el 3 de Febrero (commits 20 noviembre - 3 febrero), estuve preparando a ratos libres las bases para un juego que sostuviera el TFG, hice una *demo-intro* al menú principal y preparé algo a modo de boceto conceptual. No es muy relevante, pero por si interesa, para tener una noción del ambiente, música etc que quería transmitir, desarrollé una pequeña demostración:

Demo breve - Menú principal
Clic para ver el video con sonido

Canción: Ambience forest
Ésta y otras canciones que aparecerán en videos durante la memoria, las creé con **udio.com**. Todas accesibles en este link

No obstante, como intuía, no es factible desarrollar aunque simplemente fuera una demo del videojuego en el espacio de tiempo que dispongo para el TFG, teniendo en cuenta lo grande que es el proyecto de base. Pero, **queda pendiente para hacer un futuro**.

1.1.1. Ecosistemas vivos virtuales

La visión ha sido holística - no quería centrarme en una única cosa - mi idea era diseñar el framework tal que ofrezca soporte para sistemas de adaptación, evolución, interacción y supervivencia dinámica.

Modelar un ecosistema completo en una simulación, es algo prácticamente inviable; así que, me he centrado en implementar los mecanismos básicos: interacciones depredador-presa, influencia del clima en las especies, y ciclos de adaptación genética para que las plantas y animales evolucionen. A modo de ejemplos y primer vistazo rápido:

Ejemplos

- Mediante fotosíntesis, las plantas regulan el CO_2 atmosférico.
- Los herbívoros buscan comida e intentan huir cuando detectan depredadores.
- Estos últimos, planean estrategias de caza dependiendo de múltiples factores.
- Todo lo que ocurra en los biomas, afecta al clima a largo plazo.
- A su vez, el clima afecta al bioma y fluctúa con las estaciones.
- Los organismos vivos (flora y fauna) evolucionan con el tiempo, y se adaptan al entorno.

Gracias a mi paso por la universidad, he adquirido bases para ahora poder leer y entender mejor literatura más avanzada. He investigado un poco sobre biología, ecología, climatología e inteligencia artificial. Con ello, he diseñado multitud de modelos computacionales y utilizado diversas técnicas de IA para dejar que *gobierne la vida* (más o menos) - al lanzar la simulación, dejo que las interacciones emerjan por sí mismas. Hay muchas cosas que me hubiera gustado modelar - no ha sido fácil poner límites; me parece un proyecto apasionante y la cantidad de mecánicas tanto biológicas como de comportamiento que se pueden implementar, tiende a infinito.

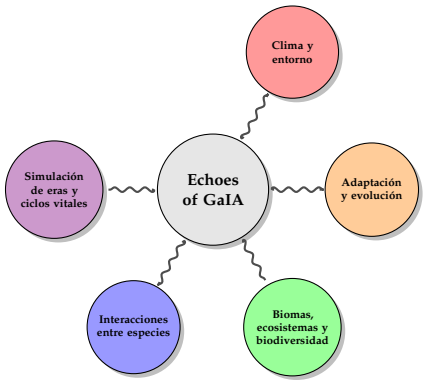


Figura 1.2: Conceptos clave de Echoes of GaIA

1.2. Un laboratorio virtual

Echoes of GaIA ha sido hecho from scratch, lo empecé desde 0 (ir al origen de los tiempos) y como comenté, he tenido la suerte de poder idear, diseñar y desarrollar el proyecto de manera completamente autónoma - para lo bueno, y para lo malo.

La simulación funciona sobre un **espacio bidimensional; un grid 2D** y cada celda puede contener diferentes tipos de terreno - agua, arena, hierba, montaña, etc. - y además albergar entidades biológicas. Estas entidades tienen diversos atributos: energía, estrés, ciclo de vida... (y muchos otros, se irán mostrando a lo largo de la memoria técnica) y están condicionadas por las interacciones con otras especies.

El objetivo principal con el que diseñé todo esto era crear una plataforma extensible - sé que hoy todo software se vende como *'extensible, modular y scalable...'* y en breves comenzará a ser también *fresh and locally sourced*. Pero como se verá, realmente es muy sencillo incorporar nuevas funcionalidades y

crear escenarios mediante simples ficheros de configuración. Así, se puede investigar con algoritmos de Reinforcement Learning, sistemas neurosimbólicos, redes neuronales o técnicas evolutivas (existe un módulo de investigación), o experimentar con diferentes biomas diseñados por el usuario; de esta forma se puede observar sus sinergias e interacciones.

1.2.1. ¿Qué es un Bioma en Echoes of GaIA?

Los biomas representan una zona geográfica con su ecosistema y sus propios parámetros. Como he mencionado antes, he conceptualizado estos como sistemas integrados donde cada elemento contribuye - desde el terreno hasta el clima y las especies que lo habitan.

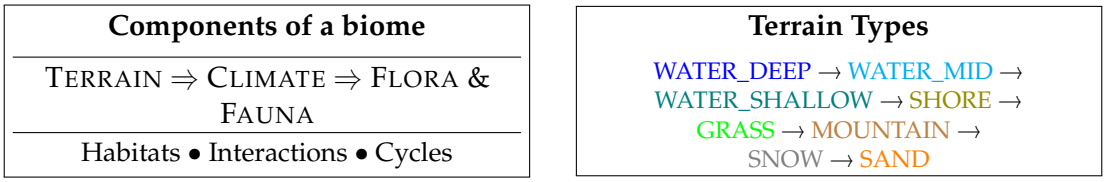


Figura 1.3: Tipos básicos de terrenos y biomas

TROPICAL	SAVANNA	DESERT	TAIGA	TUNDRA
T: 23-31°C H: 70-90 %	T: 25-35°C H: 25-70 %	T: 27-45°C H: 2-10 %	T: -20-12°C H: 40-75 %	T: -23-12°C H: 30-50 %
Oak trees, mushrooms, bromeliads	Acacia trees, savanna grass, baobabs	Cactus, date palms, aloe	Pine trees, arctic moss, spruce	Arctic willows, lichens, cotton grass
Panthers, deer	Lions, zebras, elephants	Desert foxes, camels	Wolves, moose, lynx	Polar bears, musk ox, arctic hares

Los parámetros del clima nos ayudan a definir cada bioma - el sistema simula variables como temperatura, humedad, precipitación, presión atmosférica etc. y genera patrones característicos de cada zona. Estos valores no son estáticos; se modifican según ciclos estacionales que introducen cambios en las condiciones ambientales, además el clima lo controla un agente de Reinforcement. Por ejemplo, cuando llega el invierno a la taiga, la temperatura puede bajar hasta -13°C respecto a la base, mientras que el verano puede subirla hasta +7°C - estos cambios afectan mucho al estado de las entidades.

Para el terreno he usado algoritmos procedurales que generan paisajes variados con los ocho tipos de terreno. Ya hablaré más sobre esto en su sección en la memoria - es bastante interesante. Lo importante es que esta variedad topográfica crea hábitats específicos donde ciertas especies se desarrollan mejor que otras; **no es lo mismo un cactus en la arena que un pino en la nieve**.

> **Ejemplo de Hábitat:** El hábitat `taiga_wetland` se forma en zonas donde el terreno es `WATER_SHALLOW` y está cerca de áreas de `GRASS` y `SNOW`. Es una especie de microambiente donde plantas resistentes al frío pueden nacer. Otro ejemplo, el hábitat `desert_oasis` aparece en áreas de `WATER_SHALLOW` cercanas a `SAND`.

También existen multitud de **indicadores para evaluar la salud de un bioma**, como se verá en el apartado correspondiente.

1.2.2. ¿Qué hace el framework?

Soporte del framework

1. **Simulaciones ecológicas**: Se pueden crear biomas con ecosistemas y *vida*, para simular sus procesos y observarlos a lo largo del tiempo.
2. **Generación de datasets**: También he implementado funcionalidad para producir datasets de simulaciones sobre factores poblacionales y de el entorno, para posteriormente analizar o entrenar modelos de Machine Learning.
3. **Entrenamiento de modelos de IA**: He integrado diferentes interfaces para entrenar y evaluar tanto modelos de Reinforcement Learning, como redes neuronales, con la idea de que pueda aprender patrones emergentes de los biomas.
4. **Visualización interactiva**: He preparado algunas herramientas para observar en tiempo real o a través de snapshots temporales la evolución del ecosistema. Aunque es simple, las epresentaciones gráficas ayudan bastante (terrenos, poblaciones, estadísticas, tendencias...etc). También hay multitud de generación de plots para diversos sistemas (evolución, entrenamientos de modelos...etc).
5. **Experimentación con parámetros**: Es muy fácil modificar variables ambientales, genéticas, de poblacionales..etc,, incluir nuevos agentes, nuevas especies, nuevos biomas... Con el TFG cubro las bases para que un futuro, el framework pueda ser de soporte para estudiar escenarios hipotéticos y sus consecuencia a largo plazo

1.2.3. Enlaces del Proyecto

Sitio Web

Simplemente un punto de entrada (tiene sonido)

https://www.echoes-of-gaia.com/

GitHub

Repositorio

Código fuente

github.com/geru-scotland/echoes-of-gaia

CAPÍTULO 2

Arquitectura, tech stack y sistemas

Indaguemos ahora en la arquitectura general del framework, sin entrar en mucho detalle de implementaciones. Todo esto que voy a exponer, es el esqueleto que hace de soporte para todas las dinámicas y sinergias que más adelante veremos. Con lo que, he decidido obviar ciertas áreas, entre ellas incluida la parte de visualización utilizando PyGame. Prefiero centrarme en una contextualización de la arquitectura en este documento, para que la lectora o lector puedan comprender cómo se sustenta lo que en la memoria técnica se presentará.

En general, la arquitectura se fundamenta en una combinación de patrones de diseño y sistemas que nos permiten modelar y orquestrar. El diseño lo he realizado con objeto de hacer que entidades *vivas* evolucionan, interactúen entre sí y sobre todo, con el entorno.

Considero que una simulación, es algo directamente relacionado a computación. Es por ello que he decidido explicar la arquitectura y los sistemas básico, aunque sea por encima, de cómo he preparado la simulación, antes de entrar en materia.

2.1. Arquitectura

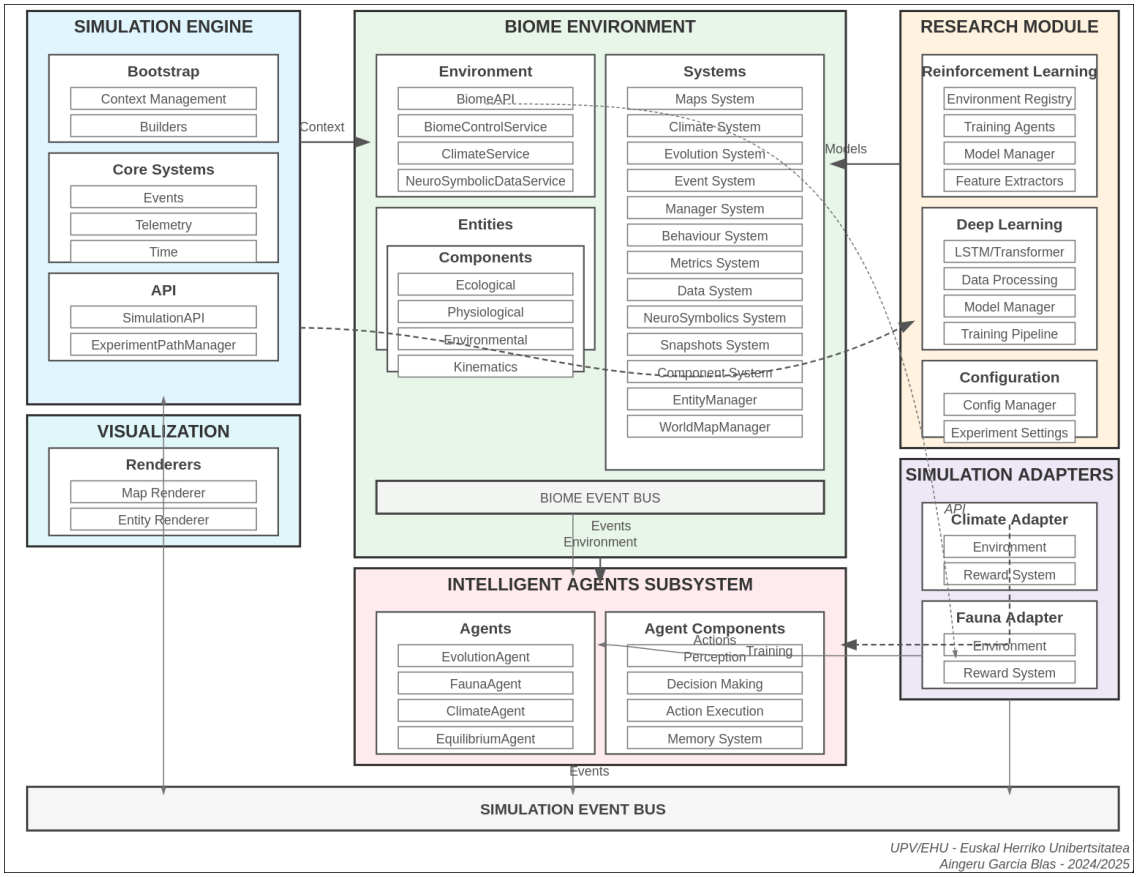


Figura 2.1: Diagrama conceptual de arquitectura

En este diagrama podemos observar la arquitectura general, con los sistemas principales - nótese que obvio muchos sistemas entre los módulos; he intentado reducir a los más generales para poder mostrar la idea general de la arquitectura, aunque en el proyecto la distribución sea algo diferente. Están organizados por módulos para resultar en algo más expresivo e intuitivo. Veamos las 5 agrupaciones generales:

1. **Simulation Engine**: El núcleo que controla el flujo de la simulación, desde el inicio, hasta el final. Se podría ver como **el director**.

2. **Biome Environment**: Si la simulación es el director, **el bioma sería el escenario**. Es el ecosistema virtual y los protagonistas son las entidades con sus componentes pero, gestiona muchos sistemas que hacen que su funcionamiento sea posible, a parte de expandir o extender las funcionalidades disponibles.

3. **Intelligent Agents Subsystem**: Los agentes son entes inteligentes e interactúan con el entorno (el bioma).

4. **Research Module**: Inicialmente lo preparé para ser únicamente un área o *playground* de experimentos, búsqueda automática de mejores hiperpárametros etc., pero conforme he ido desarrollando me ha parecido buena opción incluir aquí el entrenamiento de los diferentes modelos de IA, y exponer APIs para poder acceder a ellos durante la inferencia.

5. **Visualization**: Ya sea para visualizar mapas generados proceduralmente de manera aislada, como para lanzar un visor y poder observar una simulación entera, paso a paso.

La comunicación, como explicaré un poco más adelante, se realiza principalmente con eventos a través de event dispatchers (los buses que se pueden observar en el diagrama). La mayor parte de los sistemas son bastante autónomos, pero cuando se han de comunicar con otros o generar notificaciones a suscriptores, se utilizan estos buses; uno u otro dependiendo del nivel. La idea principal es que sea una arquitectura desacoplada, que es algo que durante el desarrollo se agradece - hacer debugging en módulos separados con pocas dependencias siempre resulta más sencillo. Además, si algo se rompe, es menos probable que afecte al resto.

2.2. Arquitectura multicapa: *Clean Architecture/Onion*

Durante la planificación del proyecto, intenté diseñar desde el principio una arquitectura que siguiera los estándares de **Clean Architecture/Onion**, básicamente creo que el establecer capas *que van hacia dentro* y no conocen a la anterior, es una forma muy práctica de estructurar el sistema de forma coherente, suele ayudar bastante con el mantenimiento, y además, me resulta elegante.

He adaptado dicha arquitectura de manera que esté orientada al patrón ECS (Entity Component System), que explicaré posteriormente. No obstante, el proyecto es grande y la idea era centrarme principalmente en otros aspectos, no en el desarrollo ni en arquitectura, con lo que no he podido ser tan estricto en ciertas ocasiones como me hubiera gustado. Así que, podemos decir que es un tímido reflejo de dichas arquitecturas.

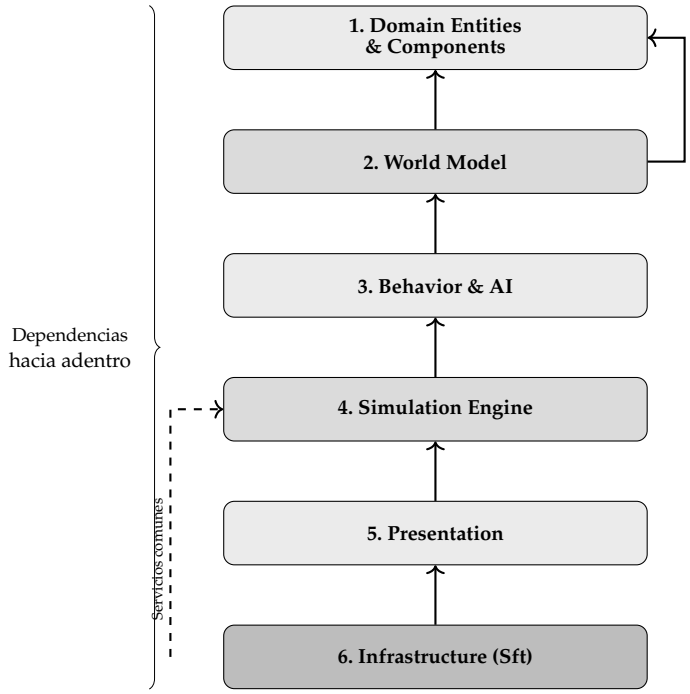


Figura 2.2: Pila de capas: el núcleo son *Domain Entities & Components*.

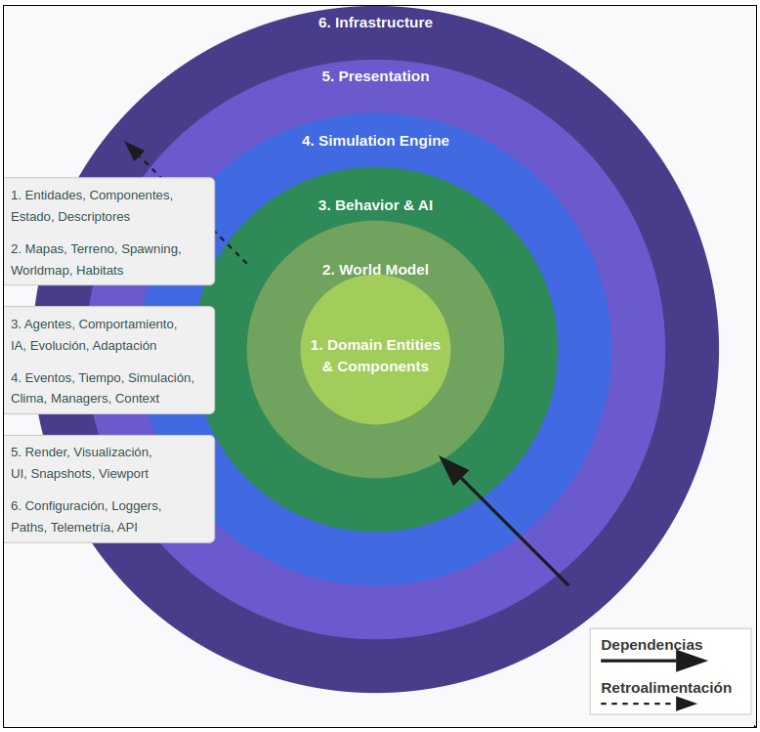


Figura 2.3: Clean-Onion arquitectura

2.2.1. Mapeo de capas - sistemas / servicios

El framework tiene en sí muchos sistemas que lo conforman, más de los que indicaré aquí. He intentado aglomerar en sistemas generales y *útiles* y obviar muchos de más bajo nivel que los hacen posibles. Como mencioné en la introducción, creo que es más conveniente que me centre en los elementos computacionales que no los intringulis de la arquitectura subyacente de la plataforma.

Sin embargo, considero que conviene contextualizar al menos algunos de los sistemas a grosso modo, para reconocer ciertas áreas de la plataforma conforme vayamos avanzando.

Capa	Subsistemas y sub-subsistemas	Servicios

1. Domain entities & components	▪ Entidades
	  • Entidades de flora
	  • Entidades de fauna
	▪ Gestión de entidades
	▪ Generación de entidades
	▪ Descriptores de entidades
	▪ Estado de entidades
	▪ Ciclo de vida
	▪ Componentes base
	▪ Componentes fisiológicos
	  • Metabolismo y fotosíntesis
	  • Nutrición autótrofa y heterótrofa
	  • Vitalidad y salud
	  • Crecimiento y desarrollo
	  • Sistemas de estrés
	  • Sistemas de energía
	▪ Componentes ambientales
	  • Adaptación al clima
	  • Dormancia y hibernación
	▪ Componentes cinemáticos
	  • Transformación espacial
	  • Movimiento y desplazamiento
	▪ Gestores de componentes
	  • Gestión de ciclo vital
	  • Gestión de crecimiento
	  • Gestión de metabolismo
	  • Gestión de nutrición
	  • Gestión de movimiento
	  • Gestión de adaptación al clima
		▪ Registro de entidades
		▪ Gestión del ciclo de vida
		▪ Creación de entidades
		▪ Estado del ecosistema
		▪ Persistencia de estado
		▪ Registro de componentes
		▪ Comunicación entre componentes
		▪ Sincronización de estados
		▪ Notificación de eventos de componentes

2. World model	▪ Biomas
	▪ Climático
	  • Gestión de estaciones
	  • Eventos climáticos
	  • Factores ambientales
	▪ Mapas
	  • Generación procedural de terreno
	  • Gestión de hábitats
	  • Asignación de posiciones
	  • Gestión del mundo
	▪ Métricas ecológicas
	  • Colectores de datos
	  • Analizadores de ecosistema
	  • Historial climático
	  • Métricas de biodiversidad
	▪ Algoritmos genéticos
	  • Genes y mutaciones
	  • Cómputo de fitness
	  • Control de población
	  • Seguimiento evolutivo
		▪ Estado climático
		▪ Datos del bioma
		▪ Historial climático
		▪ Control del ecosistema
		▪ Scores del bioma
		▪ Cálculo de estadísticas
		▪ Normalización

Capa	Subsistemas y sub-subsistemas	Servicios

3. Behavior & ai	▪ Agentes
	  • Agentes climáticos
	  • Agentes de evolución
	  • Agentes de fauna
	  • Agentes de equilibrio ecológico
	▪ Comportamientos
	  • Búsqueda de alimento
	  • Persecución y huida
	  • Exploración
	▪ Análisis de tendencias evolutivas
	▪ Aprendizaje por refuerzo
	  • Adaptadores (entornos aislados)
	  • Entornos de entrenamiento
	  • Modelos de decisión
	  • Interfaces de entrenamiento
	  • Api de inferencia
	  • Extracción de características
	▪ Neurosimbólico
	  • Integración de conocimiento simbólico
	  • Módulos neurales y simbólicos
	  • Balanceador de decisiones
	  • Gestor de intervenciones
	▪ Deep learning
	  • Modelos predictivos
	  • Preprocesamiento de datos
	  • Inferencia
	▪ Control difuso (fuzzy logic)
		▪ Datos para modelos
		▪ Predicción
		▪ Coordinación de agentes
		▪ Aprendizaje
		▪ Integración neurosimbólica
		▪ Gestión de tendencias
		▪ Control inteligente de población

4. Simulation engine	▪ Orquestación de simulación ▪ Temporal ▪ Eventos de simulación ▪ Telemetría ▪ Snapshots del ecosistema ▪ Bootstrap y contexto ▪ Gestión de experimentos	▪ Ciclo de vida de simulación ▪ Manejo de eventos ▪ Temporización ▪ Recolección de métricas ▪ Gestión de rutas de experimentos

5. Presentation	▪ Renderizado
	▪ Motor de render
	▪ Gestión de tiles
	▪ Visor de snapshots
	▪ Eventos de renderizado
	▪ Visualización de datos
	  • Visualización de evolución
	  • Visualización de telemetría
	  • Visualización de tendencias
	▪ Interfaz de usuario
		▪ Gestión de renderizado
		▪ Procesamiento de eventos visuales
		▪ Representación gráfica
		▪ Conversión de datos a gráficos

6. Infrastructure (soft)	▪ Configuración
	▪ Logging
	▪ Middlewares
	▪ Paths
	▪ Normalización de datos
	▪ Tipos compartidos
	▪ Utilidades matemáticas
	  • Funciones de crecimiento biológico
	  • Funciones estadísticas
	  • Cálculos ecológicos
	▪ Eventos
	  • Bus de eventos
	  • Notificador de eventos
	  • Manejador de eventos
	▪ Enumeraciones y constantes
	  • Definiciones del bioma
	  • Enumeraciones de tipos
	  • Configuración de temporizadores
	  • Umbrales del sistema
	▪ Manejo de excepciones
		▪ Gestión de rutas
		▪ Notificación de eventos
		▪ Middlewares
		▪ Manejo de excepciones
		▪ Medición de rendimiento
		▪ Carga de datos

2.3. Tech stack

He programado muchos años en C++ y me ha sido realmente tentador utilizarlo para el proyecto, no solo por su eficiencia si no porque es con lo que más familiarizado estoy y es un lenguaje que me apasiona. Además he observado que con el tiempo ha ido adquiriendo algo más de soporte y cariño en cuanto al Machine Learning - no obstante, por lo que he podido ver, está muy lejos de disponer del ecosistema que tiene Python. Con lo que me decidí por animarme con Python, además ahora tiene un mejor soporte para tipado, lo cual me agrada mucho. También, durante tercer y cuarto curso de carrera hemos trabajado un poco con Python.

Versión: Python 3.12.3

Veamos el tech stack que he utilizado - he intentado mapearlo de manera que se pueda asociar a la imagen 2.1. Existe quizá un poco de redundancia y es posible que haya utilizado alguna herramienta en más partes del proyecto. Pero considero que para plasmar una idea general, es un buen recurso:

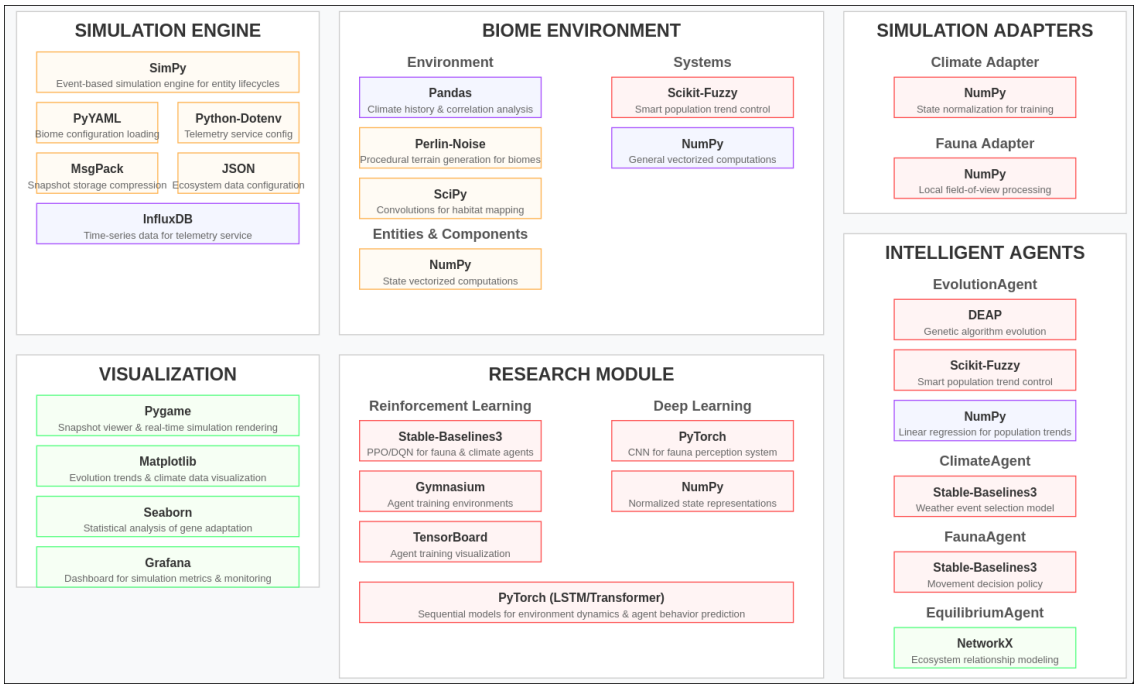


Figura 2.4: Tech stack por módulo

No hay mucho que explicar - la elección de NumPy, Pandas, SimPy... entre otros es bastante estándar y no existen realmente otras alternativas mejores. Además, las he ido utilizando poco a poco en los 2 últimos años de carrera - al igual que Pytorch y Stable Baselines 3 o incluso Scipy/Scikit. He optado por seguir profundizando con estas herramientas - son ampliamente utilizadas tanto en entornos de producción, como en ámbitos de investigación, tanto por su accesibilidad como por su tremenda eficiencia al estar la mayoría implementadas internamente en C y utilizar técnicas de optimización extrema.

Infiero que en el futuro las utilizaré, y mucho, con lo que me he animado intentar resolver los problemas que se me han ido planteando en el proyecto con ellas.

Para la parte del visor, me habría gustado hacerlo con Unreal Engine 5, que es un entorno que conozco y estoy cómodo, o incluso Godot. Sin embargo, por agilizar el proceso, me decanté por Pygame para poder disponer de un entorno *Pythonic*, al tener la mayoría de herramientas y librerías ya incorporadas. Es posible gestionar, mediante plugins y otros mecanismos, modelos preentrenados en UE5 y/o Godot, pero nunca he utilizado y he preferido no dedicar tiempo a ésta integración.

2.4. Patrones de diseño relevantes

Duarante el desarrollo del proyecto, he seguido multitud de patrones de diseño que he considerado oportunos (builder para el contexto, factoria para crear entidades, inyección de dependencias...). Pero voy a explicar los que considero más relevantes y notables a lo largo de todo el codebase:

2.4.1. Entity Component System (ECS)

Es un patrón que está muy extendido en el desarrollo de Videojuegos, en diferentes sabores pero, tanto Unreal Engine, como Unity o Godot lo implementan. Creo que es un patrón que encaja perfectamente en este contexto.

El núcleo del sistema implementa una variación del patrón, lo he planteado tal que toda entidad biológica (como flora y fauna) se constituye de componentes especializados que definen su estado fisiológico, ambiental, kinético, y muchos otros.

La idea es encapsular comportamientos específicos como metabolismo, crecimiento, adaptación climática, nutrición, movimiento, transformaciones...etc. El sistema opera sobre conjuntos de estos componentes, para orquestar de manera centralizada; se podría haber hecho de muchas formas diferentes, pero considero que esta aproximación está testada y ha demostrado en entornos de videojuegos y/o emulación proveer de mucha flexibilidad tanto para representar diversidad de factores para una entidad, como para extender su comportamiento y funcionalidades. En mi caso, tal y como he diseñado los componentes, creo que ayuda a **modularizar la diversidad biológica**, como veremos en el capítulo de Modelos Computacionales en la memoria técnica.

Hablo de modularidad, ya que la idea es implementar un componente con unas reglas y protocolos definidos en los procesos base, de manera que, siguiendo dichas reglas, se pueden extender componentes de forma muy sencilla Además, es un sistema que resulta en una especie de *LEGO*; puedo quitar y poner componentes a las entidades a demanda - lo cual, hace muy buena sinergia con el método de configuración que he preparado a través de ficheros `yaml`.

2.4.2. Event Driven Architecture (EDA)

Este es otro patrón que creo es relevante mencionar, no solo porque esté muy extendido en proyectos y metodologías de desarrollo de hoy en día, si no porque además me permite mantener muy modular y *a capas* la arquitectura; se basa en un mecanismo de eventos que desacopla los emisores de los receptores y produce el efecto de sistema reactivo (`ON_WEATHER_UPDATE`, `ON_ENTITY_DEATH`). Lo he diseñado para dar soporte a tres niveles de buses de eventos:

BiomeEventBus: gestiona eventos locales al ecosistema, por ejemplo cambios climáticos, nacimientos (**spawns** a partir de ahora) o muertes de entidades.

SimulationEventBus: por otro lado, éste coordina eventos a nivel de simulación general - desde ciclos temporales, programación de snapshots...etc

Ejemplo simplificado:

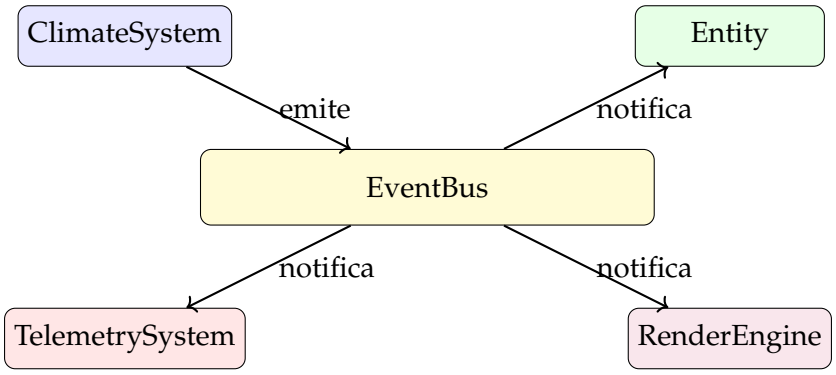


Figura 2.5: Sistema de comunicación basado en eventos

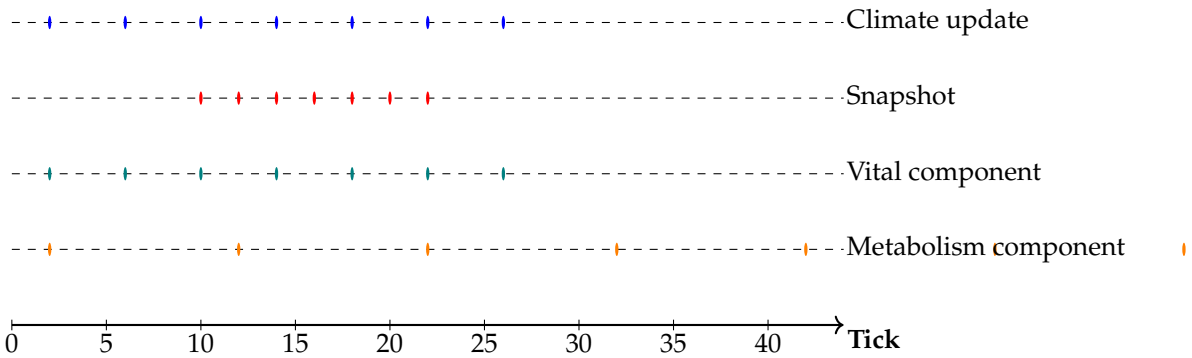
EventNotifier: éste opera a nivel de entidad, es el nexo de unión que hace que los componentes se puedan comunicar entre sí dentro de una misma entidad biológica.

2.5. Sistemas principales

A continuación, voy a presentar los sistemas que considero puede ser de utilidad conocer para disponer de un buen contexto durante el resto de la memoria técnica. Se podría decir que con sistemas intrínsecos al **core**, otros más específicos se irán presentando en sus capítulos correspondientes.

2.5.1. Motor de simulación

Para manejar todos estos subsistemas interactuando simultáneamente, necesitaba un motor que pudiera coordinar eventos a diferentes escalas temporales. La simulación requiere de una *noción* temporal y ordenar los eventos de manera que, algo que yo quiero simular que ocurre cada mes, ese espacio temporal sea consistente con lo que ocurre cada 15 días, ya que los ticks de la simulación transcurren a un nivel abstracto de tiempo.



Todo eso puede representar un espacio de tiempo de un mes, pero realmente se ejecutará en nanosegundos.

Como se puede observar, SimPy nos ofrece un entorno de eventos discretos donde tenemos *ticks* específicos cuando ocurren eventos (importante esto, si estamos en el evento 2 y el siguiente evento es en el tick 15, SimPy saltará directamente a este punto), no lo hará de manera continua.

Encaja perfectamente con la simulación que se intenta hacer; tenemos diferentes procesos a distintas escalas temporales. Esta orquestación de procesos se hace con un mecanismo de planificación basado en generadores (*yield* de Python). Cada proceso es un generador que puede dormir hasta que llegue su momento de ejecutarse.

En Echoes of GaIA, he planteado el espacio temporal de manera que un tick representa un intervalo básico de tiempo, y con ello:

- 2 ticks: Un día completo
- 60 ticks: Un mes (30 días)
- 720 ticks: Un año (12 meses)

Adicionalmente, he introducido el concepto de **Eras**, donde una era es un espacio de tiempo grande, en este caso X eventos (definido en configuración).

```
[SIMULATION][Main thread] 2025-04-28 03:53:21,436 - simulation - INFO - Simulated events: 120, Era: 2, Event in Era: 20 in 102.79 seconds
```

Figura 2.6: Eras y eventos

Lo bonito y útil, es que los procesos se ejecuten a su propio ritmo - cada uno establece cuando quiere ser lanzado y se gestiona así mismo, simplemente indicándole al entorno de SimPy cuando quiere ejecutarse. Todos estos procesos se ejecutan concurrentemente - mediante el loop de eventos - y SimPy se encarga de que se sincronizan correctamente en su línea de tiempo compartida.

Por ejemplo, los eventos climáticos o el metabolismo de la flora se actualizan prácticamente cada 2 ticks, mientras que los agentes de evolución de cada especie tienen tiempos variables.

Este sistema de eventos tiene muy buena sinergia con la comunicación mediante dispatchers/notifiers; los componentes, procesos de simulación y otros muchos sistemas pueden hablar sin problemas a través de los buses y, pese a estar desacoplados, éste orden temporal les da soporte para una correcta sincronización.

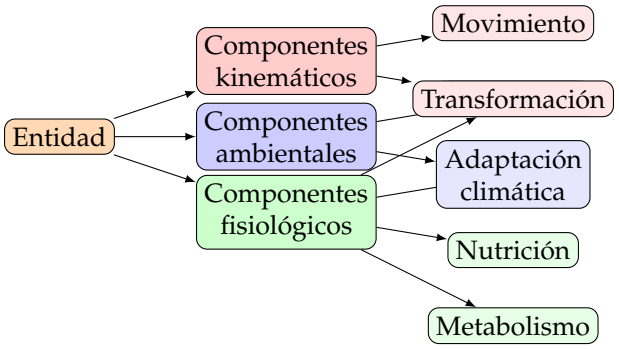
A modo de ejemplo rápido, algunos eventos que he preparado para diferentes situaciones:

- `WEATHER_UPDATE`
- `ENTITY_DEATH`
- `SEASON_CHANGE`
- `EVOLUTION_CYCLE`

Los tiempos que he utilizado, son una vertiente un tanto simplista que no busca granularizar los procesos, si no observar cambios a lo largo del tiempo. No obstante - y esto es parte positiva de preparar un framework - esto es configurable y se podria modificar.

2.5.2. Entidades y componentes

La base de todo el sistema son las entidades, al fin y al cabo son sus procesos biológicos e interacciones con el entorno lo que queremos modelar. Como se ha mencionado anteriormente, cada planta y animal tiene asociados una cantidad variable de componentes. Gracias a este enfoque puedo construir organismos complejos de manera modular:



Lo interesante de este diseño es cómo los componentes interactúan entre sí **sin acoplarse directamente** gracias al notificador de eventos local a cada entidad. Por ejemplo, cuando un animal sufre daño:

1. El componente `VitalComponent` registra el daño.
2. Emite un evento a través del event notifier de la entidad.
3. Otros componentes, como el de metabolismo, reaccionan ajustando sus parámetros. Solo aquellos que se han registrado a ese evento concreto.

La magia es el cómo todo esto nos permite gestionar cadenas relativamente complejas de causa-efecto; si la temperatura baja mucho, el componente de adaptación al clima de una entidad emite un **evento de estrés** que el componentes vital y metabólico reciben, lo que, de continuar este estado, reducirá la capacidad de fotosíntesis de la planta.

Para las especies de flora

- `PhotosyntheticMetabolismComponent` - gestión de la producción de energía.
- `GrowthComponent` para el desarrollo físico.

- `AutotrophicNutritionComponent` - absorción de nutrientes, nutrición sin ingesta externa.

Para la fauna

- `MovementComponent` - desplazamiento
- `HeterotrophicNutritionComponent` para alimentación, pero esta sí, con ingesta de alimento externo.

Existen otros componentes comunes a ambos, como el `VitalComponent` para la salud y estrés, o el `WeatherAdatptationComponent`, para la adaptación térmica, resistencias calor frio, gestión de estrés térmico etc. También hay otros más miscelaneos, el `Transform` por ejemplo, que sirve para gestionar posiciones de entidades en los mundos.

Como se verá más adelante, gracias a estas mecánicas locales de entidad, se puede llegar a conseguir que a largo plazo se termine modificando el clima del bioma. Un ejemplo rápido: si el bioma se queda sin flora y existe fauna, la fauna producirá CO_2 al respirar - al no haber flora haciendo fotosíntesis esa emisión de CO_2 no podrá ser mitigado lo que repercutirá en una alteración de los factores medioambientales (variables como temperatura, humedad etc).

Ejemplo de combinación de componentes simplemente por contextualizar un poco más

Una especie de **cactus del desierto**:

- **GrowthComponent**: Tamaño máximo grande
- **VitalComponent**: Alta vitalidad y baja tasa de envejecimiento
- **PhotosyntheticMetabolismComponent**: Con alta eficiencia de fotosint. y baja tasa de respiración
- **WeatherAdaptationComponent**: Con alta resistencia al calor y baja al frío
- **AutotrophicNutritionComponent**: Con alta capacidad de absorción de nutrientes y baja toxicidad

Por otro lado, un **lobo ártico**, por ejemplo:

- **GrowthComponent**: Tamaño medio
- **VitalComponent**: Con vitalidad baja y tasa de envejecimiento moderada
- **WeatherAdaptationComponent**: Con alta resistencia al frío y baja al calor
- **HeterotrophicNutritionComponent**: Con alta tasa de hambre y sed moderada

El funcionamiento e implementación de los componentes, será visto en el apartado de Modelos computacionales de la memoria técnica - también se hablará sobre más parámetros que estos gestionan.

2.5.3. Mundo y mapas

El mundo donde viven las entidades está estructurado como un **grid bidimensional** gestionado por varios sistemas. El mapa final, es un *layered map*, esto es, está constituido por información que proveen múltiples capas: de **terrenos**, de **entidades**, de **índices** y de **hábitats**:

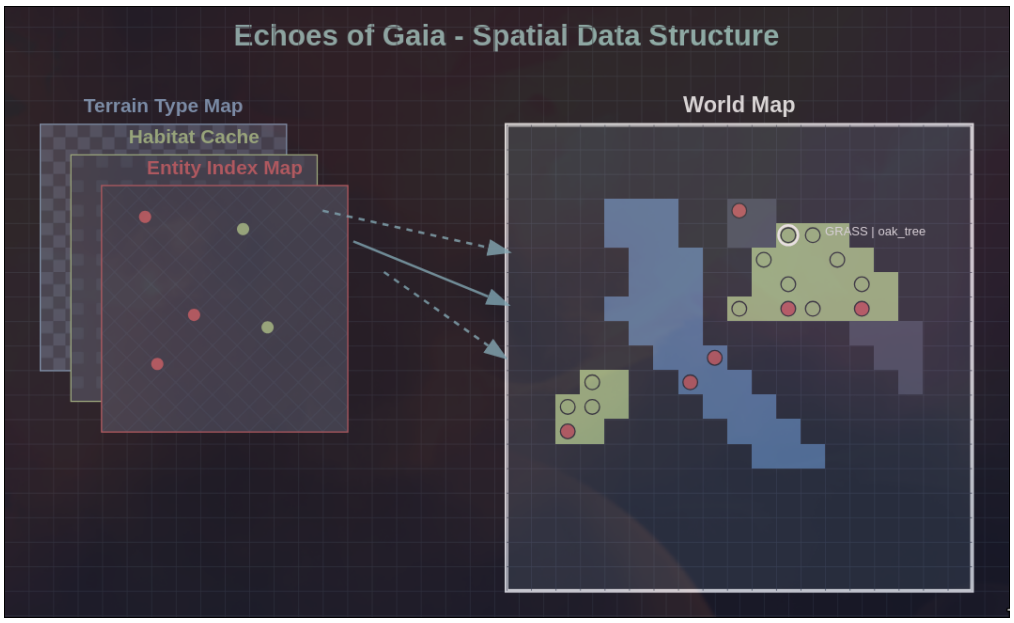


Figura 2.7: Ejemplo simplemente conceptual

En la imagen 2.7 podemos observar como cada registro, proyecta su contenido al worldmap.

Realmente, el cómo lo gestiono es un poco diferente, ya que dispongo diferentes clases encargadas de los registros per-se, aunque como se verá más adelante en el apartado de **reinforcement learning**, hago uso de un cómputo de mapas/canales para extraer características de un campo de visión local. Veamos un poco más en detalle el contenido y forma de los mapas de entidades y terrenos:

Mapa de indices de entidades						
0	14	15	22	23	16	0
0	17	3	4	24	18	0
0	19	5	25	6	20	0
0	0	26	7	8	21	0
0	0	27	9	10	0	0
0	0	11	12	13	0	0

ID	Tipo	Especie
3-8	Fauna	Pantera
9-21	Flora	Roble
22-27	Flora	Hongos
0	-	Celda vacía

Figura 2.9: Index map

Mapa de Terrenos						
0	2	3	4	4	4	4
0	2	3	4	4	4	4
0	2	3	4	4	4	4
0	2	3	4	4	4	4
0	2	3	4	4	4	5
0	2	3	4	4	5	5

ID	Tipo de eerreno	Descripción
0	WATER_DEEP	Agua profunda
2	WATER_SHALLOW	Agua superficial
3	SHORE	Costa/Orilla
4	GRASS	Pradera/Hierba
5	MOUNTAIN	Montaña
8	UNKNOWN	Terreno desconocido

Figura 2.11: Terrain map

Nota: Las IDs de los terrenos están pre-establecidas en la configuración, pero las de las entidades son identificadores dinámicos que se asignan en tiempo de ejecución.

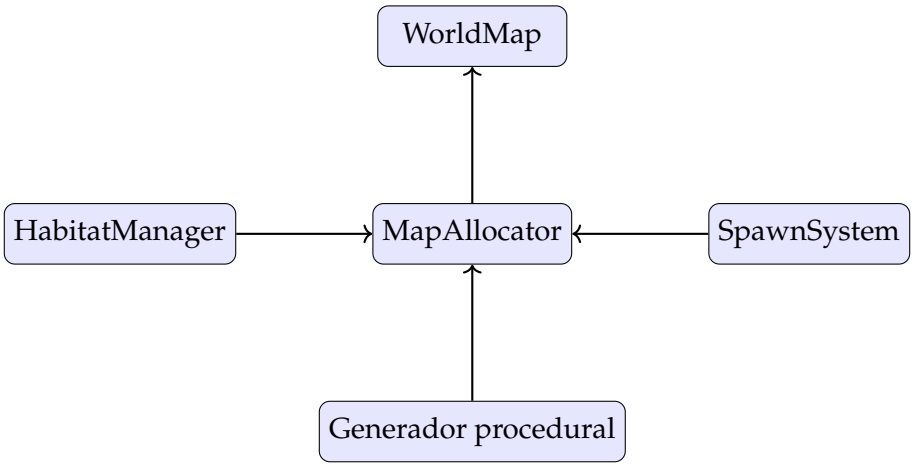


Figura 2.12: Sistema de gestión espacial

El mundo se construye comenzando con un generador de mapas procedurales que utiliza el algoritmo de Perlin Noise para crear terrenos muy variados (explicado en detalle en la memoria). Una característica importante es que los terrenos no son meramente estéticos; si no que pueden ser recursos (agua) y el cómo se combinen determina los diferentes hábitats.

Existe un sistema llamado `MapAllocator` que se basa en gestionar posiciones del mapa para las entidades, el cual tiene comunicación con un sistema de gestión de hábitats para enriquecer los spawns de las mismas. Por ejemplo, un cactus solo puede crecer en desierto, pero un pino necesita terreno montañoso o de bosque. También previene que las entidades ocupen el mismo espacio, esto es - **gestión básica de colisiones**.

Algo que creo oportuno mencionar también, es el sistema de `HabitatManager`, que precomputa qué zonas del mapa son adecuadas para cada tipo de hábitat.

Entre todos ellos, hablan, y se coordinan para, con el `SpawnSystem`, instanciar las entidades en el mundo - no es un proceso trivial, ya que requiere de muchas comprobaciones y además de una correcta gestión y análisis de los componentes a asignarle. Veamos un diagrama del proceso simplificado:

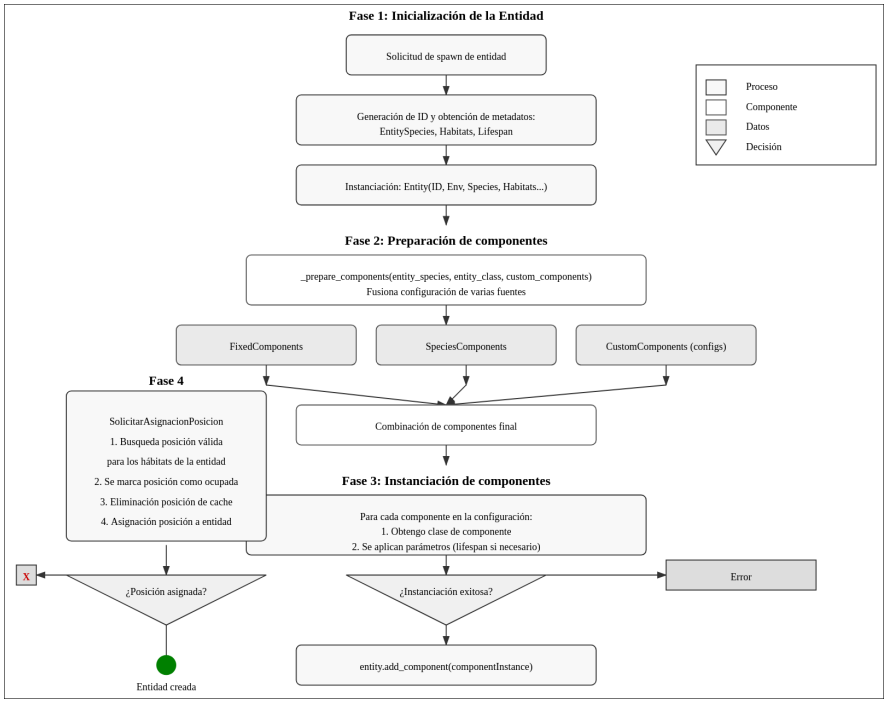


Figura 2.13: Se puede ampliar o hacer click para ver mejor

2.5.4. Sistema climático

El sistema climático es una parte muy relevante en la simulación, que además afecta a diversos subsistemas. Incluso dispone de un agente que lo guiará, como se verá en el tema de los agentes. En general, podríamos decir que determina las condiciones ambientales que afectan directamente a todos los organismos del Bioma.

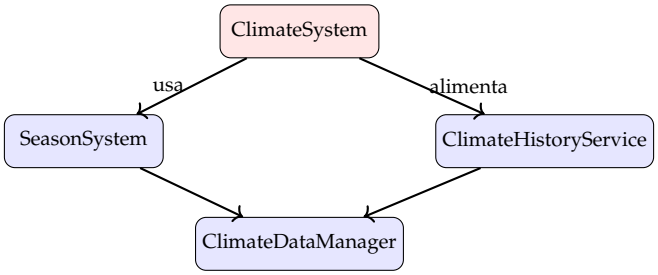


Figura 2.14: Sistema climático

Para solidificar - esas condiciones ambientales que menciono son los factores ambientales que definen el clima - **son varias dimensiones:**

- Temperatura
- Humedad
- Precipitación
- Presión atmosférica
- CO_2
- Biomasa
- Densidad de fauna

El `ClimateSystem` se encarga de gestionar el clima dinámico; para ello simula estaciones en consonancia con el SeasonSystem, y procesa la información con respecto a eventos meteorológicos que el agente climático le dictará. También se encarga de actualizar los factores ambientales en función de la actividad biológica en el bioma (transpiración de la flora, emisión de CO2...etc). Resúmen del flujo:

1. Cuando el clima cambia, emite eventos (`BiomeEventBus`)
2. Los componentes de adaptación climática en **cada entidad** son notificados de estos eventos.
3. Se desencadenan ajustes en parámetros como eficiencia fotosintética, estrés térmico...etc

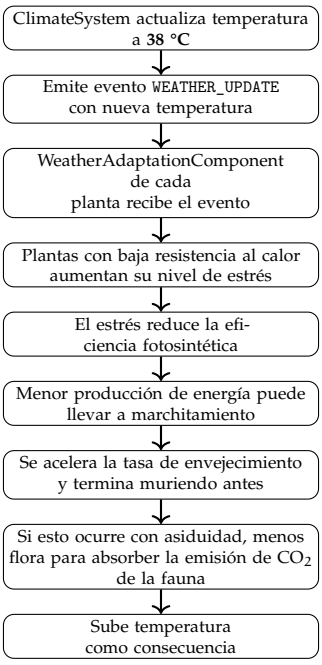


Figura 2.15: Por ejemplo, una ola de calor podría afectar a las entidades así

Pero, adelantando acontecimientos - gracias al sistema de evolución, la cosa puede cambiar en un futuro e incrementar las resistencias. Aprovecho este punto para mencionar aquí que también implementé un `ClimateHistoryService` que registra datos climáticos a lo largo del tiempo. Esto es muy importante para el proceso evolutivo, como veremos más adelante; va a permitir hacer observaciones ponderadas, a la hora de determinar si las especies se adaptan gradualmente a tendencias climáticas. Expondré más en su apartado correspondiente.

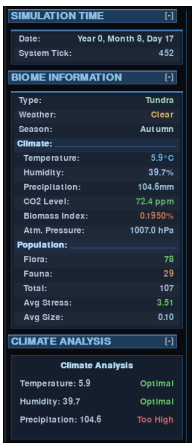


Figura 2.16: Muestra de clima en el visor (click para ver más grande)

2.5.5. Snapshots, visualización y telemetría

Existe todo un proceso de recolección de datos, ya sea de entidades, del bioma, climáticas y además estos datos se procesan, se calculan estadísticas y se almacenan. Para mostrar la idea general, veamos un diagrama meramente ilustrativo, sin entrar en detalle:

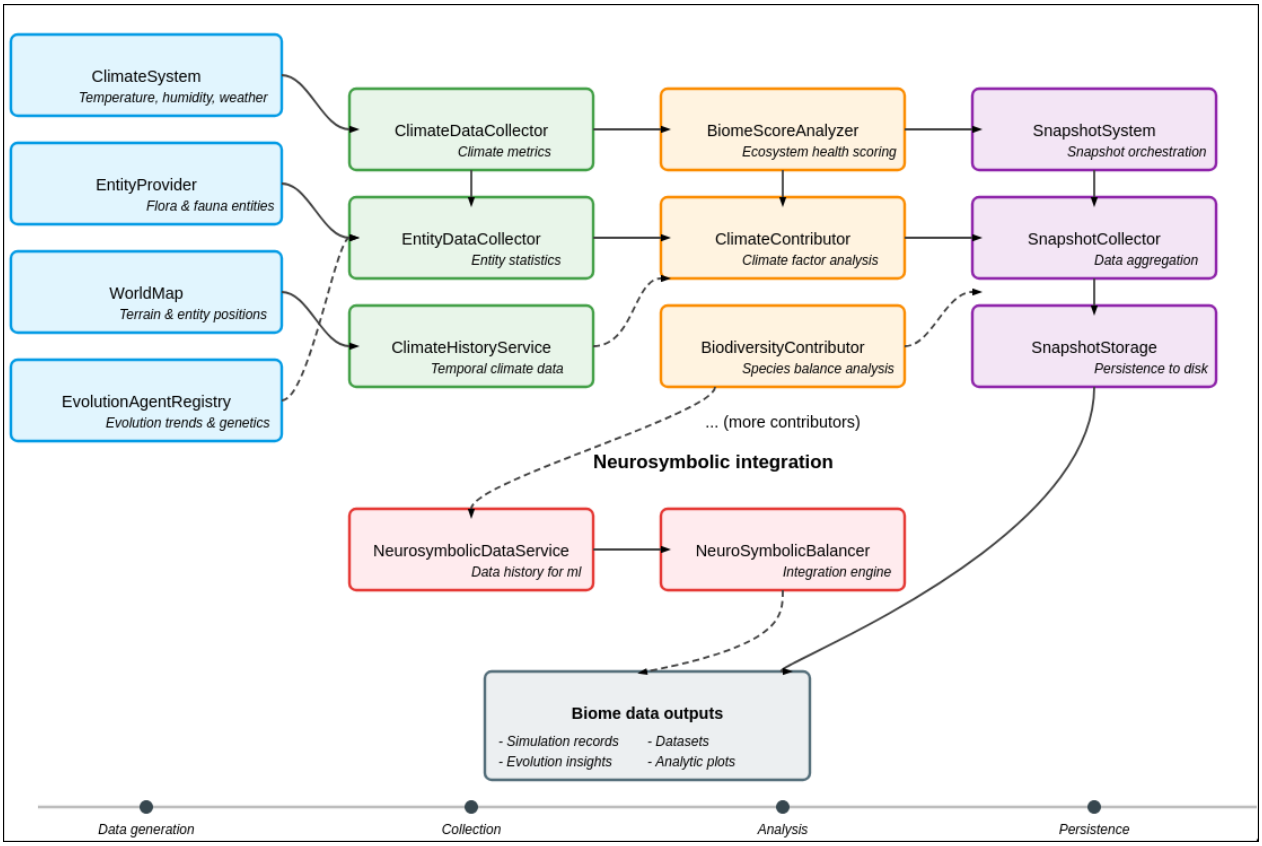


Figura 2.17: Data collectors

De este proceso, se nutren sistemas como el climático, el sistema de snapshots, el de telemetría o el sistema del agente de Equilibrium, el neurosimbólico.

2.5.5.1. Sistema de snapshots

El sentido que le he dado a los snapshots en el proyecto, no es más que el de hacer un **volcado del estado general del bioma en puntos marcados** a lo largo del tiempo, durante la simulación. También **deshidrato** las entidades para plasmar sus estados en el volcado y calculo estadísticas generales. La idea inicial era la de únicamente generar estos snapshots para disponer de un dataset en crudo para posteriormente poder hacer tareas de Machine Learning. Pero he terminado utilizándolos para varias cosas - han sido muy útiles en el visor de la simulacíon; de ésta manera se puede observar más intuitivamente lo que está ocurriendo en el bioma.

Existe un proceso periódico (configurable) que se encarga de volcar el estado completo del bioma. Está constituido por lo siguiente:

- Metadatos de la simulación (identificador, timestamp, tiempo de simulación...etc)
- Mapa de terreno completo
- Estado detallado de cada entidad viva
- Datos climáticos y ambientales
- Métricas generales del ecosistema
- Score del bioma

Para su persistencia, he implementado un *mini-pipeline* de **compresión basado en `msgpack`** (transforma JSON a bloques binarios) y `gzip` que reduce mucho el tamaño final con el que los almaceno.

Veamos un fragmento de un snapshot **antes de su compresión** (los mapas de terrenos se guardan en fichero independiente):

Ejemplo de snapshot

```
{
  "snapshot_id": "1681931456_480",
  "simulation_time": {
    "raw_ticks": 480,
    "month": 2,
    "year": 0
  },
  "creation_timestamp": 1681931456,
  "biome_type": "tropical",
  "current_season": "SPRING",
  "climate_averages": {
    "avg_temperature": 23.8,
    "avg_humidity": 64.2,
    "avg_precipitation": 45.1,
    "co2_level": 420.3,
    "biomass_index": 0.37,
    "atmospheric_pressure": 1015.2
  },
  "climate_analysis": { ... },
  "entities": {
    "12": {
      "id": 12,
      "type": "FLORA",
      "species": "oak_tree",
      "state_fields": { ... },
      "components": { ... }
    },
    "13": { ... }
  },
  "biome_score": {
    "score": 6.8,
    "quality": "healthy",
    "contributor_scores": { ... }
  },
  "metrics": { ... }
  ....
  ....
}
```

2.5.5.2. Sistema de visualización

Hay multitud de herramientas que he utilizado para poder visualizar los datos/resultados de una manera más amena, desde generación de plots con matplotlib o seaborn para analíticas, como la utilización de PyGame como motor de render para el visor. Explico brevemente la UI del visor:

2.5.5.3. Visor de Snapshots

En el visor, **rehidrato** el entorno desde un fichero msgpack.gz con el snapshot - con ello, se reconstruyen los objetos originales y nos permite visualizar toda su información y navegar a a través de los estados secuencialmente (como *TODO* tengo pendiente implementar un sistema de **lazy load** para los mismos).

Como podemos observar en la figura 2.18, disponemos de todo tipo de información para ese instante con respecto al bioma y, con respecto a la entidad seleccionada (marcada con un circulo en blanco). Conforme vamos avanzando snaphosts podemos observar los cambios que sufren ambos.

Figura 2.18: Visor básico, datos generales + entidad seleccionada

Existe una opción en los ficheros de configuración para que los cuerpos de las entidades no desaparezcan al morir, y así poder hacer diferentes comprobaciones y análisis:

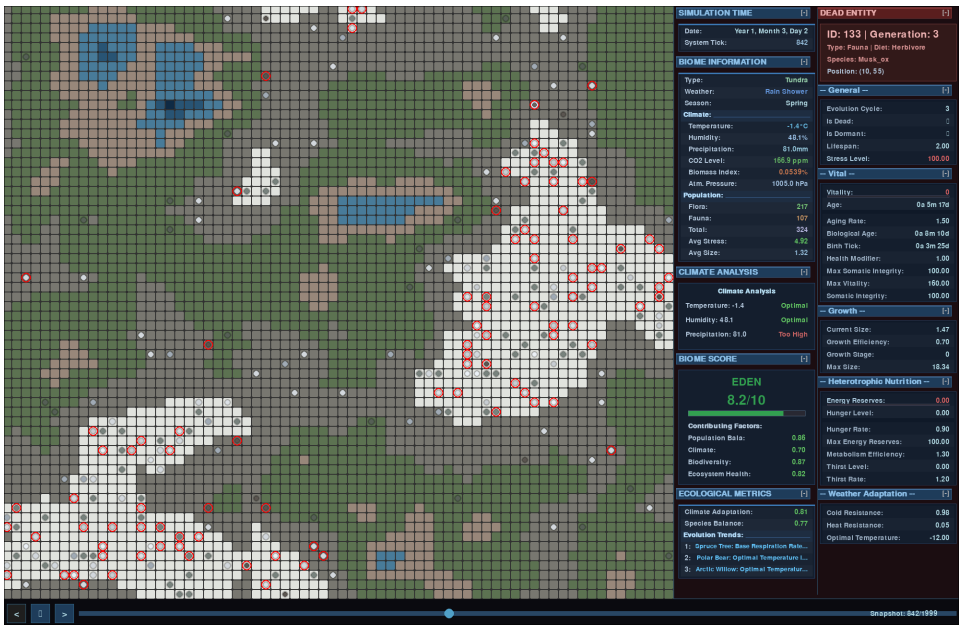
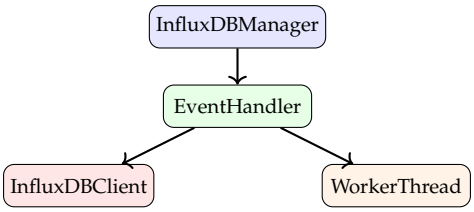


Figura 2.19: Ejemplo de entidades muertas marcadas y visibles

2.5.5.4. Visualizaciones analíticas

Además del visor, existen multitud de funcionalidades que han sido extendidas para proveer soporte de generación de **plots**: **seguimiento de tendencias evolutivas, análisis de cruces genéticos, gráficas de crecimiento, visualización de adaptaciones climáticas, representación de métricas del ecosistema** etc.

2.5.6. Sistema de telemetría



Este sistema se nutre también de los recolectores de datos y almacena en tiempo real métricas de la simulación; envía los datos a **InfluxDB**, quien los inserta en *buckets* en forma de mediciones. Hago esto para poder hacer una visualización de las métricas en **Grafana**, o bien a tiempo real o bien posteriormente pero, **quería que al menos el framework tuviera soporte para ello**. **Para evitar bloquear el hilo principal**, uso un hilo separado que procesa los datapoints de forma **asíncrona**: los eventos generan puntos (los **datapoints**) que se encolan y el worker thread los escribe en InfluxDB. Aunque Python tiene GIL (Global Interpreter Lock), al ser operaciones I/O el enfoque multihilo funciona sin problemas. Cada datapoint incluye la medición, etiquetas, campos y timestamp, para poder hacer consultas y análisis temporal. Aquí como hago consultas.

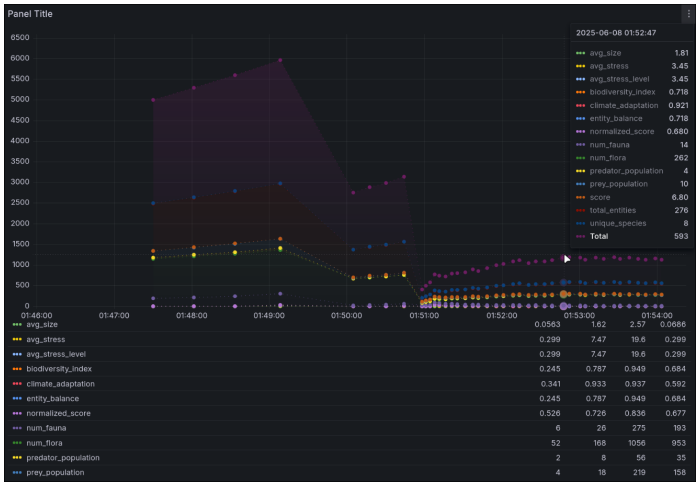


Figura 2.20: Grafana leyendo de InfluxDB

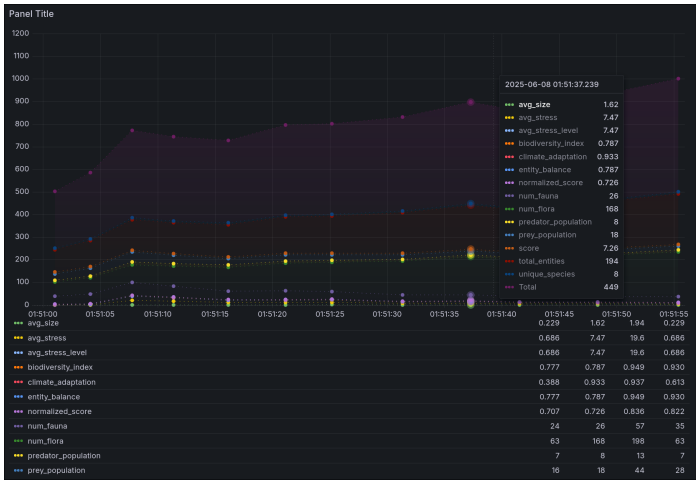


Figura 2.21: Otro ejemplo

2.5.7. Sistema de Bootstrapping y contexto

El arranque de la simulación requiere también de un método bien diseñado para que no haya problemas y además, hacer que **cada módulo tenga acceso únicamente a su información**, con lo que hice uso del patrón de diseño de `Context`. Por supuesto, la idea de hacer un sistema de bootstraping es la de desacoplar la inicialización del sistema, de su ejecución; agiliza mucho el proceso de debugging y ayuda a encontrar fallos mucho más rápidamente.

Por lo demás, he preparado un diagrama simplificado que nos puede hacer de tour para visualizar el flujo de inicialización de los sistemas de la simulación.

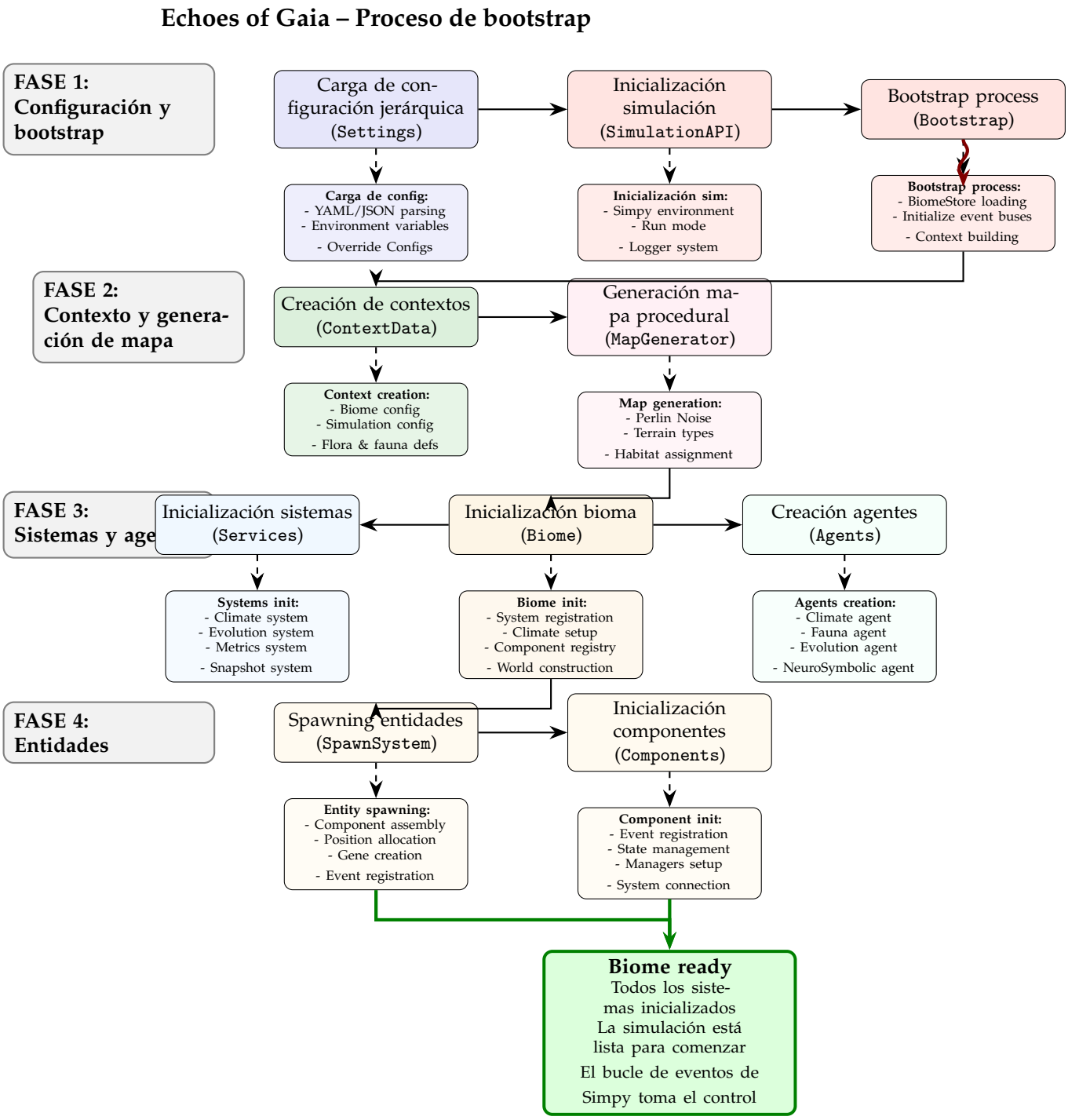


Figura 2.22: Ejemplo de arranque - bootstrap de la simulación

Para todo ello, dispongo un un **logging** exhaustivo y **middlewares** de medición de tiempos, simplemente por apoyo.

2.5.8. Parámetros y configuración

Un bioma artificial implica una cantidad de parámetros considerable que afectan a la simulación. Por ello desarrollé un sistema de configuración y parseo basado en YAML y JSON ya que suelen ofrecer mucha flexibilidad y disponen de soporte en python para gestionar de forma muy sencilla:

- **Parámetros por defecto**: Existen muchos parámetros en el sistema, y los valores por defecto están centralizados en el archivo `ecosystem.json`
- **Configuración global distribuida**: Parámetros generales de la aplicación, distribuidos entre varios subsistemas (settings, rendering...etc)
- **Configuración de simulación**: Específicos del motor de simulación, desde número de eras y eventos, hasta generación de dataset y/o visualizar analíticas.
- **Perfiles de biomas/escenarios**: Configuraciones customizables por el usuario.

Escogí YAML para la configuración que el usuario del frameworkk tiene que utilizar, ya que lo encuentro más intuitivo y fácil de modificar. Sin embargo, para hacer de *mini base de datos* y guardar todos los valores por defecto para los parámetros, pienso que JSON y su formato explícito de diccionario es más sencillo para procesar programaticamente.

Lo genial de esto es que se pueden crear muy fácilmente diferentes configuraciones y lanzar simulaciones totalmente customizadas, dependiendo del experimento o estudio que se quiera hacer (esto ha sido de grandísima ayuda en la fase de entrenamiento del agente de fauna, como se verá en su capítulo, para poder generar biomas aleatorios).

2.5.9. Valores por defecto

Los valores por defecto para los parámetros del bioma están en el archivo **ecosystem.json**. No obstante he mantenido éste método durante el desarrollo, pero me gustaría estuviera gestionado en base de datos, en éste caso y dada su forma no estructurada, probablemente una NoSQL como MongoDB (aunque para la magnitud de los datos, tampoco es algo muy relevante, podría elegirse una SQL perfectamente).

Lo interesante es el sistema de sobreescritura de configuración dinámico que está presente; un usuario puede definir biomas de manera muy sencilla. Aquí puede verse un ejemplo de los tipos de parámetros que existen en este fichero.

2.5.9.1. Configuración de la simulación

Aún no hemos visto algunos sistemas o funcionalidades a las que algunos campos del siguiente fichero de configuración apuntan, pero aún así considero que mostrarlo ahora sirve para crear una perspectiva general, podemos verlo aquí: `simulation.yaml`.

2.5.9.2. Configuración de biomas/escenarios

Además, podemos crear nuevos tipos de organismos simplemente combinando componentes de diferentes maneras **sin tener que reestructurar todo el sistema**. Para eso disponemos del fichero `biome.yaml`. Al modificar este fichero, le estamos diciendo al simulador qué Bioma preparar para ésta simulación. Las secciones de flora y fauna, sobreescriben a las de por defecto como se ha explicado - esto es, a parte de sobreescribir, tenemos opción de no especificar nada para que se cargue lo existente en `ecosystem.json`. También podemos hacer una sobreescritura selectiva: indicar un solo componente con determinado valor para un parámetro y listo, el resto de parámetros y componentes que se carguen del fichero base.

La idea es que cualquiera, incluso usuarios sin conocimientos de programación puedan experimentar con diferentes configuraciones de biomas, simplemente editando estos archivos.

Para ello he implementado un sistema de prioridades durante el ensamblado de componentes en el momento de crear una entidad.

2.5.9.3. Prioridades

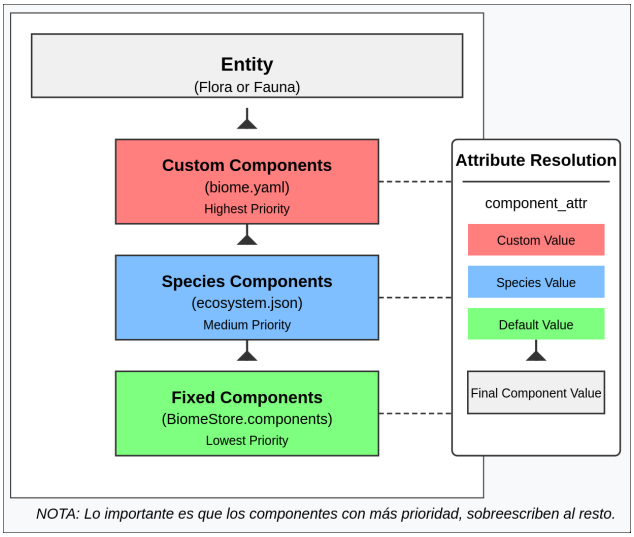


Figura 2.23: Prioridades para **atributos** de componentes

Utilizo configuración jerárquica de tres niveles para asignar los valores de los atributos o propiedades a las entidades:

1. **Componentes personalizados** (`biome.yaml`): Máxima prioridad.
2. **Componentes de Especie** (ecosystem.json): Prioridad media.
3. **Componentes Fijos**: Están también en ecosystem.json, pero tienen prioridad mínima; No obstante, **siempre** será asignado a la entidad de la especie. Haya sido definido en biome.yaml o no.

Esto da muchísima flexibilidad; el usuario del framework puede customizar de manera parcial un componente, hasta no tener valores por especie, o incluso redefinir de manera custom todos los componentes de todas las especies. Otro ejemplo, imaginemos que queremos definir **sólo** 2 atributos concretos para el componente de vitalidad - con especificarlos en el `biome.yaml` bastaría; el resto de los atributos serán ajustados por lo que diga la especie y pueden omitirse en el yaml, (si hay información al respecto, si no, los valores por defecto son los que serán establecidos).

Nota: Todo atributo de componente, ya sea los de defecto o sobreescritos por el usuario - se verán sometidos a una pequeña dosis de ruido aleatorio (distribución uniforme) durante la creación/instanciación. He hecho esto es para otorgar de cierta variabilidad, lo cual tendrá relevancia durante el apartado de algoritmos genéticos.