UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Generación Automática de Organismos Interrelacionados

Santiago Pacheco

Nicolás Ottonello

Ingeniería en Computación
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Agosto de 2019

UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Generación Automática de Organismos Interrelacionados

Santiago Pacheco

Nicolás Ottonello

Tesis de grado presentada en la Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de grado en Ingeniería en Computación.

Director:

Sergio Nesmachnow

Montevideo – Uruguay

Agosto de 2019

Pacheco, Santiago
Ottonello, Nicolás

Generación Automática de Organismos Interrelacionados / Santiago Pacheco, Nicolás Ottonello. - Montevideo: Universidad de la República, Facultad de Ingeniería, 2019.

XIV, 79 p.: il.; 29, 7cm.

Director:

Sergio Nesmachnow

Tesis de grado – Universidad de la República, Programa en Ingeniería en Computación, 2019.

Referencias bibliográficas: p. 75 – 78.

1. Algoritmos Evolutivos, 2. Neuroevolución, 3. NEAT, 4. Vida Artificial, 5. Videojuegos. I. Nesmachnow, Sergio, . II. Universidad de la República, Ingeniería en Computación. III. Título.

Montevideo – Uruguay
Agosto de 2019

Agradecimientos

Quisiéramos agradecer a nuestras familias, por apoyarnos durante todos estos años de estudio, y a la Facultad de Ingeniería, por darnos la oportunidad de estudiar lo que queríamos con una educación de primer nivel.

La imaginación de la naturaleza
es mucho, mucho mas grande
que la imaginación del hombre.

Richard Feynman

RESUMEN

En este proyecto de grado se presenta un algoritmo novedoso para la generación de múltiples especies de organismos en entornos virtuales, cuyos comportamientos se encuentren interrelacionados, diseñado para ser aplicable en la industria de los videojuegos. El algoritmo propuesto, abreviado AGIO por su sigla en ingles, se basa en el algoritmo de neuroevolución NEAT. AGIO contempla la exploración de diversas especies, el control a nivel de diseño del comportamiento de los organismos y la interrelación de los comportamientos entre especies. Junto a la descripción del algoritmo, se realizó una implementación en C++ para evaluar la propuesta. Los resultados experimentales demostraron que AGIO es capaz de generar decenas de especies de organismos distintas y cadenas de dependencias en el comportamiento de las especies encontradas. Además, el desempeño conseguido respecto a tiempo de ejecución y uso de memoria hace que AGIO sea aplicable en el desarrollo de videojuegos de alta calidad.

Palabras claves:
Algoritmos Evolutivos, Neuroevolución, NEAT, Vida Artificial, Videojuegos.

Lista de siglas

AEs Algoritmos Evolutivos 5

AGIO Automatic Generation of Interrelated Organisms 20

CE Cellular Encoding 12

ESP Enforced Sub Populations 12

IA Inteligencia Artificial 4

NEAT Neuroevolution of Augmenting Topologies 10

NSGA-II Non-dominated Sorting Genetic Algorithm II 15

SANE Symbiotic Adaptive Neuro-Evolution 12

Tabla de contenidos

Lista de figuras

Lista de tablas

Capítulo 1

Introducción

La *vida artificial* (en inglés A-Life) es un área multidisciplinaria que tiene como objetivo replicar los procesos y comportamientos naturales de la vida, comúnmente mediante simulaciones de software. Christopher G. Lanton, en una de las primeras publicaciones en el área, acuñó el termino "vida artificial" como el estudio de sistemas creados por el hombre que exhiben características de los sistemas naturales vivientes [16]. En el ámbito académico la investigación sobre el área de vida artificial es extensa, con artículos, libros, proyectos de maestría, entre otros. Algunos de estos trabajos se presentan en el capítulo 2.

Lograr simular los procesos y comportamientos naturales de la vida tiene no solo interés académico, sino que también interés económico por las aplicaciones prácticas de imitar a la vida en la industria del entretenimiento. Esta industria generó 1.9×10^9 USD en ventas en el 2016 a nivel mundial, con los videojuegos ocupando la mitad del mercado [36]. En los videojuegos se puede ver de forma directa la simulación de vida artificial, porque es común que exista una variedad de distintos individuos (personas, animales, plantas, etc.) que se encuentran en un entorno virtual e interactúan entre sí. Estos individuos pueden existir como parte de las mecánicas del juego, donde su comportamiento se rige por lo establecido en el diseño del juego, o pueden existir con el único propósito de mejorar la inmersión del jugador, haciendo que el mundo se sienta vivo.

La industria de videojuegos tuvo un incremento del 13.3 % en ganancias del 2017 al 2018 y como en toda industria en crecimiento, los requerimientos de los consumidores son cada vez más altos [26]. Steve Theodore, quien trabajó en reconocidos videojuegos como Half Life y Halo 3, respondió a la pregunta "¿Por qué los presupuestos de videojuegos se han disparado en los últimos años?" y

explica el crecimiento en el esfuerzo requerido para desarrollar un videojuego con el ejemplo del desarrollo de un personaje. Afirma que en el desarrollo de un juego en el año 2000, los personajes eran suficientemente simples para poder ser creados por un artista desde cero, pero que hoy en día crear un personaje puede involucrar fácilmente a una docena o más de personas [35].

El incremento en el esfuerzo requerido para desarrollar videojuegos incentivó a los desarrolladores a buscar maneras de poder producir más contenido más rápidamente. La automatización basada en métodos de aprendizaje automático se ha propuesto como una solución [3], donde empresas como Amazon, NVIDIA y Norah AI brindan herramientas basadas en métodos de aprendizaje automático enfocadas a videojuegos. Estas herramientas tienen diversos fines, como procesar datos de los jugadores para ajustar valores de jugabilidad, prevenir trampas, generar texturas y otros [19].

Sin embargo, existe un obstáculo a la automatización del contenido en videojuegos. Como los videojuegos tienen un componente artístico y de jugabilidad, los diseñadores deben tener cierto control de cómo van a funcionar las mecánicas del juego y de cómo se van a presentar visualmente. Mantener un control manual del diseño de los individuos y de su comportamiento se contrapone con generarlos de forma automática y puede ser difícil para un diseñador confiar en los resultados de un algoritmo.

Teniendo en cuenta la creciente dificultad para el desarrollo de videojuegos y la necesidad de mantener cierto control en su diseño, el objetivo de este proyecto de grado es presentar una herramienta capaz de generar organismos automáticamente, manteniendo un balance entre el control a nivel de diseño y la automatización. Más específicamente, el proyecto se centra en la generación de individuos que interactúan entre sí, dentro de un entorno definido por el desarrollador. El control sobre los resultados es brindado a través de lo que se denominan *componentes*. Mediante los componentes, el desarrollador define la forma en que los organismos interactúan con el entorno y cómo lo perciben.

A continuación se explica la estructura de lo que resta del documento. En el capítulo 2 se presenta el estado del arte sobre vida artificial y métodos de aprendizaje automático, junto con un análisis de las publicaciones previas más relevantes para este proyecto. En el capítulo 3 se presenta la solución propuesta para la generación automática de organismos interrelacionados, detallando cada uno de sus componentes y las razones detrás de las decisiones de diseño e implementación tomadas. El capítulo 4 se enfoca en describir los escenarios

y entornos sobre los que se realizó la evaluación experimental, y presenta la metodología del ajuste de parámetros. En el capítulo 5 se presentan los resultados obtenidos en el ajuste de parámetros y en los experimentos realizados, y se realiza una discusión de los resultados de cada experimento. Por último, el capítulo 6 da un cierre al trabajo, presentando conclusiones generales y proponiendo mejoras futuras.

Capítulo 2

Marco de trabajo

Este capítulo presenta el marco teórico con los conceptos más importantes para la comprensión del proyecto. Se describen en detalle las publicaciones consideradas cruciales y se comenta su relación con el proyecto.

2.1. Vida Artificial

Este proyecto se encuentra dentro del campo de investigación llamado *vida artificial*, campo que existe como tal desde 1986 [15] y fue nombrado de esta forma en 1988 por Christopher G. Lanton [16]. El concepto de vida artificial refiere a la utilización del modelado por computadoras para el estudio de la vida en general.

La metodología utilizada en vida artificial difiere en varias maneras de la utilizada en Inteligencia Artificial (IA) clásica [29]. La mayoría de los modelos de IA son sistemas seriales, especificados de forma top-down y basados en un controlador complejo y centralizado. El controlador se encarga de tomar las decisiones y conoce todos los aspectos del estado global del sistema. La decisiones tomadas por el controlador tienen el potencial de afectar directamente cualquiera de los aspectos del sistema.

Por otra parte, muchos de los sistemas de vida artificial que exhiben comportamientos complejos y autónomos son sistemas bottom-up, paralelos, basados en individuos relativamente simples que interactúan simultáneamente los unos con los otros. Estos sistemas se evalúan de manera iterativa, donde en cada paso todos los individuos deciden la acción a realizar y la ejecutan, resultando estas iteraciones en el comportamiento global observable del siste-

ma. Las decisiones tomadas por los individuos están basadas en información acerca de un entorno, local al agente, y solo afectan ese entorno. El comportamiento global del sistema es representado de forma indirecta y surge de las interacciones de las partes, entre ellas y con el entorno [2].

Una aspecto importante en la investigación de vida artificial es el modelado del comportamiento de los ecosistemas y la dinámica evolutiva de las poblaciones. La evolución por selección natural es una idea central en la biología y el concepto de selección natural ha influenciado en gran medida el desarrollo de los modelos evolutivos. Los Algoritmos Evolutivos (AEs) (descritos en la subsección 2.2.2) son los métodos más destacados y utilizados actualmente en vida artificial para modelar la dinámica evolutiva de las poblaciones [22]. Como ejemplos de la aplicación de AEs en investigaciones de vida artificial se puede encontrar la evolución de criaturas virtuales 3D [32], el reconocimiento de patrones en el sistema inmunológico [34] y la evolución de organismos digitales [28], entre otros.

Otro punto a destacar es que comúnmente en el área de vida artificial se trata de modelar todo el individuo, no un módulo específico como podría ser la visión, la toma de decisiones o el movimiento [29].

2.2. Métodos de aprendizaje automático

Si bien el campo de los métodos de aprendizaje automático es muy extenso, se considera relevante explicar brevemente algunos conceptos que son fundamentales para la comprensión del proyecto.

2.2.1. Aprendizaje por refuerzos

El aprendizaje por refuerzos aborda el procedimiento mediante el cual un individuo autónomo, que tiene información del entorno y puede interactúa con el entorno, puede aprender a realizar acciones para alcanzar un objetivo previamente definido. Este método de aprendizaje se basa en tener un componente externo a los agentes, denominado entrenador, que le otorga premios o sanciones a los agentes basados en su comportamiento.

Por ejemplo, si se tiene un individuo que juega un juego, el entrenador puede proveer un refuerzo positivo cuando el juego es ganado por el individuo, un refuerzo negativo cuando es perdido y no dar refuerzo ninguno en otro caso.

El objetivo del individuo es aprender de estos refuerzos a elegir la secuencia de acciones que producen la mayor recompensa acumulada [17].

2.2.2. Algoritmos Evolutivos

Los algoritmos evolutivos son una familia de algoritmos de aprendizaje automático que se inspiran en la teoría de la evolución natural y se encuentran dentro de los *algoritmos de población* por mantener un conjunto de posibles soluciones (llamado población). Son algoritmos probabilísticos, que no garantizan encontrar la solución óptima, pero si tratan de encontrar una solución aceptable, bajo algún criterio, en un tiempo que haga su utilización práctica [23].

Continuando las similitudes con la evolución natural, se utiliza la nomenclatura biológica para referirse a los distintos componentes del algoritmo. A las representaciones de las soluciones se les llama *genomas*, mientras que a la solución específica que un genoma expresa se le llama *fenotipo* o *individuo* [20].

La población es modificada iterativamente, donde las iteraciones son llamadas *generaciones*, para tratar de maximizar alguna métrica del problema que se denomina *función de fitness* [23].

Si bien existen diversos algoritmos específicos que se engloban dentro de la familia de algoritmos evolutivos, en general todos siguen las siguientes etapas en cada generación:

- Evaluación: Se evalúa la función de fitness en cada individuo. La complejidad de este paso va a depender mucho del problema a resolver, pudiendo ser desde una simple expresión matemática a una compleja simulación.
- Selección: Se eligen los individuos que se consideren adecuados para crear la nueva generación. En general se eligen individuos que tengan valores altos de fitness, porque se parte de la hipótesis de que generar un individuo nuevo a partir de otros que tienen un valor alto de fitness va a ayudar a mejorar ese valor. Sin embargo, es importante mantener también la diversidad de los individuos, a modo de poder muestrear la mayor parte posible del espacio de soluciones. Si existe poca diversidad en la población, es más probable que se converja a un máximo local de la función de fitness y no se pueda escapar de este máximo aunque sigan pasando las generaciones.

- Aplicación de operadores evolutivos: Una vez seleccionados los individuos candidatos, llamados comúnmente padres, se les aplica los llamados *operadores evolutivos*, que son funciones que toman uno o más individuos y devuelven uno o más individuos nuevos. Se dividen en dos clases, los operadores de cruzamiento, inspirados en la reproducción biológica, y los operadores de mutación, basados en el proceso natural del mismo nombre. Los operadores de cruzamiento se encargan del refinamiento de soluciones, tratando de generar nuevos individuos que tomen lo mejor de sus padres. Por otro lado, los operadores de mutación se encargan de explorar nuevas partes del espacio de soluciones, tratando de evitar que el algoritmo se estanque en máximos locales.
- Remplazo: Una vez generados los individuos de la nueva generación, llamados hijos, se decide en base a los hijos y los padres con que individuos formar la próxima generación y esos individuos pasan a ser la población. Aquí se encuentra una diferencia crucial con la evolución en la naturaleza, dado que en la gran mayoría de los algoritmos evolutivos el tamaño de la población es fijo.

Los puntos anteriores detallan como la población cambia a lo largo de las generaciones, pero no habla de cual es la población inicial. La población inicial más usada corresponde a un muestreo aleatorio uniforme del espacio de soluciones, de modo que se maximiza la diversidad y no se tienen sesgos iniciales en la búsqueda. Sin embargo, puede pasar que se sepa a priori que ciertas regiones del espacio de soluciones no aportan valor a la búsqueda y por lo tanto pueden ser evitadas en la generación inicial. Este tipo de ajustes al problema permiten mejorar los resultados obtenidos y/o el tiempo en el que se obtienen [23].

Como se menciona al inicio de la sub-sección, estos algoritmos no garantizan encontrar la solución óptima, por lo que se necesita otro criterio para terminar la ejecución. Los dos criterios más usados son tener una cantidad máxima de generaciones o un mínimo de mejora de las soluciones. El primer criterio ayuda a tener un determinismo sobre el tiempo de ejecución, mientras que el segundo evita el continuar realizando generaciones cuando la población no está mejorando significativamente [23].

Por último, los algoritmos evolutivos tienden a tener diversos parámetros que controlan su ejecución, por lo que es necesario realizar una evaluación

en distintas instancias del problema para encontrar cuales valores de estos parámetros brindan los mejores resultados. Los parámetros más comúnmente ajustados son el tamaño de la población, la probabilidad de mutación y la probabilidad de cruzamiento, aunque se pueden ajustar tantos como el algoritmo requiera [23].

2.2.3. Redes Neuronales Artificiales

Los métodos de aprendizaje automático basados en redes neuronales artificiales, comúnmente llamadas redes neuronales, permiten aproximar funciones reales, discretas y vectoriales de manera robusta y efectiva [24].

Las redes neuronales están constituidas por un conjunto de unidades simples, llamadas neuronas, que se encuentran interconectadas. Esta organización se inspira en el cerebro de los animales, que también se compone de neuronas interconectadas [24].

Las neuronas en una red neuronal pueden ser clasificadas en tres tipos: neuronas de entrada, neuronas de salida y neuronas ocultas. La figura 2.1 muestra un ejemplo de la estructura de una red neuronal. En una red neuronal que aproxima a cierta función f, las neuronas de entrada reciben los valores v a procesar y las neuronas de salida retornan la aproximación calculada por la red neuronal al resultado de $f(v)$. Las neuronas ocultas se encuentran entre las neuronas de entrada y las neuronas de salida. Las neuronas se agrupan en capas, la capa 0 esta compuesta por las neuronas de entrada y la capa n por las neuronas de salida. Las neuronas en la capa i reciben los datos de entrada de las neuronas en la capa $i-1$, realizan alguna operación simple sobre esos datos y le entregan su salida a las neuronas en la capa $i+1$. Los datos de entrada y salida son valores numéricos y las conexiones entre neuronas tienen un peso que multiplica el valor que se transfiere por esa conexión. [24].

En la mayoría de algoritmos que utilizan redes neuronales, la cantidad de neuronas y la topología de la red (las conexiones entre las neuronas) se diseña manualmente y no varía en el proceso de aprendizaje, ajustándose en ese proceso solo los pesos asociados a las conexiones. Para realizar el ajuste de los pesos el algoritmo más utilizado se llama Backpropagation y se basa en el método de optimización conocido como descenso por gradiente [24]. Sin embargo, existe un enfoque alternativo para la creación y el entrenamiento de redes neuronales, llamado *neuroevolución*, que utiliza algoritmos evolutivos para no

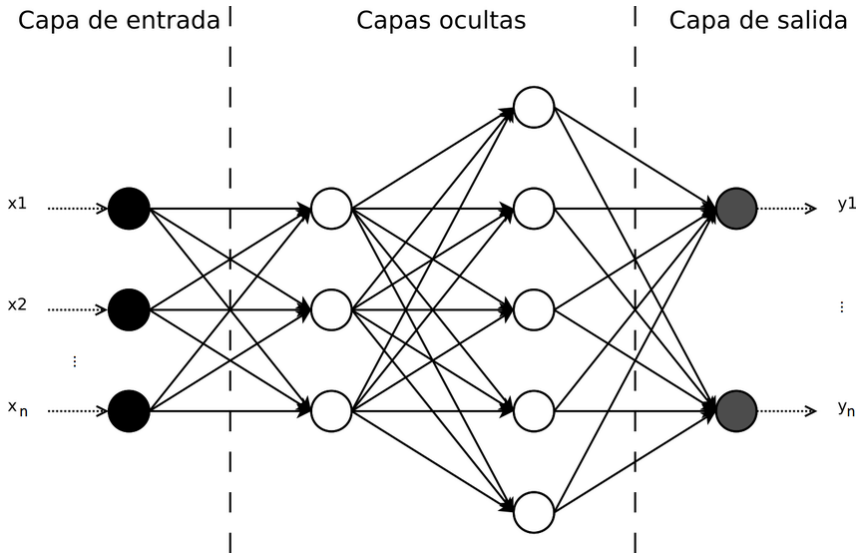


Figura 2.1: Ejemplo de una estructura de una red neuronal artificial.

solo ajustar los pesos de la red, sino que también la cantidad de neuronas y la topología. Como se utilizan algoritmos evolutivos, no se puede garantizar que se encuentre la red que mejor aproxima a la función deseada, pero la neuroevolución pude manejar topologías arbitrarias e incluso generarlas. Poder trabajar con topologías arbitrarias permite tener construcciones más complejas a nivel topológico, como por ejemplo conexiones recurrentes que actúen de memoria. Además, que el algoritmo pueda generar la topología evita la etapa de diseño de la red que dependiendo de la magnitud del problema a resolver puede ser muy compleja. El algoritmo NEAT, detallado en la subsección 2.3.2, es el algoritmo de neuroevolución que se utilizó en este proyecto para generar las redes neuronales de los individuos [33].

2.3. Estado del arte

Esta sección presenta, en orden cronológico, los distintos trabajos previos en las áreas de neuroevolución y vida artificial que se consideran cruciales para este proyecto.

2.3.1. Evolving Virtual Creatures

Karl Sims, en 1994, desarrolló un sistema capaz de generar criaturas virtuales, tridimensionales y autónomas sin requerir una exhaustiva especificación

del usuario, esfuerzo de diseño o conocimiento de los detalles del algoritmo. Las criaturas virtuales son capaces de moverse en la simulación de un mundo tridimensional. La morfología y el sistema neuronal para controlar los músculos de las criaturas son generados automáticamente utilizando algoritmos evolutivos [32].

Sims optó por definir las criaturas como una jerarquía de partes rígidas articuladas y el genoma como un grafo dirigido de nodos y conexiones, donde cada nodo contiene información que describe a una de las partes rígidas. Entre la información descrita se encuentra el tipo de unión, el cual determina restricciones sobre la movilidad entre las partes unidas, y también sensores, los cuales obtienen información del estado de la parte que lo contiene o del mundo.

Para determinar el comportamiento de la criatura se utilizaron varias redes neuronales, una para cada nodo del grafo descrito anteriormente, y además se utilizo una red neuronal para coordinar a todas las demás redes neuronales. Las redes aceptan como entrada los valores de los sensores y retornan como salida fuerzas o torques que son aplicadas a las articulaciones. Sims propuso que las neuronas pudieran aplicar diferentes funciones a sus entradas en vez de solamente una suma ponderada, como una manera de facilitar la evolución de comportamientos complejos y más cercanos a los naturales.

En la evaluación experimental se realizó una simulación dinámica, calculando el movimiento resultante de la interacción de la criatura con el mundo virtual. De la evaluación se obtuvieron, dependiendo de la función de fitness utilizada, organismos capaces de nadar, correr, saltar y de seguir una fuente de luz. Estos organismos fueron capaces de realizar la tarea de una forma eficiente, utilizando diferentes estructuras y comportamientos que emergieron del proceso evolutivo.

El artículo de Sims, pionero en el área, proporciono evidencia de que la creación de organismos de forma automática utilizando algoritmos evolutivos es factible. Los resultados obtenidos por Sims demostraron que la evolución de la morfología del individuo puede ser realizada simultáneamente a la evolución del comportamiento y que el uso de algoritmos evolutivos y redes neuronales para la simulación de vida artificial da buenos resultados .

2.3.2. Evolving Neural Networks through Augmenting Topologies

Kenneth Stanley y Risto Miikkulainen presentaron en 2002 un algoritmo de neuroevolución denominado Neuroevolution of Augmenting Topologies (NEAT). En NEAT, se evolucionan los pesos y la topología de la red utilizando los principios biológicos de aumento de complejidad y especiación [12].

Evolucionar la topología de una red requiere encontrar una manera de cruzar dos topologías distintas, pero preservando en cierto grado los comportamientos que las dos describen, un problema que no es trivial de resolver. NEAT propone utilizar marcadores históricos en los genes para poder realizar de manera eficiente un emparejamiento de genes entre padres, a modo de encontrar cuáles partes de cada red tiene sentido cruzar. Dos genes con el mismo origen histórico representan la misma estructura (potencialmente con distintos pesos) porque derivan del mismo gen original, por lo que pueden ser cruzados. Los marcadores históricos le permiten a NEAT realizar de manera eficiente un cruzamiento topológico que preserva las características evolucionadas.

La exploración del espacio de soluciones es sesgada a sub-espacios de dimensión mínima, al comenzar con una población sin nodos ocultos. Nuevas estructuras son introducidas incrementalmente a medida que las mutaciones ocurren y solo sobreviven aquellas que resultan útiles a través de la evaluación del fitness.

NEAT utiliza un método inspirado en la biología, la especiación, para proteger nuevas innovaciones en las etapas tempranas de su evolución. En el mundo biológico, los organismos evolucionan en distintas especies que conforman diferentes nichos o roles en el entorno. En un algoritmo de neuroevolución, cuando una nueva conexión o neurona es agregada a la red, es probable que el fitness de la red decaiga. Sin ninguna medida para proteger la innovación, el individuo puede ser eliminado de la población antes de que la red evolucione en un comportamiento útil. NEAT previene la extinción prematura de estas innovaciones simulando la especiación biológica. Al comenzar el algoritmo no hay especies pre-existentes, por lo que se crea la primera especie a partir de un primer genoma. A medida de que las innovaciones suceden, nuevas especies son creadas y se almacenan en una lista. Los individuos son comparados contra un representante de cada especie para calcular una distancia de compatibilidad. Si la distancia es lo suficientemente pequeña, de acuerdo con los límites definidos,

el individuo es agregado a la especie y comparte el fitness del nicho de esta. Si el individuo no es compatible con ninguna especie, se convierte en el primer miembro de una nueva especie, la cual es agregada a la lista.

En el artículo de NEAT se propusieron dos escenarios para evaluar experimentalmente el método desarrollado. El primer escenario consistió en computar una función XOR, ya que para poder hacerlo la red necesita como mínimo una neurona oculta. De esta forma se comprobó la capacidad de NEAT de evolucionar la estructura de las redes, tomando en promedio 32 generaciones para evolucionar una red que calcule correctamente la función XOR. Además, las soluciones encontradas en promedio tuvieron 2.35 neuronas ocultas, mostrando que NEAT mantiene las topologías de las redes neuronales generadas pequeñas. El segundo escenario propuesto se trata del Double Pole Balancing Problem, diagramado en la figura 2.2, el cual fue utilizado para comparar los resultados obtenidos por NEAT con los resultados obtenidos utilizando otros métodos como la programación evolutiva, la neuroevolución convencional, Symbiotic Adaptive Neuro-Evolution (SANE) [25], Cellular Encoding (CE) [7] y Enforced Sub Populations (ESP) [6]. Se concluyó que el método NEAT es más eficiente que los demás métodos porque las redes evolucionadas obtienen mejor fitness, requiere menos generaciones y las redes generadas son más pequeñas.

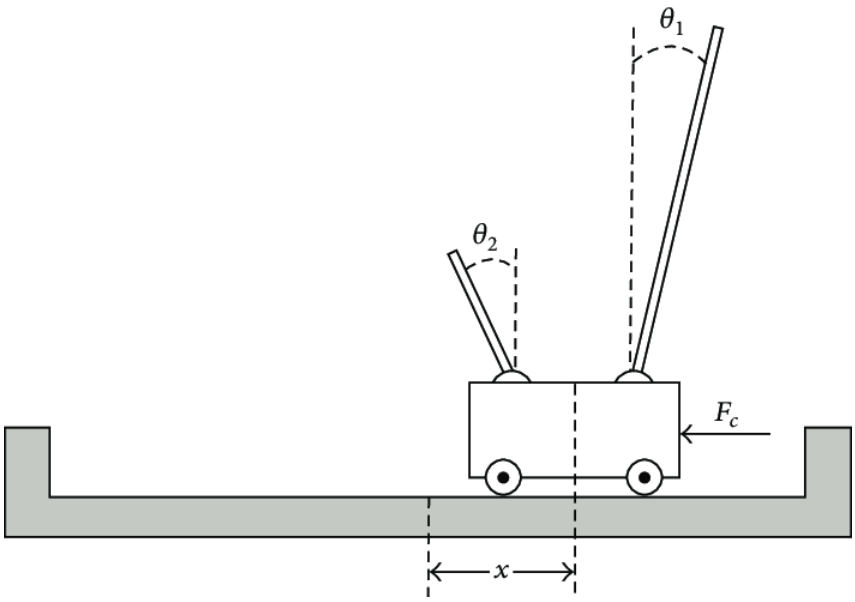


Figura 2.2: El problema Double Pole Balancing Problem consiste en mantener balanceadas dos varas de distinta medida que se encuentran montadas con una articulación móvil en un carro, usando como entrada los ángulos de las varas y la posición del carro.

En base a los resultados obtenidos y a la cantidad de trabajos precedentes a este proyecto que utilizaron NEAT con éxito, se puede concluir que es un método poderoso para la tarea de evolucionar redes neuronales artificiales.

2.3.3. Evolutionary Development of Robotic Organisms (ERO)

En su tesis de maestría del 2007, Peter Krčah realizó un estudio sobre los métodos evolutivos aplicados a la generación de criaturas virtuales y presentó luego una mejora a los operadores evolutivos utilizados por Sims en el modelo propuesto en "Evolving Virtual Creatures" [14].

El punto que trató de mejorar Krčah es el operador de cruzamiento, el cual identificó que es demasiado destructivo en el modelo de Sims. Krčah propuso utilizar un sistema derivado de NEAT, el cual denomina *Hierarchical NEAT*, para realizar el cruzamiento de la morfología de los individuos y de sus redes neuronales. Al igual que NEAT, Hierarchical NEAT utiliza marcadores históricos para realizar de manera eficiente el cruzamiento entre distintas topologías.

La evaluación experimental mostró que el método propuesto, en las tareas de seguir una luz, caminar, saltar y nadar evoluciono al mismo nivel de fitness en 1.67, 2.47, 1.52 y 2.15 veces menos generaciones, respectivamente, que el método de Sims. Se realizaron adicionalmente pruebas de ablación, donde se confirmó que cada componente Hierarchical NEAT es necesario. Dentro de las pruebas de ablación se estudió si la propuesta de Sims de utilizar distintas funciones en las neuronas realmente mejoraba los resultados. Los resultados mostraron que no habían diferencias significativas al utilizar diversas funciones en las neuronas o usando la misma función. Por último, las pruebas de ablación mostraron que para algunas tareas no es necesario un sistema que realice una coordinación entre las redes que controlan las distintas partes del individuo.

La mayor contribución de este trabajo fue actualizar el método propuesto por Sims en 1994, presentando un algoritmo que es más simple y permite obtener mejores resultados.

2.3.4. Abandoning Objectives: Evolution through the Search for Novelty Alone

Joel Lehman y Kenneth Stanley propusieron en un artículo de 2011 abandonar el uso de una función de fitness como medida de que tan cerca está una solución de alcanzar un objetivo dado, y buscar en cambio solamente comportamientos novedosos, ignorando el objetivo. Utilizar una función de fitness puede ser adecuado cuando el problema es simple y una función de fitness ideal es fácil de definir, pero a medida que los problemas se hacen más complejos, también se complejiza la definición de las funciones de fitness necesarias para saber que tan buena es una solución. La propuesta de los autores, *Novelty Search*, se basa en reducir la posibilidad de introducir sesgos que impidan al algoritmo alcanzar el objetivo [8].

Lehman y Stanley fundamentan que en el espacio de comportamientos distintos genomas pueden colapsar al mismo comportamiento, por lo que si bien el espacio de genomas puede ser muy grande o incluso infinito, el espacio de comportamientos no necesariamente lo es. La propuesta se separa de lo que a priori sería una simple búsqueda aleatoria al recorrer el espacio de genomas de menor a mayor complejidad. Además, se guarda un registro de los individuos que fueron novedosos a lo largo de la ejecución del algoritmo, para evitar repetir genomas ya vistos.

La evaluación experimental utilizó dos problemas distintos, uno que involucra individuos recorriendo un laberinto y otro que consiste en la tarea de controlar un bípedo para que recorra la mayor distancia posible. En los dos casos el método Novelty Search generó mejores resultados que el método NEAT clásico.

Los resultados experimentales mostraron que en algunas situaciones, ignorar el objetivo produce mejores resultados que intentar alcanzarlo mediante el uso de una función de fitness. El enfoque propuesto es una nueva manera de utilizar los algoritmos evolutivos, tomando un camino más abierto y focalizado en la exploración.

2.3.5. Evolving a Diversity of Creatures through Novelty Search and Local Competition

Joel Lehman y Kenneth Stanley propusieron en 2011 utilizar Novelty Search para generar una variedad de distintos organismos, pero a su vez realizar una competencia local para garantizar que los organizmos sean de alta calidad [9]. Los autores basan su trabajo en el trabajo de Krčah, "Evolutionary Development of Robotic Organisms".

En general, los algoritmos evolutivos buscan converger a una solución lo más cercana posible a la solución óptima. La convergencia a una solución causa que al aplicarlos a la generación de organismos virtuales, cada ejecución tienda a generar un solo individuo, requiriéndose múltiples ejecuciones para generar una diversidad de organismos. Los autores argumentaron que obtener el organismo óptimo no es tan interesante como generar una diversidad de organismos, todos adaptados de maneras distintas al objetivo buscado, aunque no sean necesariamente los mejores. La propuesta consiste en utilizar un algoritmo evolutivo múlti-objetivo, donde uno de los objetivos sea la novedad, como fue planteado en la publicación de los mismos autores sobre Novelty Search, pero en vez de considerar la novedad en comportamientos, consideran la novedad en morfologías.

Lehman y Stanley reconocen además que en la naturaleza los distintos organismos no compiten todos entre ellos, sino que compiten dentro de su nicho. Por ejemplo, una bacteria no compite directamente con un oso. Los autores introducen así el concepto de competición local, donde los individuos solo compiten para reproducirse contra individuos en su mismo nicho. En la competencia local, los individuos reciben un puntaje en relación a cuántos de sus vecinos más cercanos en el espacio morfológico superan.

La novedad en morfología y la competitividad local son los dos objetivos que se consideran en el algoritmo. Se utiliza una variante de Non-dominated Sorting Genetic Algorithm II (NSGA-II) para realizar la evolución, donde la variante es eliminar el mecanismo que tiene NSGA-II para mantener la diversidad en el frente de Pareto [11]. La razón para modificar NSGA-II es que tanto la novedad como la competitividad local son medidas relativas, dos individuos con la misma novedad o con la misma competitividad local pueden ser muy diferentes morfológicamente o tener comportamientos muy distintos.

En la evaluación experimental realizada, cuatro funciones objetivo fueron

probadas: solo fitness global, solo novedad, novedad con competencia global y novedad con competencia local. A partir de los resultados se concluyó que los individuos con mayor fitness se encuentran al utilizar novedad con competencia global, pero la cantidad de morfologías exploradas es menor que al utilizar solo la novedad como función objetivo. Al utilizar solo novedad como función objetivo se explora la mayor cantidad del espacio de morfologías, pero no todas las morfologías encontradas son capaces de realizar el objetivo de moverse. En cambio, al utilizar novedad con competencia local, se explora un espacio de morfologías similar al que se explora buscando solo novedad y los individuos obtenidos son capaces de realizar el objetivo de moverse tan bien como lo hacen los individuos encontrados utilizando como función objetivo solo el fitness global. Se concluye que efectivamente la búsqueda de novedad con competencia local explora el espacio de soluciones y a su vez encuentra individuos con buen desempeño.

El trabajo de Lehman y Stanley mostró que se puede crear y mantener una amplia variedad de organismos funcionales a través de la competencia local y la búsqueda de novedad.

2.3.6. Simulation of Animal Behavior Using Neural Networks

Sebastian Echegaray y Wenbin Luo (2006) desarrollaron un entorno para la simulación del comportamiento animal, concluyendo que es posible simular comportamientos simples de animales mediante el uso de algoritmos evolutivos [31].

La simulación propuesta consistió de un modelo presa-depredador-comida. Cada individuo tenía su propia red neuronal para controlar su comportamiento, que fue evolucionada utilizando NEAT. El fitness de los individuos fue evaluado en tiempo real durante la simulación y luego de una cantidad de generaciones determinada, la población entera fue regenerada, usando los individuos con mayor fitness como padres de la nueva generación. Cuando un individuo moría en la simulación, aparecía nuevamente en una posición aleatoria del mundo.

Los individuos reciben entradas de su entorno a través de sensores de distintos tipos: sensores que detectan muros, sensores para detectar depredadores, sensores para detectar presas y sensores para detectar comida.

Para la evaluación experimental fue creado un mundo donde el depredador,

la presa y la comida coexisten. El objetivo de los individuos es sobrevivir lo suficiente para poder reproducirse, por lo que deben competir entre sí. El fitness fue evaluado de la siguiente forma: cada vez que un depredador come una presa, su fitness aumenta. Cada vez que la presa come alimento, su fitness aumenta. Cada vez que la presa es comida, su fitness decrece. Se observó que a medida que las generaciones avanzaban, los individuos comenzaron a mostrar comportamientos inteligentes.

El artículo de Echegaray y Luo presentó un enfoque útil para la construcción de la función de fitness de los individuos en la simulación . También se mostró una solución al problema de qué hacer cuando un individuo muere y cada cuánto regenerar la población.

2.4. Análisis y relación con el proyecto de grado

Los trabajos reseñados muestran que es posible evolucionar diversos individuos simultáneamente, junto a una red neuronal para cada individuo que define sus comportamientos, formando la base para este proyecto.

En Evolving Virtual Creatures se presenta un modelo de morfología que permite individuos con formas arbitrarias. Si bien eso es muy interesante desde el punto de vista académico, utilizar estos métodos en un videojuego es complejo, porque no existe nada que un desarrollador pueda hacer para tener control sobre que individuos se forman. Al menos que el juego sea diseñado en base al algoritmo, como en *Evocommander* [5], la falta de control sobre los individuos formados hace que los algoritmos presentados no sean directamente aplicables a un videojuego cualquiera. Un algoritmo que sea aplicable a videojuegos en general tiene que brindar control al desarrollador a nivel de diseño.

Para lograr tener control a nivel de diseño sobre los individuos formados, este proyecto propone un modelo más restrictivo de morfología, donde el desarrollador es el responsable de definir qué acciones pueden realizar los individuos, qué información tienen los individuos del entorno y como se pueden combinar las acciones y los sensores, que son lo que provee la información del entorno a los individuos.

Respecto al mecanismo de decisión de los organismos, se optó por utilizar una red neuronal en cada individuo, evolucionadas con NEAT. NEAT es un

método probado, que ha obtenido buenos resultados y logra que las redes resultantes sean pequeñas, lo que las hace ocupar poco espacio en memoria y ser rápidas de evaluar. El ocupar poco espacio en memoria y la rápida evaluación son dos propiedades deseables teniendo en cuenta que uno de los principales objetivos de este proyecto de grado es la realización de una herramienta aplicable en videojuegos. Como los videojuegos son aplicaciones interactivas, el tiempo de ejecución y uso de memoria de los distintos algoritmos utilizados es importante, porque tienen que mantener cierta cantidad de cuadros por segundo y tiempo de respuesta para que el videojuego sea fluido.

En la tabla 2.1 se resumen los puntos fundamentales y los principales aportes de los trabajos previos reseñados.

Año	Título	Autores	Puntos clave
1994	Evolving virtual creatures	Karl Sims	Evolucionar morfología y control para organismos en entornos de vida artificial.
2002	Evolving Neural Networks through Augmenting Topologies	Kenneth Stanley y Risto Miikkulainen	Evolucionar topología y pesos de redes neuronales de manera incremental.
2006	Simulation of Animal Behavior Using Neural Networks	Sebastian Echegaray y Wenbin Luo	Simular comportamientos básicos de animales utilizando NEAT.
2007	Evolutionary Development of Robotic Organisms	Peter Krčah	Extender el trabajo de Sims de 1994 con los conceptos de NEAT para el cruzamiento de individuos.
2011	Abandoning Objectives: Evolution through the Search for Novelty Alone	Joel Lehman y Kenneth Stanley	Dejar de lado los objetivos y buscar novedad como una manera de encontrar soluciones nuevas y potencialmente mejores.
2011	Evolving a Diversity of Creatures through Novelty Search and Local Competition	Joel Lehman y Kenneth Stanley	Aplicar Novelty Search al trabajo de Krčah de 2007.

Tabla 2.1: Puntos clave de los trabajos previos en el área de neuroevolución y vida artificial considerados cruciales para este proyecto.

Capítulo 3

Generación automática de organismos interrelacionados

En este capitulo se presenta los detalles del sistema propuesto, las decisiones de diseño tomadas y los pseudocódigos de los algoritmos desarrollados.

3.1. Composición de los individuos

La solución propuesta, Automatic Generation of Interrelated Organisms (AGIO), genera especies de individuos y para cada individuo genera un comportamiento. Una *especie* es un conjunto de individuos que pueden cruzarse entre sí en el proceso evolutivo y que comparten una misma morfología.

Con el fin de brindar al desarrollador control sobre los resultados a obtener, la morfología de un individuo está formada por un conjunto de componentes, como se muestra en la figura 3.1. Los componentes son definidos por el desarrollador y son agrupados. Cada grupo tiene asociado dos números enteros que representan la cantidad mínima y la cantidad máxima de componentes del grupo que un individuo debe poseer. Cada componente determina un conjunto de *sensores* con los que el individuo percibe su entorno y un conjunto de *acciones* que el individuo puede realizar para alterar el entorno. Al formar un individuo, se toma de cada grupo una cierta cantidad de componentes comprendida entre el mínimo y el máximo indicados. La formación de la morfología de un individuo se detalla en el algoritmo 1.

A modo de ejemplo, en la generación de una fauna animal, una definición posible para un grupo de componentes sería “tren inferior”. Este grupo podría

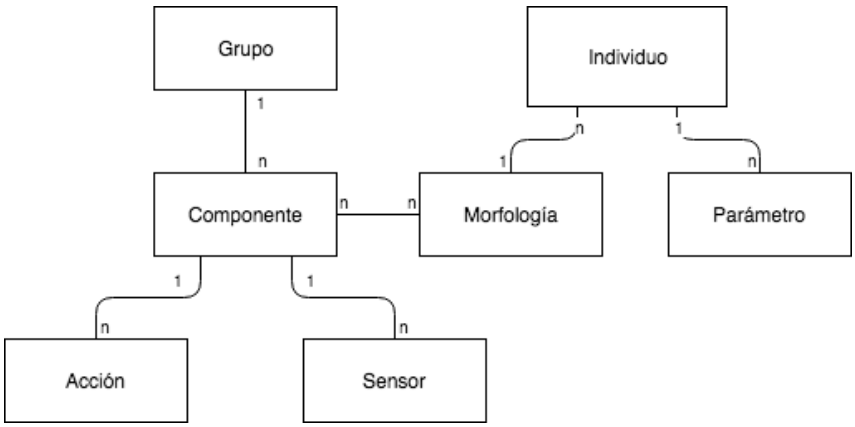


Figura 3.1: Composición de un individuo.

estar compuesto por los componentes "pata" y "aleta caudal", brindando al individuo las acciones de caminar y nadar, respectivamente.

Además de los componentes, existen valores denominados *parámetros*. Los parámetros se encuentran asociados a los distintos individuos y existen con el objetivo de dar mayor poder de expresión al desarrollador . Por ejemplo, el desarrollador puede definir un parámetro como "fuerza" y gracias al proceso evolutivo se podrán encontrar organismos con distintos valores de fuerza y comportamientos acordes a ese valor. AGIO trata a los parámetros como valores numéricos continuos en un intervalo cerrado definido por el desarrollador. Los parámetros pueden ser requeridos (deben estar presentes en todos los individuos) o no. Si un parámetro no es requerido, estará presente en el individuo

Algoritmo 1: Formación de la morfología de un individuo.

```
función formar_morfología_aleatoria():
    morfología = AGIO:Morfología()
    para cada grupo en grupos_de_componentes hacer
        mín = grupo.cantidad_mínima
        máx = grupo.cantidad_máxima
        cantidad_componentes = entero_aleatorio(mín, máx)
        componentes_elegidos =
          grupo.componentes_aleatorios(cantidad_componentes)
        morfología.agregar(componentes_elegidos)
    fin
    devolver morfología
fin
```

con probabilidad 0,5. A lo largo del proceso evolutivo, se evolucionan también los parámetros, tomando de NEAT la idea de utilizar marcadores históricos para no realizar cambios destructivos.

3.2. Comportamiento de los individuos

El modelo propuesto para definir el comportamiento de los individuos consiste en que cada individuo cuente con una red neuronal. Dado un individuo, su red neuronal toma como entrada los valores de los sensores que el individuo posee y devuelve un vector V de tamaño n de valores en el rango $[0, 1]$, siendo n el número de distintas acciones que el individuo puede realizar. Formalmente, un sensor es una función $Estado \times Población \times Individuo \times Mundo \to \mathbb{R}$. Cada valor $v_i \in V$ corresponde a la probabilidad, sin normalizar, de realizar la acción i.

Algo importante a tener en cuenta es que el modelo de decisión es discreto. Por ejemplo, si se tiene una acción A que indica que rota al individuo θ grados, el individuo solo podrá rotar $m \cdot \theta, m \in \mathbb{N}$ grados. Rotar $m \cdot \theta$ grados se consigue al realizar la acción A m veces.

3.3. Especiación

El concepto de nichos es utilizado en el ámbito de los algoritmos evolutivos como una manera de preservar la diversidad, dividiendo la población en distintos grupos [18]. Por otra parte, en biología las especies son grupos de poblaciones naturales que se reproducen entre ellas y están reproductiva (genéticamente) separadas de otros grupos [21].

En el contexto de este proyecto se define *especie* como una combinación específica de componentes (morfología) y el modeo de evolución no permite el cruzamiento entre individuos de especies distintas. El hecho de no permitir cruzar individuos que no pertenezcan a la misma especie se debe a que si dos organismos tienen sensores o acciones distintos, las redes neuronales que definen sus comportamientos tendrán distintos nodos de entrada y/o salida. Se tendrían entonces neuronas con semánticas distintas, por lo que no es posible cruzar las redes de una forma que tenga sentido a lo que las redes están aproximando funciones distintas. Además, si dos individuos tienen acciones y/o

sensores distintos, sus rangos de fitness pueden ser completamente distintos, por lo a priori sus fitness no son comparables (esto es similar a lo expuesto en [10]).

En base a lo planteado, cada especie se considera como una sub-población independiente desde el punto de vista evolutivo y se evoluciona por separado del resto. Dada una especie, se crea una población de NEAT con sus individuos y la población de NEAT se utiliza para evolucionar las redes neuronales asociadas al comportamiento de los individuos. Además, la población de NEAT también se utiliza para evolucionar los parámetros de los individuos. La evolución de los parámetros es realizada extendiendo la implementación de los organizamos de NEAT.

El algoritmo 2 presenta el método utilizado para inicializar las especies. Primero, por cada especie que va a ser creada se crea una morfología. Luego, para cada morfología se calcula cuántos sensores y acciones posee. Conociendo la cantidad de sensores y acciones se crea un genoma de NEAT que especifica cuantos nodos de entrada, ocultos y de salidas tendrán las redes neuronales producidas y los enlaces que existirán entre sus nodos. Para finalizar, se crean los individuos de la especie asignandole a cada uno la morfología y la red neuronal de la población de NEAT. Todos los individuos comparten la misma morfología, pero cada uno cuenta con una red neuronal distinta a la del resto.

Algoritmo 2: Inicialización de las especies.

```
cantidad_especies = tamaño_población / individuos_por_especie
para i=1 a cantidad_especies hacer
    morfología = formar_morfología_aleatoria()
    cantidad_sensores = morfología.cantidad_sensores()
    cantidad_acciones = morfología.cantidad_acciones()
    genoma = NEAT:Genoma(cantidad_sensores, cantidad_acciones)
    población_neat = NEAT:Población(genoma, individuos_por_especie)
    para j=1 a individuos_por_especie hacer
        red_neuronal = población_neat[j]
        individuo = AGIO:Individuo(morfología, red_neuronal)
        individuos.agregar(individuo)
    fin
    nueva_especie = AGIO:Especie(individuos, población_neat)
    especies.agregar(nueva_especie)
fin
```

3.4. Exploración de morfologías

Un componente crucial del sistema desarrollado es la exploración de las posibles combinaciones de componentes que forman la morfología de un individuo. Idealmente, la población consideraría todas las combinaciones posibles, pero en la practica considerar todas las combinaciones no es viable porque requeriría un tiempo de computo demasiado alto para los objetivos del sistema.

Para demostrar que no es viable considerar todas las posibles combinaciones de componentes se plantea el siguiente ejemplo. Sea n la cantidad de grupos de componentes definidos por el desarrollador y k_i el cardinal del conjunto de componentes correspondientes al grupo i. Sean a_i y b_i la cantidad mínima y máxima de componentes del grupo i a usar, respectivamente. La cantidad C de posibles morfologías está dada por la ecuación (3.1).

$$\begin{aligned} C &= \prod_{i=1}^{n} \binom{k_i}{a_i} + \binom{k_i}{a_i + 1} + \cdots + \binom{k_i}{b_i} \\ &= \prod_{i=1}^{n} \sum_{j=a_i}^{b_i} \binom{k_i}{j} \end{aligned} \tag{3.1}$$

Para ilustrar lo rápido que crece C al aumentar n y/o los distintos k_i, considérese el caso donde todos los grupos tienen la misma cantidad k de componentes ($k_i = k_j \ \forall \ i, j$), $a_i = 1$ y $b_i = 2 \ \forall i$. En este caso la función C solo depende de dos variables, k y n, por lo que la ecuación (3.1) se reduce a la ecuación (3.2).

$$C = \left(k + \frac{k(k-1)}{2} \right)^n \tag{3.2}$$

La tabla 3.1 muestra el valor de C para distintos valores de k y n en el caso anterior.

Como se puede observar, incluso para el caso pequeño con $k = 3$ y $n = 2$, es posible formar 36 especies distintas. Teniendo en cuenta que el tamaño de población utilizado en la investigación de NEAT [12] es de 150 individuos y que el comportamiento de cada especie es evolucionado utilizando una población de NEAT, se necesitaría una población de $36 \times 150 = 5400$ individuos para poder evolucionar las 36 especies simultáneamente. Utilizar una población de tal magnitud requeriría un tiempo de computo y un uso de memoria más

n \ k	2	3	4	5
1	3	6	10	15
2	9	36	100	225
3	27	216	1000	3375
4	81	1296	10000	50625

Tabla 3.1: Valores de C para distintos valores de k y n.

grande que el disponible en un videojuego, como se detalla en la sección 4.3.5.

Una aparente solución al problema de tener demasiados individuos en la población sería reducir la cantidad de individuos por especie. Si bien esto soluciona el problema cuando se tiene 36 especies, el problema vuelve a surgir al aumentar n y/o k. La solución es no evolucionar todas las especies posibles de forma simultánea, sino evolucionar un subconjunto elegido de forma aleatoria e ir remplazando algunas especies por otras para evolucionar especies nuevas. El criterio para decidir si una especie es remplazada se basa en su progreso en cuanto a el fitness de sus individuos.

Se dice que una especie tiene edad t cuando ha pasado por t generaciones. El progreso p_t de una especie con edad t queda definido por la ecuación (3.3). En la ecuación (3.4), $\hat{f}_t$ es el promedio de fitness en la generación t de los mejores k individuos de la especie. La formulación de α_t toma en cuenta la edad de la especie y permite que valores iniciales de $\hat{f}_t$ afecten más rápido la métrica de progreso. El parámetro $\lambda \in [0, 1]$ utilizado en la ecuación (3.5) amortigua las variaciones aleatorias de $\hat{f}_t$ y p_t, es decir, controla que tanta importancia se le da a los nuevos valores con respecto a los valores anteriores.

$$p_0 = 0$$
$$p_t = (1 - \alpha_t)p_{t-1} + \alpha_t \frac{s_t - s_{t-1}}{s_{t-1}} \tag{3.3}$$

$$s_0 = \hat{f}_0$$
$$s_t = (1 - \alpha_t)s_{t-1} + \alpha_t \hat{f}_t \tag{3.4}$$

$$\alpha_t = \frac{\lambda t + 1}{t + 1} \tag{3.5}$$

En la figura 3.3 se puede observar un ejemplo de la evolución en el tiempo de p_t para distintos valores de λ. Los valores de $\hat{f}_t$ utilizados para el ejemplo son los mostrados en la figura 3.2.

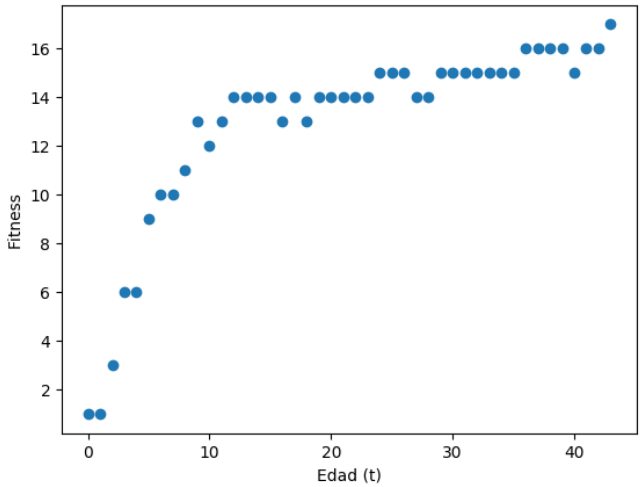
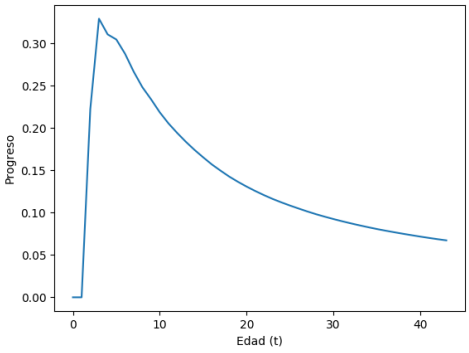
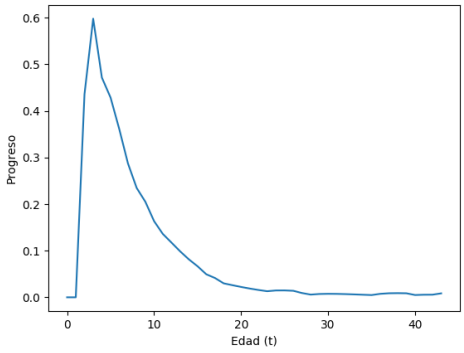


Figura 3.2: Valores de fitness promedio $\hat{f}_t$ de ejemplo utilizados para el cálculo de progreso ilustrado en la figura 3.3.

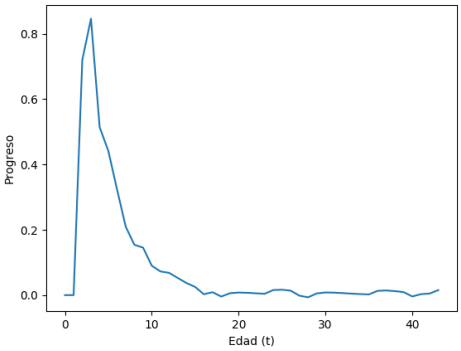
Se considera que una especie se encuentra estancada cuando su valor de progreso se encuentra por debajo de un cierto umbral fijo durante n generaciones consecutivas. Al detectarse el estancamiento de una especie, ésta es reemplazada. De forma similar a [8], se mantiene un registro de las especies que han sido evolucionadas, lo que permite explorar distintas combinaciones de especies sin perder los resultados ya encontrados. Para remplazar una especie, primero se



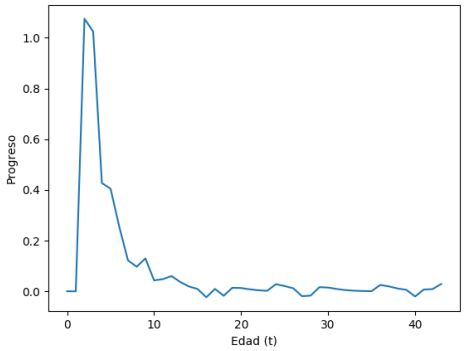
(a) $\lambda = 0$



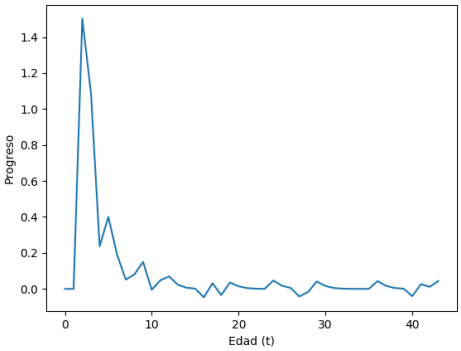
(b) $\lambda = 0.2$



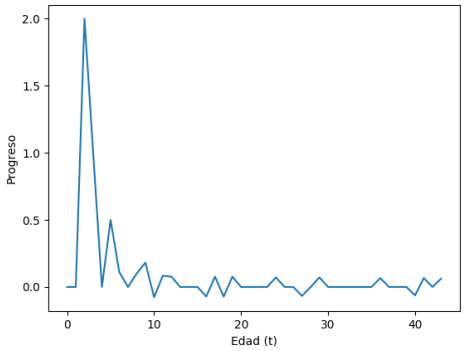
(c) $\lambda = 0.4$



(d) $\lambda = 0.6$



(e) $\lambda = 0.8$



(f) $\lambda = 1$

Figura 3.3: Gráficas de progreso para distintos valores de lambda.

la almacena en el registro. Luego, se intenta formar una nueva especie, distinta de las que se encuentran en ese momento en la simulación. Buscar que la nueva especie sea distinta solamente a las que en ese momento se encuentran en la simulación y no a todas las encontradas en la ejecución del algoritmo causa que la misma especie pueda encontrarse varias veces en el registro. Sin embargo, debido a que los organismos interactúan entre ellos, distintas combinaciones de especies en la simulación pueden requerir distintos comportamientos por parte de los organismos, por lo que dos entradas en el registro de la misma especie pueden corresponder a individuos con comportamientos distintos.

3.5. Simulación

La simulación del comportamiento de los individuos para el cálculo de fitness y los detalles de la función de fitness son dejados en manos del desarrollador, abstrayéndose AGIO de los detalles de implementación. El desarrollador debe implementar la función *computarFitness*, que recibe como parámetro un conjunto de individuos y calcula sus valores de fitness. Cabe destacar que esta función no siempre es invocada para el conjunto de todos los individuos de la población, porque no todos los individuos son simulados a la vez. La cantidad de individuos que se simulan en un llamado a *computarFitness* es un parámetro configurable por el desarrollador, distinto al tamaño de la población. Si bien el fitness se calcula para toda la población en cada paso evolutivo, el cálculo se realiza en grupos disjuntos de individuos, denominados batches, con la cantidad de individuos por batch indicada por el desarrollador.

La razón para realizar el cálculo del fitness en batches es que como los individuos interactuar entre ellos y con el entorno, la cantidad de individuos presentes al mismo tiempo puede afectar los valores de fitness y el comportamiento óptimo. Si existe, por ejemplo, una gran cantidad de individuos en un entorno de presa-depredador, el comportamiento óptimo de los depredadores podría ser estar quieto hasta que una presa pase lo suficientemente cerca como para poder comerla. En cambio, en un entorno con escasos individuos, el depredador se verá obligado a moverse hacia las presas para poder comer. Realizar la simulación en batches permite tener un tamaño de población definido en base a consideraciones de tiempo y de nivel de fitness esperados, sin afectar el comportamiento de los individuos en las simulaciones.

La función computarFitness es libre de asignar a los individuos valo-

res de fitness en el intervalo $(-\infty, +\infty)$, pero la implementación utilizada de NEAT no funciona correctamente cuando existen individuos con valores de fitness negativos o cero. Para solucionar este problema, los resultados de computarFitness son mapeados al intervalo $[1, +\infty)$ aplicando la función $f(x) : \mathbb{R} \mapsto [1, +\infty)$ detallada en la ecuación 3.6. La gráfica de f se muestra en la figura 3.4. La función de mapeo elegida f es monótona estrictamente creciente para conservar el orden del fitness de los individuos, es decir, si se cuenta con dos individuos con fitness a y b tales que $a < b$, entonces, $f(a) < f(b)$. Sin embargo, la elección de la función de mapeo es arbitraria, y podría ser cualquier otra función monótona estrictamente creciente.

$$f(x) = log_2(2^{0.1x} + 1) + 1 \tag{3.6}$$

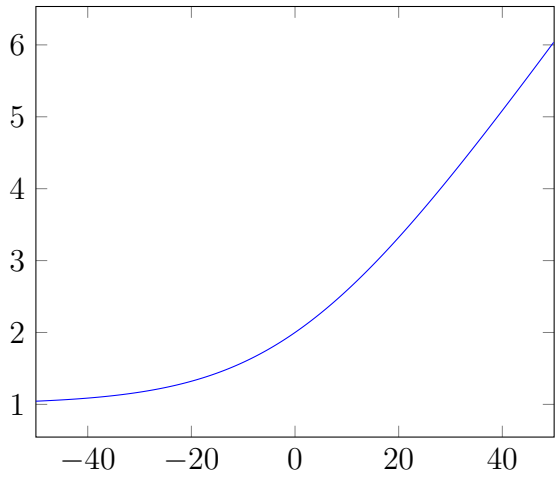


Figura 3.4: Gráfica de la función de mapeo de fitness utilizada.

3.6. Pasos de evolución

Al igual que en cualquier algoritmo evolutivo, el progreso en AGIO se da en pasos llamados *epoch*, que corresponden a las distintas generaciones de la población. El procedimiento realizado en un epoch es descripto en el algoritmo 3.

Un epoch comienza con la evaluación de la población, de la forma detallada en la sección 3.5, para tener un valor de fitness en cada individuo. Luego de obtener el fitness para cada uno de los individuos de la población el siguiente

Algoritmo 3: Epoch

```
// Se calcula el fitness de los individuos de la población
evaluar_poblacion()
// Para cada especie se ejecuta el epoch de NEAT
para cada especie en especies hacer
    especie.neat_epoch()
    // Se asigna a los individuos las nuevas redes de NEAT
    para i=1 a especie.cantidad_individuos hacer
        individuo = especie.individuos[i]
        individuo.red_neuronal = especie.población_neat[i]
    fin
fin
// Se actualizan las especies estancadas
actualizar_especies()
```

paso es procesar las especies. El procesamiento de una especie consiste en realizar el epoch de la población de NEAT asociada a la especie y actualizar la red neuronal de cada individuo de la especie por una de las nuevas redes neuronales producidas por NEAT. Se procesan todas las especies que se encuentran actualmente en la población.

Con las redes de las especies ya actualizadas, se procede a evaluar el progreso de cada especie. En base al progreso, se decide si la especie continuara la población o si será reemplazada por una nueva especie. Como se describe en la subsección 3.4, si se decide que una especie debe ser reemplazada se la almacena en el registro, se busca una morfología distinta a las que se encuentran actualmente en la población y si se encuentra una morfología, se crea una nueva especie con esa morfología que remplaza a la anterior. En caso de no encontrar una morfología nueva, se regenera la población de NEAT asociada a la especie tomando como genomas base para las redes neuronales no solo el genoma mínimo de NEAT, donde todas los nodos de entrada se encuentran conectados con todos los nodos de salida y no hay nodos ocultos, sino que también los genomas pertenecientes a los individuos existentes en el registro para esa especie. Regenerar la población de NEAT utilizando los genomas de los individuos que fueron agregados al registro permite comenzar el proceso evolutivo de la especie nuevamente pero sin perder la información ya encontrada. El procedimiento para la actualización de las especies se detalla en el algoritmo 4 .

Algoritmo 4: Procedimiento para la actualización de las especies.

```
para cada especie en especies hacer
    progreso = calcular_progreso(especie)
    // La especie es remplazada si se detecta estancamiento
    progreso_minimo = configuración_parámetros.progreso_mínimo
    si progreso < progreso_mínimo entonces
        registro.agregar(especie.mejor_individuo)
        morfología = buscar_nueva_morfología()
        si no se encontró una nueva morfología entonces
            morfología = especie.morfología
        fin
        cantidad_sensores = morfología.cantidad_sensores
        cantidad_acciones = morfología.cantidad_acciones
        cantidad_individuos = especie.cantidad_individuos
        // Para la morfología encontrada se busca en el registro
        // genomas de individuos que ya hayan sido evolucionados
        // y a partir de ellos se genera una nueva población de NEAT
        genomas = registro.genomas_guardados(morfología)
        genoma_mínimo = NEAT:Genoma(cantidad_sensores,
          cantidad_acciones)
        genomas.agregar(genoma_mínimo)
        población_neat = NEAT:Población(genomas,
          cantidad_individuos)
        especie = crear_especie(morfología, población_neat)
    fin
fin
```

3.7. Implementación y flujo de uso

Se realizó una implementación del algoritmo utilizando C++ 17 para poder evaluar la propuesta y presentar una biblioteca que pueda ser posteriormente utilizada para otros proyectos. Para la serialización se utilizó Boost y para las gráficas Matplotlib, a través de la biblioteca Matplotlib-cpp.

El flujo de uso que un desarrollador tiene que seguir al utilizar AGIO es el siguiente:

1. Especificar las acciones, sensores, parámetros, componentes y grupos de componentes.
2. Implementar las funciones de evaluación de la población, y de manejo del estado del individuo.
3. Especificar la configuración a utilizar.
4. Crear una población de AGIO y los recurso que sean necesarios para la simulación.
5. Llamar al paso evolutivo de AGIO tantas veces como sea deseado.
6. Guardar la población actual al registro. Como las especies solo se guardan en el registro cuando se estancan, puede pasar que el registro esté vacío porque ninguna especie se estancó.
7. Serializar el registro a un archivo en disco.

El registro contiene toda la información necesaria para poder crear nuevamente los individuos de todas las especies encontradas. El archivo del registro sería lo que se distribuiría con un juego que use AGIO, y en la inicialización del juego se cargaría ese archivo en memoria y se generarían los individuos que se necesiten a partir de la información guardada.

Capítulo 4

Metodología del análisis experimental

En este capítulo se presentan los experimentos realizados para evaluar el proyecto y se detalla el proceso de ajuste de parámetros.

4.1. Escenarios

Esta sección presenta y detalla los escenarios sobre los que se realizaron los experimentos. Se presentan dos escenarios, uno simple para validar que el algoritmo puede generar individuos con comportamientos interrelacionados, y otro más complejo para evaluar todos los componentes del algoritmo. Los entornos en los que se encuentran los individuos en cualquiera de los escenarios son 2D, discretos y cíclicos (si un individuo sale por un borde, aparece por el borde opuesto). Las posiciones discretas del entorno se denominan celdas y tienen un tipo asociado que afecta como los individuos interactúan con esas posiciones. Para medir la distancia entre dos posiciones se utiliza la norma infinito. Por ejemplo, la distancia entre la posición $(1, 1)$ y la posición $(3, 7)$ es 6.

4.1.1. Escenario simple

Se tiene un escenario simple, para poder verificar la interrelación entre el comportamiento de los individuos de distintas especies. El entorno utilizado en este escenario es de 40×40 celdas, donde las celdas son todas de un mismo tipo denominado Suelo. Realizando una analogía con la naturaleza, se define una

especie de herbívoros y una especie de carnívoros. La especie de herbívoros, tiene la acción de consumir la comida que se encuentra distribuida aleatoriamente en el entorno. Por otra parte, la especie de carnívoros tiene la acción de comer a los herbívoros (no se simula peleas entre individuos). Un carnívoro solo puede comer a un herbívoro que esté a dos o menos celdas de distancia y un herbívoro solo puede comer si la comida se encuentra a cero o una celda de distancia. El movimiento de los individuos en el entorno es común a las dos especies y está definido por cuatro acciones. Estas acciones permiten a los individuos moverse a la celda anterior o siguiente, tanto en el eje X como en el eje Y.

Para formar a los individuos se definieron dos grupos de componentes, como se muestra en las figuras 4.1 y 4.2. El primer grupo, denominado Movimiento, tiene un único componente. Este único componente no cuenta con sensores pero sí tiene asociadas las cuatro acciones de movimiento. El segundo grupo, denominado Alimentación, tiene dos componentes: Componente-Herbívoro y Componente-Carnívoro. El denominado Componente-Herbívoro, tiene asociada la acción de comer comida del entorno y dos sensores que indican la diferencia en cada eje entre la posición del individuo y la comida más cercana. A su vez, el denominado Componente-Carnívoro tiene asociada la acción de comer a un herbívoro y dos sensores que indican la diferencia en cada eje entre la posición del individuo y el herbívoro más cercano. El grupo Movimiento esta configurado para que se requiera como mínimo y como máximo un componente de ese grupo en los individuos. Dado que el grupo tiene solamente el componente Movimiento, todos los individuos van a contar con ese componente. El grupo alimentación se configuró de la misma manera, pero como hay dos componentes, AGIO elige aleatoriamente uno de los dos al momento de formar un individuo.

Respecto a las reglas de la simulación, cada individuo lleva un *puntaje* que es utilizado como su fitness. El puntaje de un individuo se incrementa cuando come, ya sea comida en el caso de los herbívoros o a otro individuo en el caso de los carnívoros. Además, en el caso de los herbívoros, el puntaje se decrementa cuando son comidos. Para otorgar a los individuos el mayor tiempo posible para la evaluación de su comportamiento, cuando un herbívoro es comido no es retirado de la simulación, sino que es movido a una posición aleatoria dentro del mundo. Dado que existe una restricción de distancia para realizar la acción de comer, puede suceder que un individuo trate de realizar la

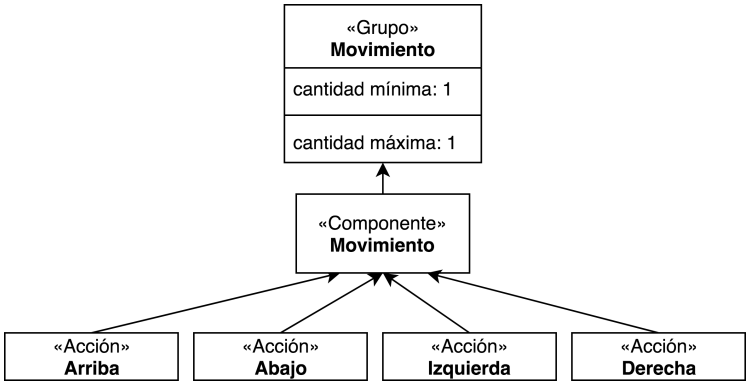


Figura 4.1: Estructura del grupo de componentes que define las acciones de movimiento.

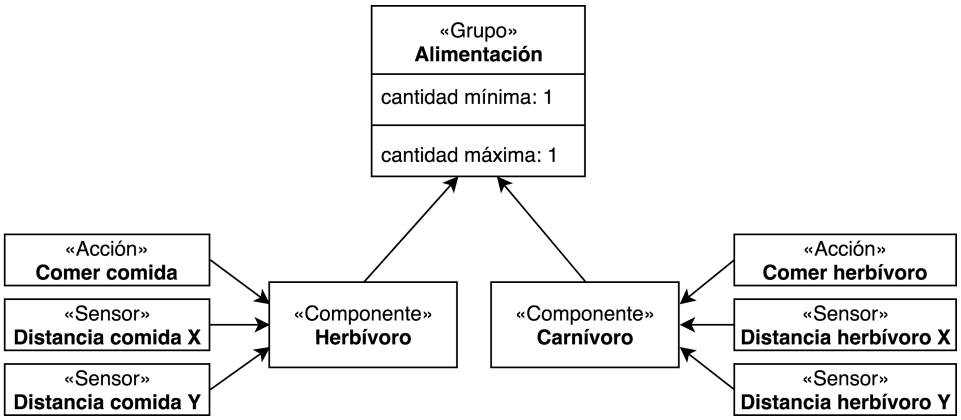


Figura 4.2: Estructura del grupo de componentes que define las acciones de comer y los sensores para detectar la comida más cercana.

acción en un momento en el cual la acción no es válida. Cuando un individuo decide ejecutar la acción de comer sin tener comida a una distancia adecuada, se dice que la acción falló y se penaliza al individuo restándole un valor fijo a su fitness.

Finalmente, para poder comparar el comportamiento generado de los individuos contra un comportamiento conocido, se desarrollo un algoritmo greedy para la toma de decisiones de un individuo. El algoritmo greedy decide realizar la acción que mueva al individuo a la comida más cercana y solo realiza la acción de comer cuando la comida se encuentra a una distancia adecuada.

4.1.2. Escenario principal

El escenario principal, llamado así porque es el que se utilizó en la mayoría de los experimentos, es una complejización del escenario simple. La primer di-

ferencia se encuentra en los tipos de celda utilizados. Mientras que el escenario simple solo cuenta con el tipo de celda Suelo, en el escenario principal se cuenta con los siguientes tipos de celda:

- Pared: no permite que los individuos se muevan a la celda.
- Agua: le quita vida a los individuos que no tienen la acción de nadar.
- Suelo: no tiene ninguna regla particular.

Los entornos en este escenario son de 50×32 celdas, más grandes que en el escenario simple.

Otra diferencia es que se reemplaza el concepto de puntaje por el concepto de vida. La *vida* es un valor numérico asociado a cada individuo y si durante la simulación llega a cero, el individuo es considerado muerto y no puede realizar más acciones. Los individuos comienzan la simulación con una cantidad fija de vida. Cuando un individuo falla una acción se le resta una cantidad fija de vida y cuando realiza exitosamente la acción de comer se le incrementa otra cantidad fija. A diferencia del escenario simple donde el puntaje era el fitness, en el escenario principal el fitness se define como la vida acumulada del individuo, desde el comienzo de la simulación hasta que el individuo muere o la simulación termina.

El escenario principal introduce un nuevo componente al grupo Alimentación y por lo tanto otro tipo de individuos, los individuos omnívoros, continuando con la analogía a la naturaleza presentada en el escenario simple. La composición del grupo Alimentación se muestra en la figura 4.3. Los herbívoros se alimentan de igual forma que en el escenario simple. Los carnívoros y omnívoros, en este escenario, solo pueden comer un individuo de otra especie que este muerto y a una celda o menos de distancia. A los individuos de otras especies se los denomina competidores. Adicionalmente, los individuos carnívoros y omnívoros tienen definida la acción de matar a un competidor que se se encuentre a una celda o menos de distancia.

Respecto a el movimiento, se parte del grupo Movimiento definido en el entorno simple agregandose la restricción de que un individuo no puede moverse a una celda del tipo pared. Además, al grupo se le agrega el componente Movimiento-Acuático. Este componente define las acciones Arriba-Acuático, Abajo-Acuático, Derecha-Acuático y Izquierda-Acuático que permiten moverse sobre las celdas de tipo agua sin perder vida. El grupo Movimiento se muestra en la figura 4.4.

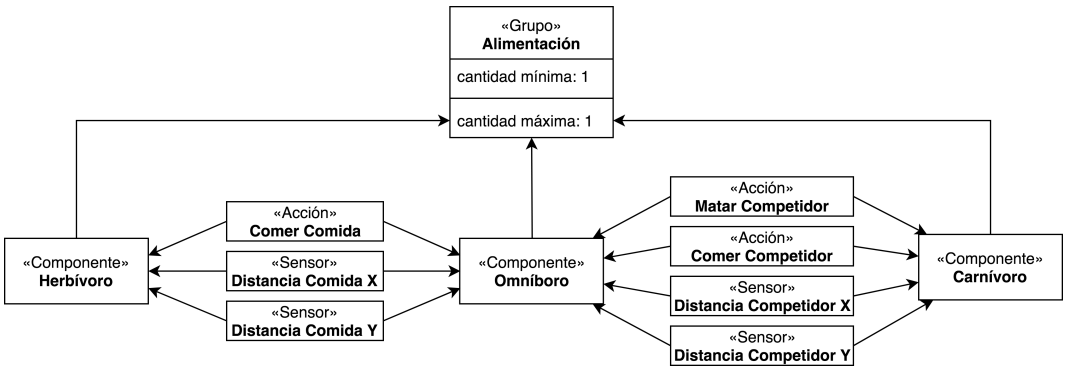


Figura 4.3: Estructura del grupo de componentes que define las acciones de comer y los sensores para detectar la comida más cercana, donde se agrega el nuevo componente Omníboro.

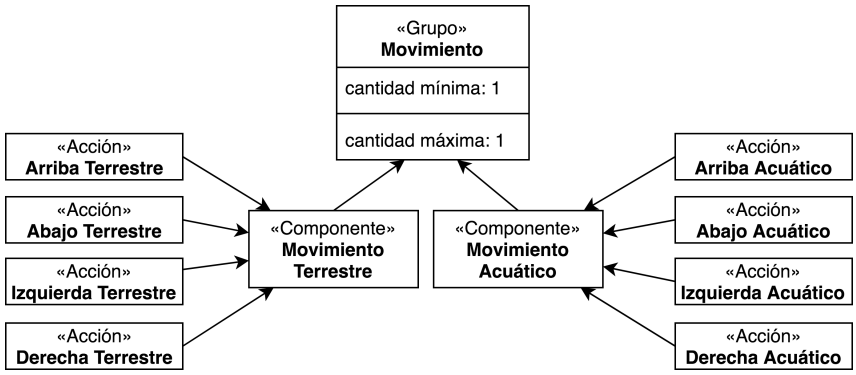


Figura 4.4: Estructura del grupo de componentes que definen las acciones de movimiento para el escenario principal.

Para permitirle a los individuos superar las paredes, se agrega el grupo Salto. Este grupo es de presencia opcional en los individuos, es decir, la cantidad mínima de componentes del grupo que deben de estar presente en los individuos es cero. El grupo cuenta con un único componente, el cual define cuatro nuevas acciones: Saltar-Arriba, Saltar-Abajo, Saltar-Izquierda, Saltar-Derecha. Las acciones de saltar permiten moverse una distancia mayor a una celda y en consecuencia permiten pasar a través de las celdas de tipo Pared. Esta distancia es un parámetro del individuo (como se define en la sección 3.1), cuyo dominio es $[2, 5]$. La composición del grupo Salto se muestra en la figura 4.5. Las individuos que tienen las acciones de saltar no pierden las acciones de movimiento normales, ya sea movimiento terrestre o acuático.

Por último, se agregan dos nuevos grupos: Tipo-Celda y Detección-Competidor. Estos grupos se muestran en las figuras 4.6 y 4.7, respectivamente. El grupo Tipo-Celda cuenta con un único componente, el cual debe de estar

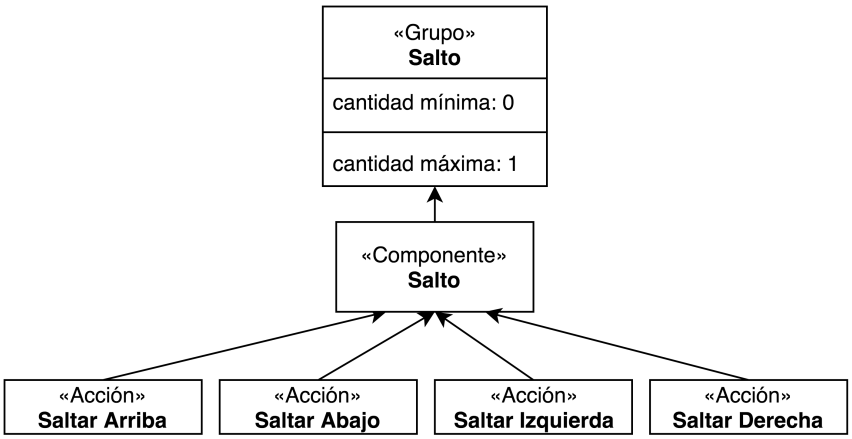


Figura 4.5: Estructura del grupo Salto, en el cual se define las acciones de saltar.

presente en todos los individuos. Este componente agrega a los individuos cinco sensores, que indican el tipo de celda donde el individuo se encuentra y el tipo de las cuatro celdas adyacentes al individuo en los ejes X e Y. Por otra parte, el grupo Detección-Competidor también cuenta con un único componente, pero en este caso el componente es opcional. Este componente brinda a los individuos dos sensores, los cuales indican la diferencia entre la posición del individuo y el competidor más cercano. La existencia de este grupo permite a los herbívoros tener información de los competidores, para potencialmente poder realizar acciones evasivas. Acciones o sensores repetidos en un individuo son ignorados.

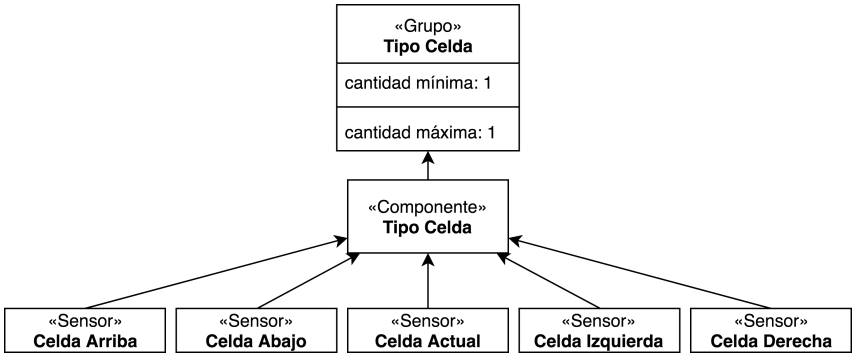


Figura 4.6: Estructura del grupo de componentes que definen los sensores de detección del tipo de celdas que rodea al individuo.

A diferencia del escenario simple donde se tenían solo dos especies, en el escenario principal la configuración utilizada para los grupos permite generar 24 especies distintas, donde algunas van a corresponder a herbívoros, otras a carnívoros y otras a omnívoros.

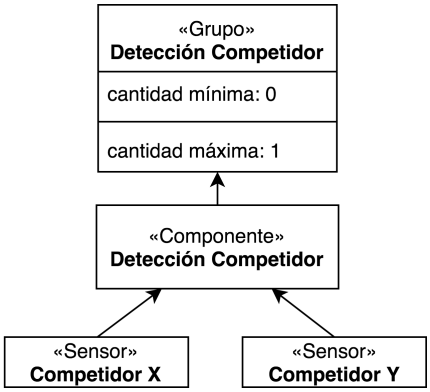


Figura 4.7: Estructura del grupo de componentes que definen los sensores de detección de competidores.

4.1.3. Entornos

En las figuras 4.8, 4.9, 4.10, 4.11 y 4.12 se muestran los distintos entornos utilizados en el escenario principal. Las celdas de tipo pared se representan en gris, las celdas de tipo agua en celeste y las celdas de tipo suelo en naranja. Adicionalmente, se muestra la comida que los herbívoros pueden consumir en verde. La posición de la comida mostrada es solo a modo de ejemplo, al comienzo de cada simulación las posiciones son elegidas aleatoriamente.

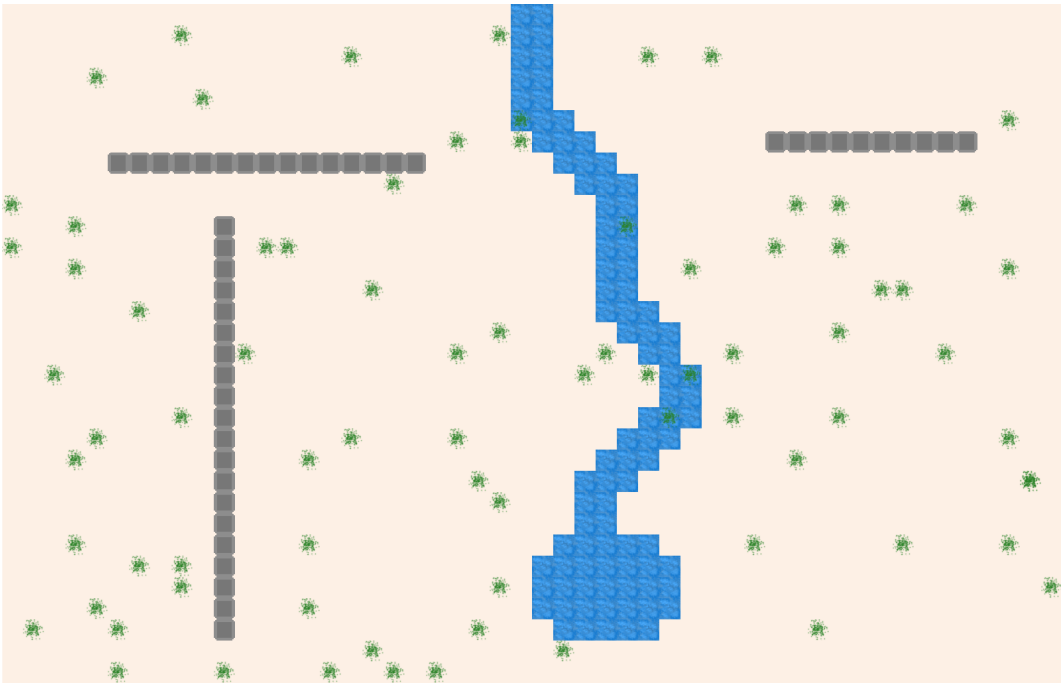


Figura 4.8: Entorno con todos los tipos de celdas, para realizar una prueba completa de las distintas especies.

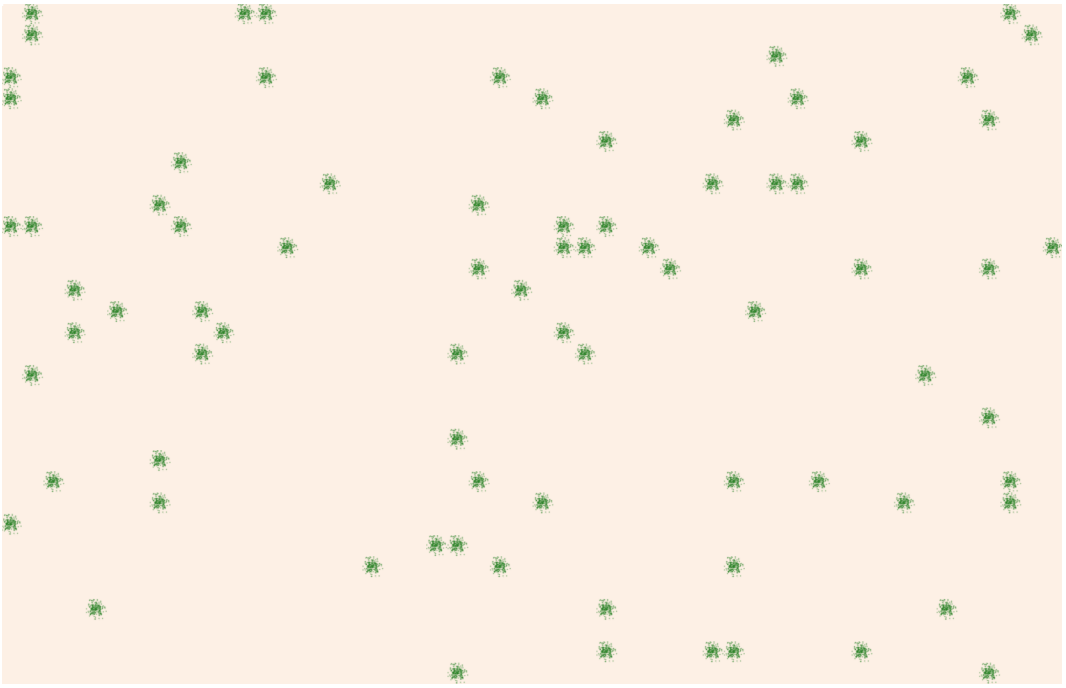


Figura 4.9: Entorno mínimo, solo con celdas de tipo suelo. Este entorno corresponde al utilizado en el escenario simple.

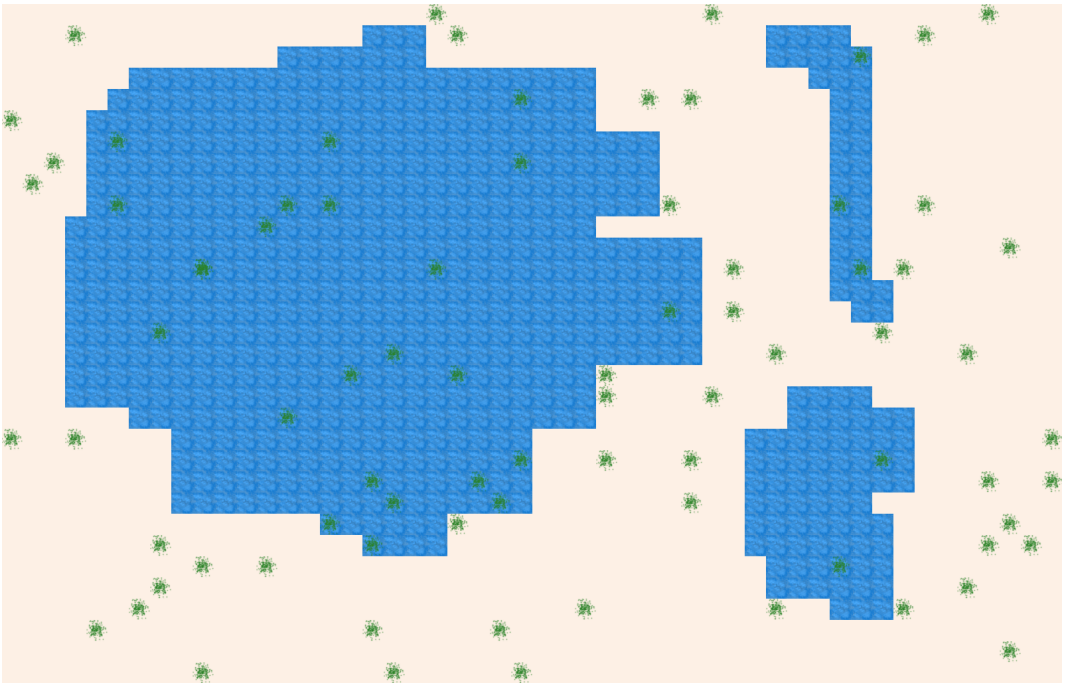


Figura 4.10: Entorno solamente con celdas de tipo agua y suelo. Una gran extensión de agua contigua causa que los individuos sin la acción de nadar pierdan mucha vida si tratan de cruzar de un lado al otro.

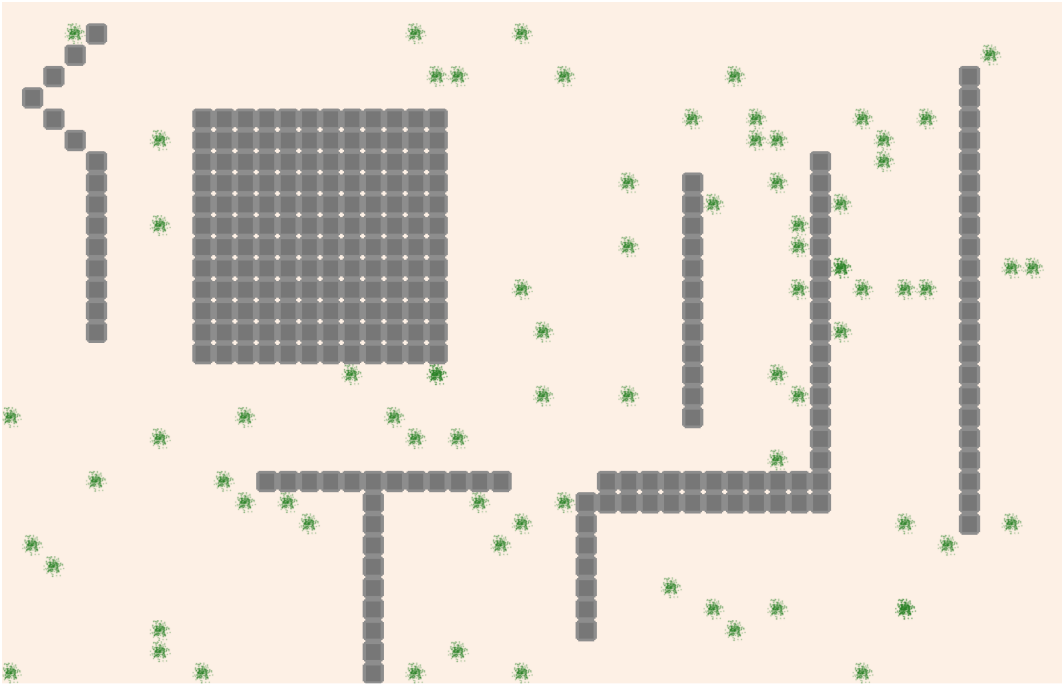


Figura 4.11: Entorno solamente con celdas de tipo pared y suelo. Este entorno está pensado para dificultar la exploración por la cantidad y disposición de las celdas de tipo pared.

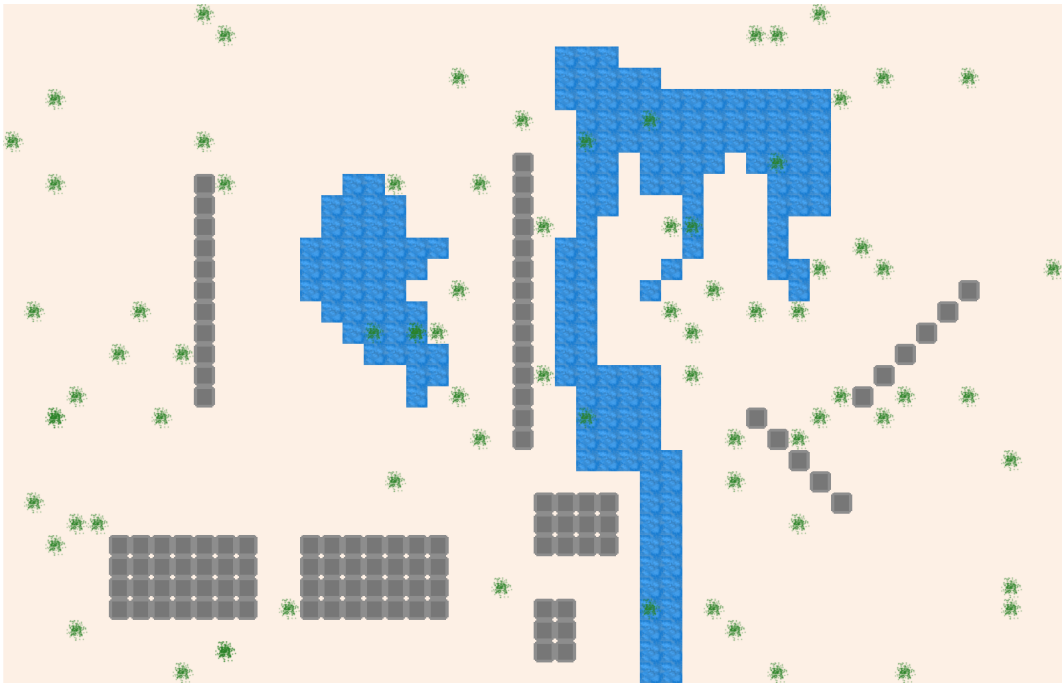


Figura 4.12: Un segundo entorno con todos los tipos de celdas, para introducir variedad a los experimentos.

4.2. Configuración paramétrica

En esta sección se detalla la etapa de ajuste de los parámetros del algoritmo. Es importante notar que cuando se refiere en esta sección a parámetros se habla de los valores numéricos que controlan el funcionamiento del algoritmo, como por ejemplo la cantidad de generaciones utilizadas en la evolución, no se hace referencia a los parámetros de los individuos como se describe en la sección Sección 3.1.

4.2.1. Metodología

Como es habitual al utilizar algoritmos evolutivos, se tiene un conjunto de diversos parámetros que controlan la ejecución y por lo tanto tienen influencia en la calidad de los resultados obtenidos. El ajuste de parámetros es el proceso de encontrar la combinación de valores de los parámetros que devuelva los mejores resultados.

En este proyecto de grado se tienen dos conjuntos de parámetros, los heredados de NEAT y los introducidos por el algoritmo propuesto, AGIO. Dado que AGIO tiene 14 parámetros (detallados en la tabla 4.1) y la biblioteca utilizada de NEAT tiene 33 parámetros, no es posible ajustar todos los parámetros simultáneamente. Probar cuatro valores candidatos por parámetro resulta en aproximadamente 2.6×10^{22} combinaciones posibles y cada ejecución del algoritmo demora varios minutos (como se detalla en la subsección 4.3.5). Además, no se tiene documentación suficiente como para decidir cuales son rangos adecuados de valores a probar para cada parámetro de NEAT. Por estas razones, se decidió no ajustar los valores de los parámetros de NEAT y utilizar una asignación de valores provista por la biblioteca.

A pesar de no ajustar los parametros de NEAT, aún se tiene los 14 parámetros de AGIO. Considerando nuevamente cuatro valores candidatos para cada parámetro, resultan en 268.435.456 combinaciones distintas y por lo tanto requeriría un tiempo tiempo total de ejecución de aproximadamente 1532 años, lo que muestra que no es viable ajustar todos los parámetros simultáneamente. Como solución se introduce el concepto del análisis de sensibilidad, que consiste en probar distintos valores de un parámetro, dejando el resto de los parámetros fijos y evaluar que tanto varían los resultados del algoritmo con las variaciones en ese parámetro [4]. De esta manera se pueden identificar parámetros que no es necesario ajustar por tener poca influencia en los resultados.

Nombre	Descripción
MinIndividualsPerSpecies	Cantidad mínima de individuos que puede tener una especie cuando se genera.
MinSpeciesAge	Edad mínima de una especie antes de que se empiece a considerar su progreso para el chequeo de estancamiento.
ProgressMetricsIndividuals	Cantidad de individuos de una especie utilizados para medir el progreso.
ProgressMetricsFalloff	Intensidad del suavizado de la métrica de progreso de las especies.
ProgressThreshold	Mínimo progreso que tiene que hacer una especie para que no se la considere estancada.
SpeciesStagnancyChances	Cantidad de generaciones que se le permite a la especie tener un progreso por debajo del mínimo.
MorphologyTries	Cantidad de intentos antes de asumir que no existe una morfología que no esté entre las morfologías actuales cuando se está buscando una nueva morfología.
ParameterMutationProb	Probabilidad de mutar un parámetro de un individuo.
ParameterDestructiveMutationProb	Probabilidad de que la mutación de un parámetro sea destructiva.
ParameterMutationSpread	σ de la distribución normal con la que son mutados los parámetros para la mutación no destructiva.
PopSizeMultiplier	Valor que multiplicado por el tamaño de la simulación determina el tamaño de la población.
MaxSimulationSteps	Cantidad de pasos a ejecutar en cada simulación.
GenerationsCount	Cantidad de generaciones.
SimulationReplications	Cantidad de repeticiones de cada simulación que se realizan para calcular resultados con significancia estadística.

Tabla 4.1: Descripciones de todos los parámetros de AGIO como se encuentran en los archivos de configuración, sin contar los heredados de NEAT.

La figura 4.13 muestra el proceso de configuración de parámetros, donde

en fondo negro se muestran las etapas de configuración y en fondo blanco las entradas y salidas de esas etapas.

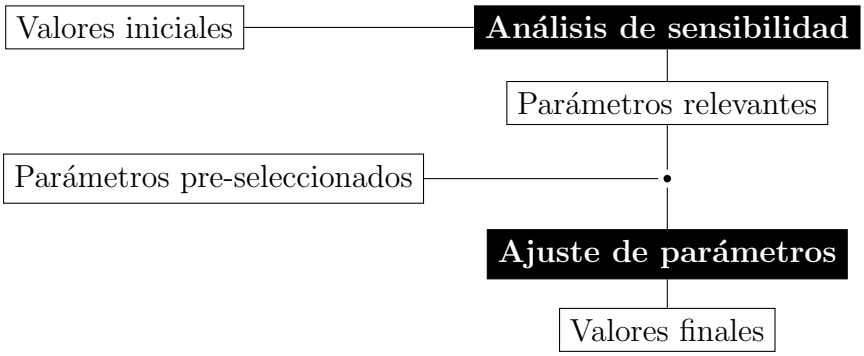


Figura 4.13: Proceso de ajuste de parámetros. El flujo del proceso es de arriba hacia abajo en el diagrama.

4.2.2. Análisis de sensibilidad

Para el análisis de sensibilidad se seleccionaron diez parámetros, MinIndividualsPerSpecies, MinSpeciesAge, ProgressMetricsIndividuals, ProgressMetricsFalloff, ProgressThreshold, SpeciesStagnancyChances, MorphologyTries, ParameterMutationProb, ParameterDestructiveMutationProb y ParameterMutationSpread. Se probaron cuatro valores distintos para cada parámetro, especificados en la tabla 4.3. Los cuatro parámetros que se dejaron fuera del análisis de sensibilidad fueron PopSizeMultiplier, MaxSimulationSteps, SimulationReplications y GenerationsCount. Tanto el multiplicador del tamaño de la población como la cantidad de generaciones fueron dejados de lado porque es un resultado conocido en algoritmos evolutivos que el tamaño de la población y la cantidad de generaciones afectan significativamente los resultados obtenidos [23]. Respecto a la cantidad máxima de pasos de simulación, este parámetro afecta directamente la fitness posible de los individuos, porque controla cuantas acciones pueden realizar y por lo tanto se asume que va a afectar los resultados. Por último, la cantidad de replicaciones de la simulación fue dejada de lado de la configuración paramétrica porque el valor ideal de este parámetro es el mayor posible y cual valor usar depende de cuanto tiempo se disponga para la ejecución del algoritmo.

Partiendo de la configuración listada en la tabla 4.2 se realizaron 40 pruebas, donde en cada prueba se cambió el valor de un solo parámetro, tomando los valores de la tabla 4.3. Cada prueba consistió en realizar 50 ejecuciones

independientes del algoritmo para cada uno de los entornos de prueba presentados en la subsección 4.1.3, resultando en 250 ejecuciones por prueba. De cada ejecución se almacenan las especies obtenidas y para cada especie se almacena el promedio del fitness con el que aparece la especie en el registro.

Una vez realizadas todas las pruebas, se calcula para cada prueba el promedio de la cantidad de especies encontradas en cada ejecución y para cada ejecución se guarda el fitness de cada especie encontrada, para luego promediar en cada especie los valores de fitness. De esta manera se tiene información sobre la cantidad y la calidad de las especies encontradas con cada valor candidato de cada parámetro, obteniéndose una lista de cantidad de especies y fitness de cada especie. Como interesa saber qué impacto tiene el parámetro sobre los resultados, se calcula el coeficiente de variación de cada lista, que corresponde al cociente entre la desviación estándar y la media de los valores de la lista. Con los coeficientes de variación calculados, se tiene para cada parámetro una métrica que permite medir cuanto afecta la variación del parámetro a los resultados en cuanto a la cantidad de especies y otra métrica sobre el efecto en la calidad de las especies. Realizando el promedio entre las dos métricas se tiene una métrica final de la sensibilidad de AGIO a distintos valores de cada parámetro y por lo tanto se puede decidir que parámetros incluir en el ajuste.

Parámetro	Valor
MaxSimulationSteps	150
PopSizeMultiplier	15
GenerationsCount	150
MinSpeciesAge	20
ProgressMetricsIndividuals	5
ProgressMetricsFalloff	0.025
ProgressThreshold	0.005
SpeciesStagnancyChances	20
MinIndividualsPerSpecies	50
MorphologyTries	10
SimulationReplications	10
ParameterMutationProb	0.1
ParameterDestructiveMutationProb	0.1
ParameterMutationSpread	0.025

Tabla 4.2: Valores iniciales utilizados en el análisis de sensibilidad

Parámetro	Valores candidatos			
MinIndividualsPerSpecies	5	10	50	100
MinSpeciesAge	1	10	50	100
ProgressMetricsIndividuals	1	5	20	40
ProgressMetricsFalloff	0.001	0.025	0.1	0.9
ProgressThreshold	0.0001	0.005	0.1	0.9
SpeciesStagnancyChances	1	10	50	100
MorphologyTries	1	10	50	100
ParameterMutationProb	0.01	0.1	0.2	0.5
ParameterDestructiveMutationProb	0.01	0.1	0.2	0.5
ParameterMutationSpread	0.001	0.025	0.1	0.9

Tabla 4.3: Valores candidatos utilizados en el análisis de sensibilidad

4.2.3. Ajuste de parámetros

A los tres parámetros seleccionados en la subsección 4.2.2 se agregaron tres parametros más para el ajuste: el multiplicador del tamaño de la población, la cantidad de generaciones y la cantidad máxima de pasos de simulación. El multiplicador del tamaño de la población define, junto con el tamaño de la simulación, el tamaño de la población. Como se menciona en la subsección 4.2.2, el tamaño de la población y la cantidad de generaciones son parámetros clásicos de ajuste en un algoritmo evolutivo y como el tamaño de la simulación viene definido por el usuario, es necesario hacer un ajuste del multiplicador del tamaño de la población [23]. Por otra parte, cuanto mayor sea la cantidad máxima de pasos de simulación, los individuos pueden realizar más acciones y por lo tanto su evaluación va a representar más exactamente su comportamiento. Sin embargo, la cantidad de pasos aumenta linealmente el tiempo de simulación y por lo tanto el tiempo de evolución. Es necesario entonces encontrar un balance para la cantidad máxima de pasos de simulación. Resumiendo, los parámetros ajustados fueron PopSizeMultiplier, GenerationsCount, MaxSimulationSteps, ProgressMetricsFalloff, ProgressThreshold y MinIndividualsPerSpecies.

Para los parámetros estudiados, los valores candidatos se detallan en la tabla 4.4.

Con los valores de la tabla 4.4 se tienen 729 combinaciones posibles, que se evaluaron tres veces para cada uno de los entornos presentados en la subsección 4.1.3, lo que resultó en 10935 ejecuciones en total. Para cada configuración se registraron los mismos valores que para el análisis de sensibilidad: la cantidad de especies y el fitness promedio de cada especie. Para poder realizar

Parámetro	Valores candidatos		
PopSizeMultiplier	10	20	30
GenerationsCount	100	250	400
MaxSimulationSteps	50	150	300
ProgressMetricsFalloff	0.001	0.025	0.1
ProgressThreshold	0.001	0.005	0.02
MinIndividualsPerSpecies	25	50	75

Tabla 4.4: Valores utilizados para el ajuste de parámetros.

comparaciones de fitness entre distintas especies, se normaliza el fitness promedio de cada especie con el fitness máximo alcanzado por esa especie. Luego para cada configuración se promedian los valores normalizados de fitness de las distintas especies encontradas, llamándose a este valor *fitness normalizado promedio*. Resulta entonces que para cada configuración se tienen dos métricas, la cantidad de especies encontradas y el fitness normalizado promedio de las especies.

4.3. Experimentos realizados

En esta sección se presentan los experimentos realizados sobre los escenarios detallados en la sección 4.1.

4.3.1. Experimento mínimo

El experimento mínimo, que ejecuta sobre el escenario simple, fue diseñado para poder validar el algoritmo en las condiciones más simples posibles. Trabajar con un escenario reducido permite ejecutar un algoritmo greedy para el comportamiento de los individuos y tenerlo como punto de comparación. El algoritmo greedy se toma como el comportamiento esperado de cada especie. Además, trabajar con solo dos especies permite visualizar más fácilmente las métricas de evolución. Dado que el fitness no permite saber de manera simple cual fue el comportamiento del individuo se decidió evaluar las siguientes tres métricas:

- Cantidad de veces que los individuos comen (a otro individuo en el caso de los carnívoros o la comida del entorno en el caso de los herbívoros).
- El cociente entre la cantidad de acciones realizadas que fallaron y la cantidad de acciones que podrían haber fallado. En el escenario utilizado,

solo la acción de comer puede fallar. El concepto de fallar acciones se presenta en la subsección 4.1.1.

- Cantidad de diferentes celdas del mundo recorridas en promedio por los individuos de cada especie.

Respecto a la cantidad de veces que los individuos comen, es esperable que se incremente a medida que pasan las generaciones, aunque puede pasar que si los carnívoros mejoran, los herbívoros empeoren (no viceversa, porque los herbívoros no tienen conocimiento de donde están los carnívoros dado que no tienen ningún sensor que provea esa información). Respecto a la cantidad de acciones fallidas, su número debería disminuir para las dos especies a medida que avanzan las generaciones. La última métrica, la cantidad de celdas recorridas, no tiene un comportamiento esperado, ya que un organismo bueno puede recorrer poco espacio, pero permite tener más información sobre como se comportan los individuos. Conocer la cantidad de celdas recorridas permite saber si los individuos exploran el entorno o se quedan explotando una región cercana a donde inician la simulación.

Durante cada generación se registraron las tres métricas detalladas anteriormente para cada una de las dos especies, promediando entre los individuos de cada especie.

4.3.2. Interrelaciones entre especies

Uno de los objetivos de este proyecto de grado es que el comportamiento de los individuos de las especies generadas se encuentre relacionado, existiendo dependencias entre las distintas especies. Para analizar si ese objetivo se cumple o no, se propuso realizar una simulación normal con las especies ya evolucionadas registrando el fitness promedio de los individuos de esa especie al final de la simulación y luego realizar distintas simulaciones removiendo una especie y dejando las demás. Si existen relaciones entre las especies, los valores de fitness de cada especie debería variar significativamente al remover otras especies. Para determinar si una especie A se encuentra relacionada con una especie B, una vez realizadas las simulaciones descritas anteriormente, se evalúa la distancia relativa a la desviación estándar (σ) de la media que se encuentra el fitness obtenido por la especie A cuando B no se encuentra en la simulación. Para confirmar que el fitness de las especies sigue una distribución normal en las simulaciones se realizó previamente un test de normalidad uti-

lizando el test de Shapiro-Wilk [30] y el test de D'Agostino-Pearson [27]. Se considera que si el fitness de la especie A al retirar la especie B se encuentra a una distancia de la media del fitness simulando todas las especies superior a dos veces la desviación estándar, entonces las especie A se encuentra relacionada con la especie B. La decisión de fijar el umbral en dos veces la desviación estándar es porque para una variable aleatoria que sigue una distribución normal, aproximadamente el 95 % de los valores obtenidos van a encontrarse a esa distancia, por lo que encontrar un valor por fuera de esa distancia siguiendo la misma distribución tiene probabilidad igual o menor a 0.05. Esto se puede probar partiendo de la propiedad de la función de probabilidad presentada en la ecuación 4.1, donde F_x es la función de probabilidad, X la variable aleatoria y x, a y b valores en el domino de la variable.

$$P(a < X < b) = F_x(b) - F_x(a) \tag{4.1}$$

Si μ es la media, σ la desviación estándar y $X \sim N(\mu, \sigma)$, la probabilidad de que la variable aleatoria X se encuentre en el rango $(\mu - 2\sigma, \mu + 2\sigma)$ está dada por la ecuación 4.2

$$\begin{aligned} P(\mu - 2\sigma < X < \mu + 2\sigma) &= F_x(\mu + 2\sigma) - F_x(\mu - 2\sigma) \\ &= \Phi\left(\frac{\mu + 2\sigma - \mu}{\sigma}\right) - \Phi\left(\frac{\mu - 2\sigma - \mu}{\sigma}\right) \\ &= \Phi(2) - \Phi(-2) \\ &\approx 0.95 \end{aligned} \tag{4.2}$$

4.3.3. AGIO vs Jugador

Es difícil tener una idea de qué tan buenos (cercanos a lo que un posible jugador puede esperar a priori) son los comportamientos de los individuos analizando solo los resultados numéricos, más aún para el caso del escenario principal donde no se tiene un algoritmo de comparación. Considerando este punto y el foco de este proyecto de grado en videojuegos, donde el jugador humano es un componente crucial, se propuso realizar un experimento donde una persona, llamada jugador, esté a cargo de controlar uno de los organismos evolucionados por AGIO.

Se propusieron dos pruebas, variando la cantidad de información que se le da a al jugador. En la primera, solo se muestran los valores de los sensores y

se pide que elija una acción, sin indicar cual es la correspondencia entre los sensores o acciones y su nombre, denominando los sensores y acciones simplemente por su posición interna en el vector de sensores y acciones del individuo. Mostrarle solamente los valores numéricos de los sensores al jugador se acerca a la información que tienen los individuos evolucionados por AGIO y trata de permitir una comparación justa entre el fitness de los individuos evolucionados por AGIO y el fitness del individuo controlado por el jugador. En la segunda prueba se le otorgó al jugador el nombre de las acciones que puede elegir y de los sensores que ve en pantalla (el jugador ve el valor numérico del sensor), para que pueda suponer lo que hace cada acción y por lo tanto poder tomar decisiones más informadas. Toda la información se presenta mediante texto, mostrando además de los valores de los sensores el valor actual de fitness, en que ranking de fitness se encuentra comparado con los otros organismos de la especie (si es el de mayor fitness, el de segundo mayor fitness, etc), y la vida del individuo. La información extra del fitness y la vida es para darle una guía al jugador de cuán buenas resultan sus decisiones. Durante la ejecución, se registran en un archivo para cada paso las dos métricas mostradas al usuario: fitness y posición dentro de la especie.

4.3.4. Efecto del tamaño de simulación

En el capítulo 3 se dan las razones por las cuales se realiza una simulación en batches de tamaño fijo, pero es importante justificar esas razones con resultados experimentales. El experimento propuesto para justificar la simulación en batches consistió en realizar el proceso evolutivo sobre el escenario simple variando la cantidad de individuos que se simulan simultáneamente (el tamaño de los batches de simulación), desde 10 a 100 individuos. El multiplicador del tamaño de población (PopSizeMultiplier) se modificó en cada evolución para tratar de mantener el mismo tamaño de población con los distintos tamaños de batch en 200 individuos. Como el multiplicador del tamaño de población tiene que ser un entero, no es posible mantener el tamaño de población en 200 para todas las pruebas, pero el valor es siempre cercano. Luego de realizada la evolución, se promedió la cantidad de comida consumida por los individuos de cada especie en la última generación y ese valor se usó para analizar el efecto del tamaño de la simulación en el comportamiento de los individuos.

4.3.5. Tiempo de ejecución y uso de memoria

Como se menciona en la introducción, la motivación de este proyecto de grado está en el ámbito de los videojuegos. En este ámbito se pone un gran énfasis en la eficiencia, respecto a tiempo de ejecución y al uso de memoria para poder obtener la mejor calidad visual o de contenido posible, manteniendo la cantidad de cuadros por segundo normalmente a 30 o 60 en una o varias plataformas, ya sean celulares, consolas o computadoras personales. Teniendo en cuenta la importancia que se le da en videojuegos a la eficiencia, se planteó como un estudio importante analizar el tiempo de ejecución y el uso de memoria de la solución implementada en este proyecto.

AGIO tiene dos etapas bien diferenciadas: la primera es la evolución de las especies y la segunda es la utilización de las especies evolucionadas en el programa deseado. Es importante notar que estas dos etapas son ejecutadas en distintos ambientes. La primera etapa es ejecutada en un ambiente de desarrollo, mientras se está desarrollando el juego en cuestión. La segunda etapa corresponde a la ejecución del juego por los usuarios en sus propias plataformas, esperando tener una experiencia de juego fluida, lo que normalmente implica que los individuos puedan ser evaluados en tiempo real. Por lo tanto, los requerimientos son muy distintos para las dos etapas.

En base a la experiencia en el área y a discusiones en el ámbito laboral (desarrollo de videojuegos) por parte de uno de los integrantes del equipo del proyecto, se concluyó que un tiempo adecuado para la etapa de evolución se encuentra entre cinco minutos a dos horas en un procesador del 2014 en adelante y que cuente con 6 núcleos físicos o más. El extremo de cinco minutos corresponde a querer realizar una evolución rápida sin preocuparse por que tan buenas sean las especies encontradas ni cuantas se encuentren, para poder realizar pruebas rápidas durante el desarrollo del juego, mientras que el tiempo de dos horas corresponde a una ejecución final del algoritmo, buscando los mejores resultados posibles para utilizar en el juego final. Respecto a la cantidad de memoria RAM usada, el algoritmo no debería consumir más de 8 GB para no interferir con otros programas, puesto a que por lo general un equipo en el ámbito de desarrollo cuenta con 32 o 64 GB de RAM. Es mucho más complejo dar un estimativo del consumo de memoria esperado en la etapa de uso del cliente, porque el uso dependerá del producto que se esté desarrollando. Los requerimientos para un juego de celular son mucho más estrictos que para un

juego enfocado a computadoras personales de alta gama, por ejemplo. Asumiendo como producto un juego AAA (juegos con alto presupuesto y grandes equipos de desarrollo) multiplataforma (diseñado para ejecutar sobre PlayStation 4, XBox One y PC), la memoria utilizada por un individuo no debería superar 1 MB, mientras que el tiempo de decisión de una acción no debería ser mayor a 0.25 ms. Si bien estos límites pueden parecer excesivamente estrictos, es importante notar que el consumo de memoria y el tiempo de decisión se tienen que multiplicar por el número de individuos en la simulación y que si se quieren obtener 60 cuadros por segundo, se dispone de solo 16.6 ms para trabajar en cada cuadro. Por último, debe tenerse en cuenta que los resultados de la evolución son guardados a disco. Manteniendo la suposición de un juego AAA, el tamaño de ese registro en disco no debería ser mayor a 100 MB. Si bien 100 MB puede parecer pequeño comparado con los tamaños de juegos AAA actuales que superan normalmente los 50 GB, los resultados de la evolución consisten simplemente en un registro de descripciones de organismos, donde estas descripciones deberían ocupar menos memoria que los individuos finales. Como se definió que los individuos no deberían ocupar en memoria más de 1MB, 100 MB tendrían que permitir almacenar más de 100 descripciones de individuos distintas.

Para evaluar si se lograron los objetivos de memoria, tiempo de evolución y tiempo de decisión, se realizaron distintas ejecuciones en el escenario principal registrando los tiempos de decisión de cada individuo, el tiempo total de evolución y el uso de memoria de AGIO durante la evolución. La memoria utilizada por los individuos se midió sobre los individuos de la última generación de cada ejecución. Las distintas ejecuciones se realizaron sobre los distintos entornos definidos en la subsección 4.1.3 y los resultados de las ejecuciones fueron promediados. Los valores utilizados para los distintos parámetros se encuentran detallados en la tabla 4.5.

Parámetro	Valor
MaxSimulationSteps	150
PopSizeMultiplier	15
GenerationsCount	150
MinSpeciesAge	20
ProgressMetricsIndividuals	5
ProgressMetricsFalloff	0.025
ProgressThreshold	0.005
SpeciesStagnancyChances	20
MinIndividualsPerSpecies	50
MorphologyTries	10
SimulationReplications	10
ParameterMutationProb	0.1
ParameterDestructiveMutationProb	0.1
ParameterMutationSpread	0.025

Tabla 4.5: Valores de parámetros utilizados en el análisis de tiempo de ejecución y uso de memoria

Capítulo 5

Resultados experimentales

En este capítulo se presentan y discuten los resultados obtenidos en la evaluación experimental y se plantean tanto conclusiones generales como posibles trabajos a futuro, incluyendo mejoras y futuras lineas de investigación.

5.1. Configuración paramétrica

Esta sección presenta los resultados obtenidos en el ajuste de parámetros y en el análisis de sensibilidad. Se detalla además las especificaciones de los equipos utilizados en la configuración paramétrica y en los experimentos realizados.

5.1.1. Equipos de prueba

La tabla 5.1 detalla los equipos utilizados durante el desarrollo del proyecto y la evaluación experimental. Los experimentos y la configuración paramétrica fueron realizadas sobre el equipo uno por tener el mayor poder de computo, y ser similar a los equipos de desarrollo utilizados en la industria de videojuegos, mientras que el desarrollo se hizo en los tres equipos. Por temas de disponibilidad del equipo 1, el análisis de tiempos de ejecución y uso de memoria fue realizado en el equipo 2.

	Equipo 1	Equipo 2	Equipo 3
SO	Windows 10 x64 Pro	Windows 10 x64 Home	macOS High Sierra
CPU	Intel Xeon E5-2683v3	Intel Core i7-6700HQ	Intel Core i7-4980HQ
RAM	32 GB DDR4	16 GB DDR4	16 GB DDR3L
GPU	Nvidia RTX 2070	Nvidia GTX 970M	Nvidia GT 720M

Tabla 5.1: Especificaciones de los equipos utilizados para la evaluación experimental y el desarrollo

5.1.2. Análisis de sensibilidad

La ejecución del análisis de sensibilidad llevó 48 horas 49 minutos, realizando 20 ejecuciones en paralelo. Los resultados obtenidos se muestran en la tabla 5.2.

Parámetro	Coeficientes de variación		Promedio Final
	Fitness	N° de Especies	
MinIndividualsPerSpecies	0.44	0.12	0.28
ProgressMetricsFalloff	0.33	0.04	0.19
ProgressThreshold	0.30	0.04	0.17
SpeciesStagnancyChances	0.30	0.03	0.17
MinSpeciesAge	0.18	0.01	0.09
ProgressMetricsIndividuals	0.04	0.06	0.05
MorphologyTries	0.05	0.01	0.03
ParameterMutationProb	0.01	0.01	0.01
ParameterDestructiveMutationProb	0.00	0.01	0.01
ParameterMutationSpread	0.00	0.01	0.01

Tabla 5.2: Resultados obtenidos en el análisis de sensibilidad.

El análisis de la cantidad de especies encontradas mostró que el parámetro con el mayor impacto sobre los resultados es la cantidad mínima de individuos por especie. Que la cantidad mínima de individuos por especie afecte la cantidad de especies encontradas se puede explicar porque la cantidad de especies que se pueden evolucionar simultanemante, para un tamaño de población fijo, esta definido por el tamaño mínimo de las especies. La cantidad de individuos utilizados para calcular la métrica de progreso de cada especie afecta a la cantidad de especies encontradas, pero con un coeficiente de variación que es la mitad que el coeficiente de variación de la cantidad mínima de individuos por especie. El resto de los parámetros presentan un coeficiente de variación que es de tres a doce veces más bajo que el de la cantidad mínima de individuos

por especie.

En cuanto al fitness de los individuos, al igual que en el análisis de la cantidad de especies encontradas, el factor al cual es más sensible el fitness de los individuos es la cantidad mínima de individuos por especie. Que la cantidad de individuos por especie sea un factor importante en el fitness de los individuos encontrados se encuentra dentro de lo esperado, dado que la cantidad de mínima de individuos por especie determina de forma directa cuantas especies se pueden evolucionar simultánemante y por lo tanto el tamaño de las poblaciones de NEAT utilizadas por especie. Es conocido que los resultados de un AE dependen del tamaño de la población [23]. Como cada especie esta asociada a una población de NEAT y ejecuta un AE, los resultados obtenidos van a depender del tamaño de las especies.

El fitness de los individuos encontrados es igual de sensible a cuanto se suaviza la métrica de progreso (la métrica de progreso y su proceso de suavizado se detalla en la subsección 3.4), el umbral de progreso mínimo y la cantidad de generaciones que una especie puede estar por debajo del umbral antes de ser considerada estancada. En menor medida, la calidad de los individuos encontrados es sensible a la edad mínima de una especie antes de que se empiece a medir su progreso. La calidad de los individuos encontrados no es sensible a la cantidad de individuos utilizados para calcular la métrica de progreso, la cantidad de intentos que se realizan cuando se busca una morfología nueva o cualquiera de los parámetros que controlan a los parámetros relacionados con la morfología de los individuos.

Resulta interesante que los parámetros que controlan los parámetros relacionados con la morfología de los individuos no tengan un efecto significativo en los resultados, tanto para la cantidad de especies como para el fitness de los individuos. Una posible explicación a este fenómeno es que el parámetro relacionado con los individuos existente en el entorno utilizado (la distancia a la que se salta, que se define en la sección 4.1.2) no sea de gran importancia para el comportamiento de los individuos.

En base a los resultados obtenidos, se seleccionaron la cantidad mínima de individuos por especie, cuanto se suaviza la métrica de progreso y cuantas generaciones puede estar una especie debajo del umbral de progreso antes de ser considerada estancada, junto con el multiplicador de tamaño de población, la cantidad de generaciones y la cantidad de pasos máximos en cada simulación como los parámetros a ajustar.

5.1.3. Ajuste de parámetros

El ajuste de parámetros fue fue realizado con doce ejecuciones en paralelo, y requirió un tiempo de 119 horas. La gráfica de la figura 5.1 muestra cómo se distribuyen los resultados obtenidos con las distintas configuraciones probadas.

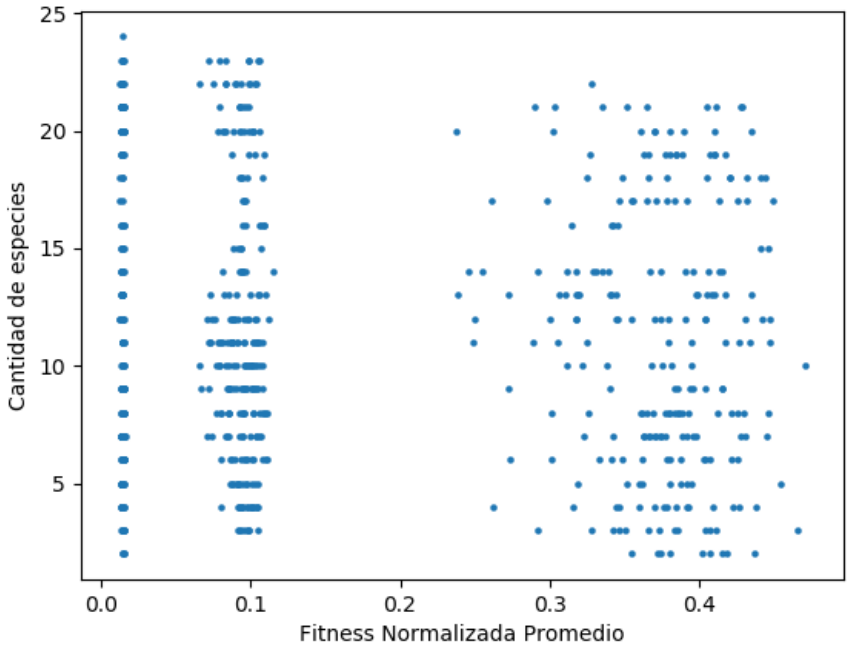


Figura 5.1: Representación visual de los resultados obtenidos en el ajuste de parámetros.

Se distinguen tres áreas bien diferenciadas en los resultados presentados en la figura 5.1. La primer área es una línea de distintas configuraciones que resultan en un fitness similar y muy bajo, aunque con una gran variedad en la cantidad de especies encontradas. La segunda área se centra en un fitness de 0.1 y si bien se extiende nuevamente en un rango amplio de cantidad de especies, se encuentra mucho más dispersa en fitness que el área anterior. Por último, los valores restantes se agrupan en un área con fitness desde 0.25 a 0.45 y cantidad de especies de 1 a 22.

Como uno de los objetivos de este proyecto de grado es diseñar e implementar un algoritmo que genere tanto diversidad como calidad, se toma la configuración más cercana a la esquina superior derecha del gráfico presentado en la figura 4.13, porque esta configuración presenta los mejores resultados en

diversidad de especies y fitness, sin darle prioridad a un aspecto sobre el otro. Los valores seleccionados para los parámetros se detallan en la tabla 5.3.

Parámetro	Valor
PopSizeMultiplier	30
GenerationsCount	400
MaxSimulationSteps	300
ProgressMetricsFalloff	0.025
ProgressThreshold	0.001
MinIndividualsPerSpecies	50

Tabla 5.3: Valores seleccionados para los parámetros en base a los resultados obtenidos.

5.2. Experimentos

Esta sección presenta los resultados obtenidos en los distintos experimentos realizados, junto con un breve análisis de los resultados para cada experimento.

5.2.1. Experimento mínimo

En el experimento mínimo se consideran tres métricas: la cantidad de comida consumida por los individuos, la proporción de acciones de comer fallidas y la cantidad de celdas recorridas. Los resultados obtenidos en cuanto a la comida consumida para los carnívoros se presenta en la figura 5.2 y para los herbívoros en la figura 5.3. Las gráficas presentadas muestran el mínimo, máximo y promedio de los individuos de la especie para cada generación. En el caso de los carnívoros, se detectó una rápida mejora en las primeras generaciones y luego una mejora lenta si se observa el máximo o estancamiento si se observa el promedio. Respecto a los herbívoros no se notó una mejora a medida que pasan las generaciones, lo que sugiere que el escenario y el entorno es lo suficientemente favorable para los herbívoros como para que comportamientos simples obtengan suficiente fitness. Comparando con el algoritmo greedy, los carnívoros lograron superarlo considerando el máximo, mientras que los herbívoros se encontraron siempre por debajo. Que los herbívoros se encuentren siempre por debajo del algoritmo greedy se puede explicar por el hecho de que no parecen haber mejorado sus comportamientos a lo largo de las generaciones y por lo tanto realizan un comportamiento muy simple.

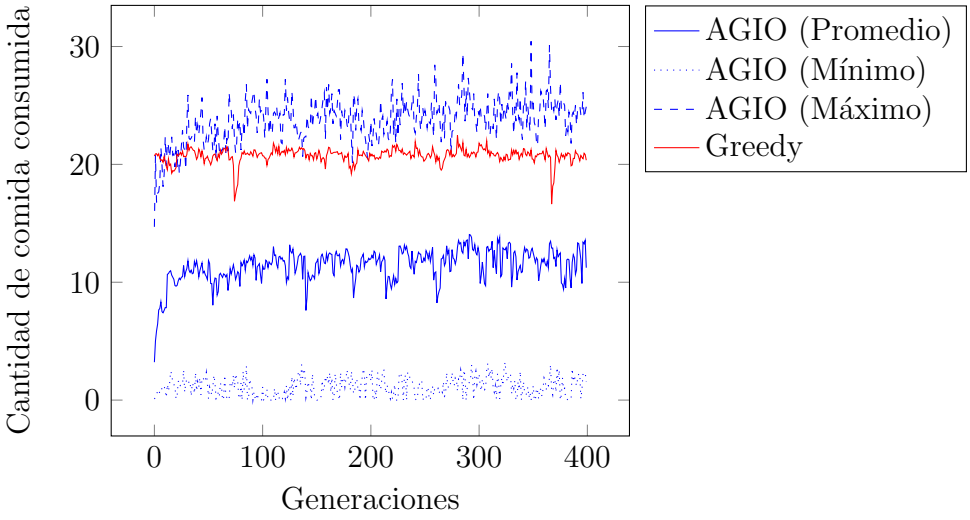


Figura 5.2: Cantidad de comida consumida por los carnívoros a lo largo de las generaciones.

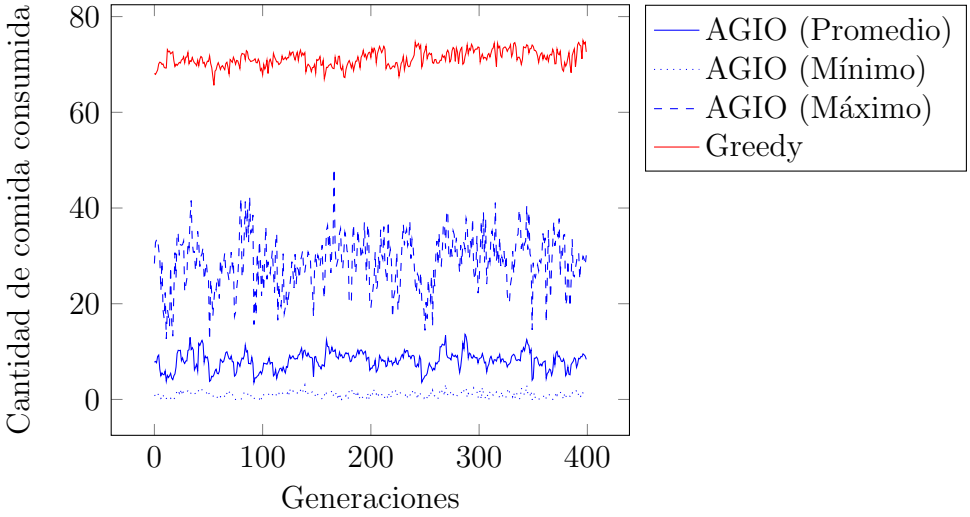


Figura 5.3: Cantidad de comida consumida por los herbívoros a lo largo de las generaciones.

El análisis de la proporción de acciones fallidas (figura 5.4 los carnívoros y 5.5 para los herbívoros) permite entender mejor los resultados obtenidos en cuanto a la cantidad de comida consumida. El rápido incremento en la cantidad de organismos consumidos por los carnívoros se correspondió con un rápido decremento en la proporción de acciones fallidas y el poco incremento para los herbívoros se explica con una variación pequeña en la proporción de acciones fallidas. La proporción de acciones fallidas permite tener una idea de qué tanto los organismos aprendieron el significado de la acción de comer,

porque mide si realizan la acción de comer en el momento en el que es válida o no.

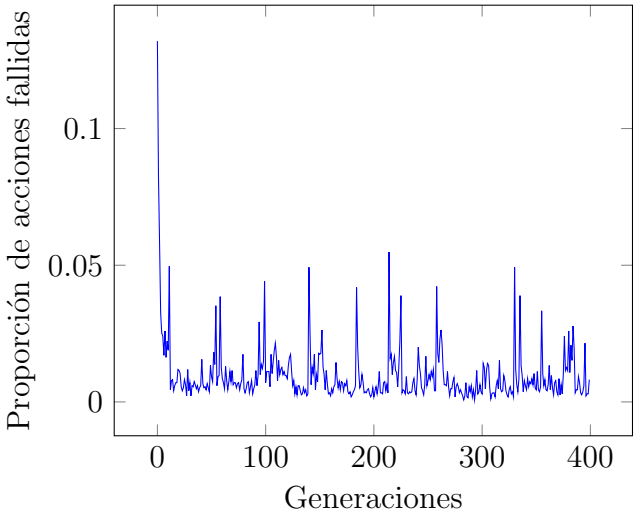


Figura 5.4: Proporción de acciones fallidas por los carnívoros a lo largo de las generaciones.

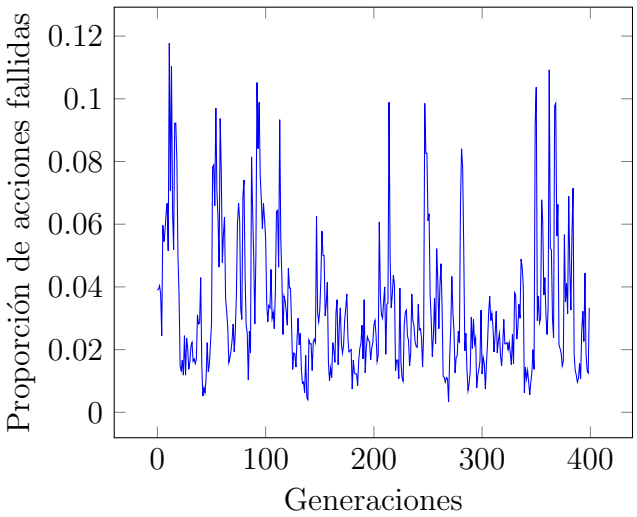


Figura 5.5: Proporción de acciones fallidas por los herbívoros a lo largo de las generaciones.

La última métrica analizada es la cantidad de celdas únicas recorridas en cada simulación por los individuos, promediando a lo largo de cada simulación realizada. Estos resultados se presentan en la figura 5.6 para los carnívoros y 5.7 para los herbívoros.

Los resultados son consistentes con los obtenidos analizando la cantidad de comida consumida y la proporción de acciones fallidas, encontrándose un

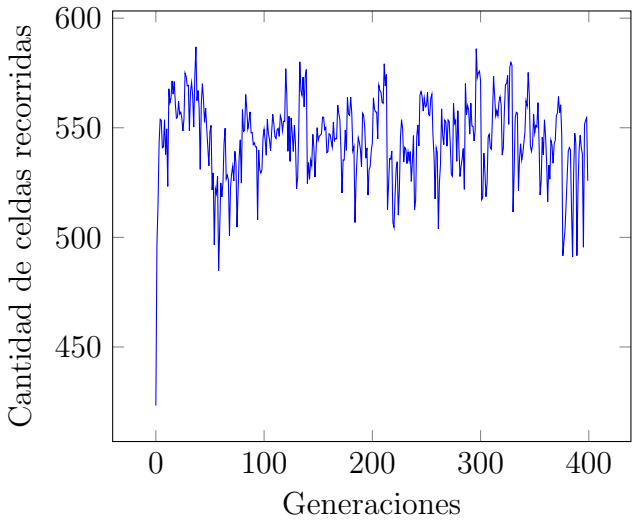


Figura 5.6: Cantidad de celdas únicas recorridas por los carnívoros en cada simulación promediando a lo largo de las generaciones.

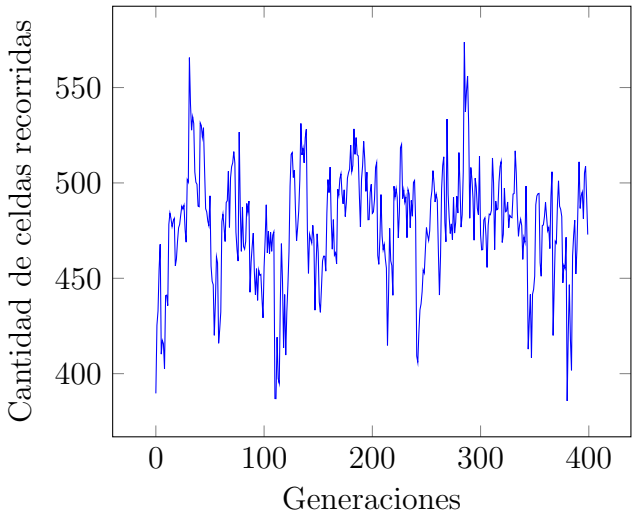


Figura 5.7: Cantidad de celdas únicas recorridas por los herbívoros en cada simulación promediando a lo largo de las generaciones.

rápido incremento y luego una estabilización en la cantidad de celdas recorridas en los carnívoros y gran variedad en los herbívoros. En promedio, los carnívoros recorrieron más celdas que los herbívoros, lo que resulta razonable teniendo en cuenta que tienen que seguir a los herbívoros para poder alimentarse de ellos. El entorno considerado en este experimento es de 40 por 40 celdas (1600 celdas en total), por lo que en promedio los carnívoros recorrieron el 34 % de las celdas del entorno, mientras que los herbívoros recorrieron un 30 %.

En conclusión, los resultados para los carnívoros fueron muy buenos, con

una clara mejora durante las primeras generaciones en el comportamiento, medido por la cantidad de comida consumida y la proporción de acciones fallidas. Si se consideran los mejores carnívoros, estos fueron capaces de obtener mejor fitness que los controlados por el algoritmo greedy. Para los herbívoros los resultados no fueron tan buenos, no observándose una mejora significativa con el paso de las generaciones. Esto pudo deberse a que la mejora de los carnívoros hizo más compleja la evolución de los herbívoros, o más probablemente, como se dijo anteriormente, porque la complejidad requerida del comportamiento de los herbívoros es suficientemente baja, dado que la comida no se mueve y es abundante. Si un comportamiento simple permite obtener un buen valor de fitness, es poco probable que surjan durante el proceso evolutivo individuos con comportamientos complejos que resulten en mejor fitness y por lo tanto la especie se va a estancar en individuos similares y simples.

5.2.2. Interrelaciones entre especies

Como se describe en la subsección 4.3.2, para analizar las posibles relaciones entre las especies, se asume primero que el fitness de una especie en distintas simulaciones se puede modelar como una variable aleatoria con distribución normal. Para verificar esta suposición se realizaron 100 pruebas con los individuos obtenidos del escenario complejo, donde cada prueba consistió en realizar 30 simulaciones y promediar el fitness de los individuos de cada especie por separado. De esta manera se generó una lista de 100 valores para cada especie y a estos valores se les aplicó el test de Shapiro - Wilk [30] y el test de D'Agostino - Pearson [27], utilizando la biblioteca de Python *SciPy*. Se considero un nivel de significancia de 0.05 para las dos pruebas. La tabla 5.4 reporta el valor p de los tests para cada especie. Las especies 0, 2, 3, 15 y 16, marcadas en rojo, fallaron los tests porque no evolucionaron correctamente, reportando 0 como fitness para todas las simulaciones, por lo que fueron ignoradas para el análisis de interrelaciones (no se reporta el valor p del test de Shapiro - Wilk porque su estimador no se encuentra definido cuando todas las muestras son iguales). El resto de especies logró pasar los dos tests. Si bien estrictamente eso significa que no se puede descartar la hipótesis nula (que la distribución es normal) y no que la hipótesis nula es verdadera, se asumió que lo es y se trabajó con el fitness como si siguiera una distribución normal.

La tabla 5.5 presenta los pares de especies que se consideran relacionados

Especie	Valor p	
	Shapiro - Wilk	D'Agostino - Pearson
0	-	0.00
1	0.28	0.52
2	-	0.00
3	-	0.00
4	0.33	0.89
5	-	0.00
6	-	0.00
7	0.51	0.38
8	0.07	0.38
9	0.40	0.42
10	0.24	0.36
11	0.70	0.79
12	0.70	0.93
13	0.16	0.56
14	0.57	0.54
15	-	0.00
16	-	0.00
17	0.77	0.86
18	0.66	0.46

Tabla 5.4: Tests de normalidad para la fitness de las especies. En rojo se marcan las especies que como fallaron en evolucionar generan siempre una fitness de 0 en todos los pasos de la simulación.

en base al criterio estadístico presentado en la subsección 4.3.2, reportando los tipos de especies y los resultados obtenidos, que son la distancia a la media en múltiplos de la desviación estándar, la probabilidad de obtener un valor igual o más lejano al obtenido (o sea, que el resultado es un error estadístico y en realidad las especies no se encuentran relacionadas) y la diferencia con la media, expresada en porcentaje respecto a la media.

De las 18 especies encontradas, 10 (55.5 %) se encuentran relacionadas con otras. No se observan relaciones simétricas (A con B y B con A), aunque sí existen cadenas de dependencias, como es el caso de la especie 4, una especie de omnívoros que depende de la especie 12, una especie de carnívoros, que a su vez depende de la especie de herbívoros 5 (una de las especies que falló en evolucionar).

La figura 5.8 muestra el grafo de relaciones entre especies, generado a partir de la tabla 5.5, donde los herbívoros se representan con elipses, los carnívoros con rectángulos y los omnívoros con rombos. En rojo se encuentran las espe-

Especie 1		Especie 2		Estadísticas		
ID	Tipo	ID	Tipo	Distancia en σ	Probabilidad	Fitness
1	Herbívoro	6	Carnívoro	2.39	0.0169	-19.83 %
4	Omnívoro	8	Omnívoro	3.98	0.0001	-59.89 %
4	Omnívoro	12	Carnívoro	2.18	0.0293	-32.80 %
8	Omnívoro	5	Herbívoro	2.13	0.0335	-45.39 %
8	Omnívoro	7	Omnívoro	2.30	0.0213	49.19 %
9	Omnívoro	7	Omnívoro	2.17	0.0301	47.03 %
10	Omnívoro	7	Omnívoro	2.06	0.0394	44.41 %
12	Carnívoro	5	Herbívoro	2.02	0.0433	-40.32 %
13	Herbívoro	0	Omnívoro	2.66	0.0079	-37.58 %
17	Herbívoro	5	Herbívoro	2.11	0.0346	-35.92 %

Tabla 5.5: Interrelaciones entre especies.

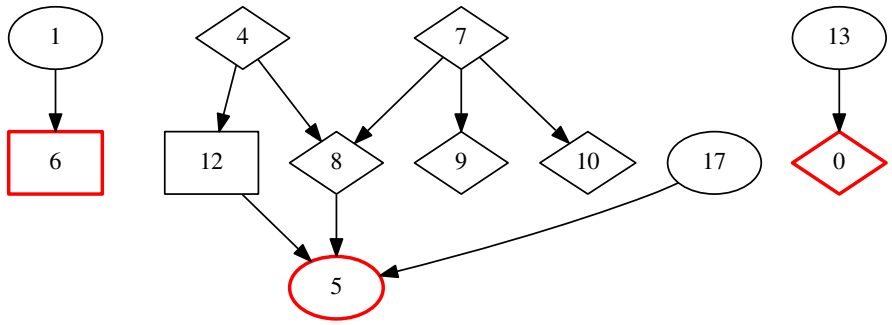


Figura 5.8: Grafo de relaciones entre las distintas especies a partir de la tabla 5.5. Las elipses son herbívoros, los rectángulos carnívoros y los rombos omnívoros.

cies que fallaron en evolucionar. El grafo resultante es dirigido, y la dirección está dada por el signo del cambio de fitness: $A \to B$ implica que al retirar la especie B, baja el fitness de la especie A. Este tipo de dependencia da la idea de que A se alimenta de B, como es el caso de la especie 7, que se alimenta de las especies 8, 9 y 10. Sin embargo, la dependencia $A \to B$ no siempre implica que A se alimenta de B, porque retirar a la especie 6, una especie de carnívoros, afecta negativamente a la especie 1, una especie de herbívoros. Que la ausencia de una especie de carnívoros que falló en evolucionar cause un decremento del 20 % en la fitness de la especie de herbívoros 1 da la idea de que existen relaciones más sutiles y difíciles de detectar que simplemente relaciones de cadenas alimenticias.

En conclusión, los resultados del estudio mostraron que se logró el objetivo buscado de evolucionar especies con comportamientos interrelacionados, incluso formándose cadenas de dependencias visibles y significativas.

5.2.3. AGIO vs Jugador

El jugador participante en las pruebas fue un joven de 17 años ajeno al proyecto, para evitar sesgos causados por conocer el funcionamiento interno del escenario sobre el que se desarrolló la prueba. La figura 5.9 muestra el fitness del organismo controlado por el jugador y el de los restantes organismos de la especie controlados por AGIO. Sin darle al jugador ninguna descripción de que son los sensores y las acciones, o sea, mostrando simplemente los valores numéricos, el jugador no fue capaz de superar el fitness obtenido por los individuos evolucionados por AGIO. El fitness obtenido por el mejor individuo de la especie fue 656, mientras que el obtenido por el jugador fue de 595.

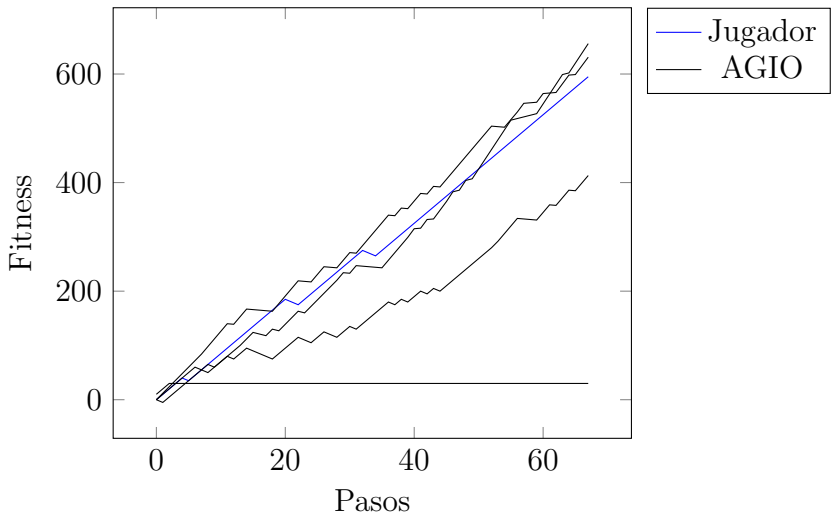


Figura 5.9: Fitness de distintos individuos de la misma especie que la del individuo controlado por el jugador comparados contra el fitness del individuo del jugador, sin mostrarle las descripciones.

La figura 5.10 presenta las mismas métricas que en la figura 5.9, pero cuando al jugador se le otorga una breve descripción en texto de las acciones, los sensores y los tipos de celdas. Los resultados muestran como darle información al jugador del significado de lo que se le muestra en pantalla ayuda a mejorar su desempeño, logrando superar a los organismos controlados por AGIO con un fitness de 1056 en el último paso de la simulación contra 953 de AGIO.

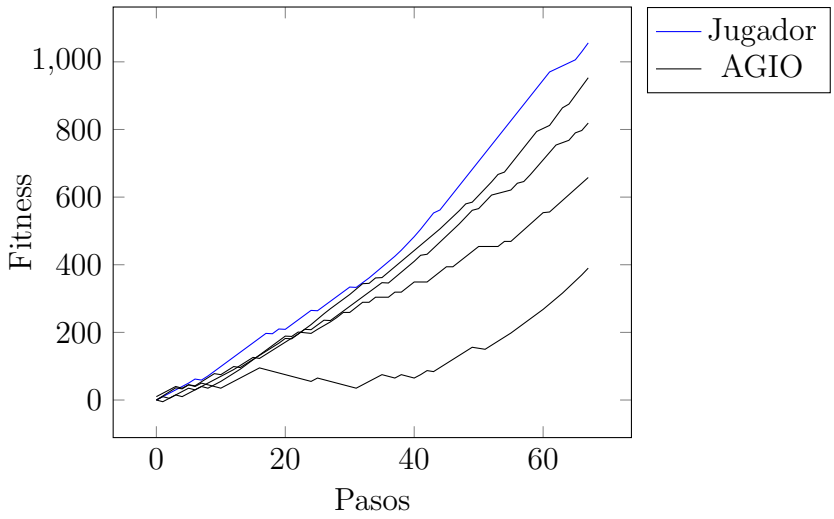


Figura 5.10: Fitness de distintos individuos de la misma especie que la del individuo controlado por el jugador comparados contra el fitness del individuo del jugador, mostrando las descripciones de las acciones, los sensores y los tipos de celda.

Los resultados indican que cuando el jugador se encuentra bajo las mismas condiciones que los organismos evolucionados por AGIO, obtiene un rendimiento similar al de los organismos controlados por las redes neuronales evolucionadas. Que el rendimiento sea similar muestra que AGIO es capaz de encontrar individuos con comportamientos adecuados al escenario sobre el que se evoluciona.

5.2.4. Análisis del efecto del tamaño de simulación

La gráfica de la figura 5.11 reporta los resultados obtenidos en cuanto a la cantidad de comida consumida por los organismos de las dos especies del escenario simple cuando se varía el tamaño de simulación como se describe en la sección 4.3.4, tomando el promedio de todos los organismos para la última generación. El efecto que tiene el tamaño de la simulación en la cantidad de comida consumida, particularmente para los carnívoros, se destaca claramente. Los carnívoros pasan de consumir 6.5 organismos en promedio a consumir 51.4, un incremento del 790 %. Este aumento se puede explicar porque como el tamaño del mundo es fijo, al tener más individuos en la simulación es más probable encontrar un individuo lo suficientemente cerca para comerlo. Respecto a los herbívoros, la cantidad de comida consumida varía notoriamente con los distintos tamaños de simulación, teniendo un máximo de 32 para un tamaño

de simulación de 15 y un mínimo de 5.4 para un tamaño de simulación de 45.

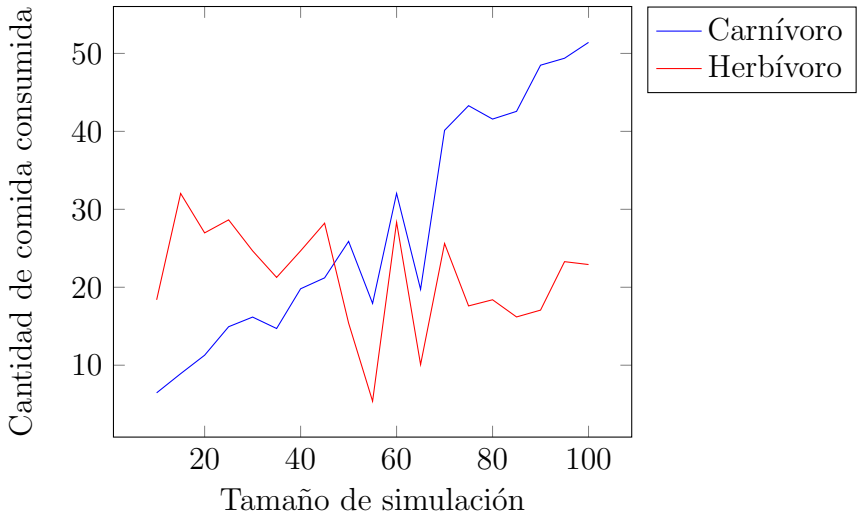


Figura 5.11: Cantidad de comida consumida promedio para la última generación de cada especie respecto al tamaño de simulación.

Los resultados obtenidos en el experimento confirman la hipótesis de que la cantidad de individuos simulados simultáneamente afecta significativamente el comportamiento de los individuos y por lo tanto la simulación en bloques de tamaño fijo y definido por el desarrollador es una consideración necesaria.

5.2.5. Análisis del tiempo de ejecución y del uso de memoria

Como se detalla en la subsección 4.3.5, se analizó el tiempo promedio de evolución en relación al número de generaciones, el tiempo promedio que lleva ejecutar una red neuronal evolucionada para obtener una decisión, el uso de memoria de AGIO durante el proceso evolutivo y el uso de memoria de los individuos evolucionados.

El tiempo de decisión promedio resultó ser 0.062 ms, con una desviación estándar de 0.029 ms. Utilizando la desigualdad de Chebyshev, se puede estimar que en un 75 % de los casos el tiempo de decisión estará en el intervalo 0.062 ± 0.058, que se encuentra dentro de el límite definido de 0.25 ms.

La cantidad de generaciones y el tamaño de la población afectan directamente el tiempo de evolución, por lo que se analizó como se comporta el tiempo de evolución respecto a esos dos parámetros. Los resultados obtenidos para el

tiempo de evolución variando la cantidad de generaciones, para una población de 600 individuos, se reportan en el gráfico de la figura 5.12. El resultado fue el esperado, obserbandose una dependencia lineal entre el tiempo de evolución y la cantidad de generaciones. Se verificó que el tiempo de evolución es menor a 5 minutos para todos los distintos números de generaciones probadas.

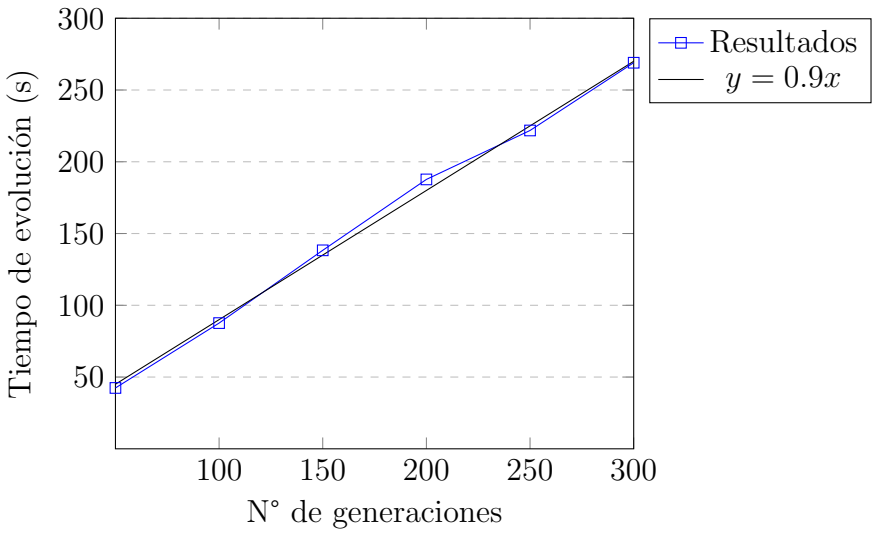


Figura 5.12: Tiempo total de evolución en relación a la cantidad de generaciones. Se muestran los resultados obtenidos y una regresión lineal como comparación, con $r^2 = 0.9974$.

El tamaño de la población debería tener una relación lineal con el tiempo de evolución, al igual que la cantidad de generaciones. El gráfico de la figura 5.13 presenta el tiempo de evolución promediado entre varias ejecuciones en base al tamaño de población, para 400 generaciones, como se detalla en la subsección 4.3.5. Los resultados confirman la suposición de dependencia lineal entre el tamaño de la población y el tiempo de evolución. Al igual que en el análisis de la cantidad de generaciones, para el tamaño máximo de población el tiempo de evolución es menor a 5 minutos, que fue es el límite establecido en la subsección 4.3.5 para una configuración rápida.

Si bien tanto el tamaño de la población como la cantidad de generaciones afectan linealmente el tiempo de evolución, una regresión lineal muestra que la pendiente del tiempo de evolución respecto al número de generaciones es dos veces más grande que la pendiente del tiempo de evolución respecto al tamaño de población, por lo que la cantidad de generaciones afecta en mayor medida al tiempo de evolución que el tamaño de la población.

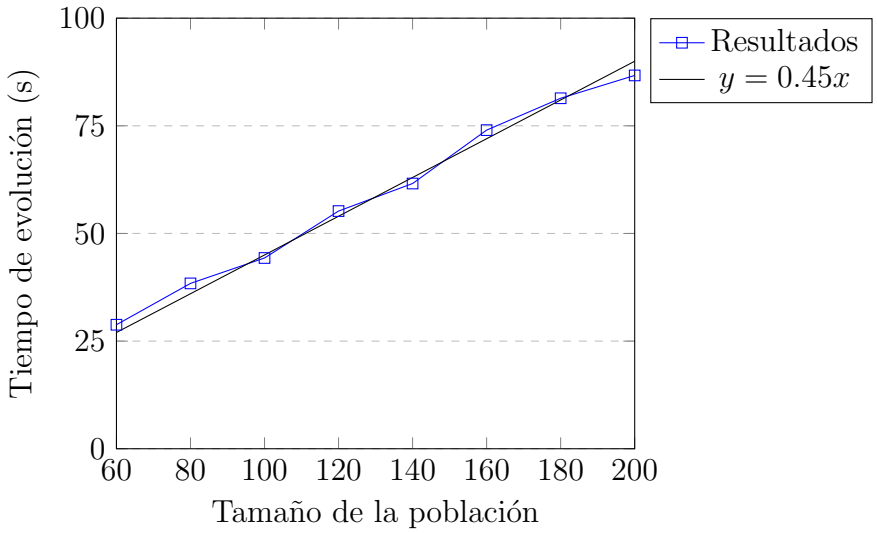


Figura 5.13: Tiempo total de evolución en relación al tamaño de la población. Se muestran los resultados obtenidos y una regresión lineal como comparación, con $r^2 = 0.9909$.

Se concluye, en base a los resultados obtenidos en el análisis del tiempo de evolución y de decisión que la implementación realizada se encuentra dentro de los límites establecidos en la subsección 4.3.5 y por lo tanto es aplicable en cuanto a tiempos en un proyecto de un videojuego AAA.

Respecto al uso de disco del archivo que almacena la información de los individuos luego de terminada la evolución, en ninguno de todos los experimentos realizados el tamaño de este archivo superó 1 MB. Considerando que el límite propuesto en la subsección 4.3.5 es de 100 MB, el uso de disco de la implementación realizada se encuentra muy por debajo de ese límite.

Para medir el uso de memoria del sistema, se registró la memoria utilizada por el proceso del sistema operativo en el que AGIO ejecutaba durante el proceso de evolución. Una vez que terminó el proceso y se obtuvieron los individuos evolucionados finales, se midió la memoria ocupada por cada individuo. Como no se detectaron diferencias significativas en el uso de memoria entre un mundo y otro en una prueba preliminar con pocas generaciones, se utilizó solo el mundo 0, con la finalidad de reducir el tiempo utilizado en el análisis de uso de memoria.

La gráfica de la figura 5.14 reporta el uso de memoria para cada generación. El uso aumenta de manera lineal, un resultado esperable porque se almacena información en cada generación. El uso de memoria se encontró siempre por

debajo de los 800 MB, estando muy por debajo del límite objetivo.

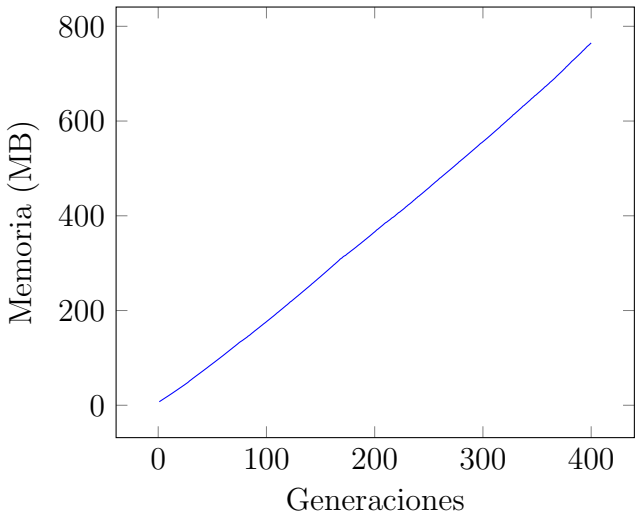


Figura 5.14: Uso de memoria a lo largo de las generaciones, en MB.

Por último, en la gráfica de la figura 5.15 se muestra un histograma de la memoria utilizada por los individuos después de 400 generaciones. La cantidad de bytes de memoria ocupados por los individuos varía entre cada uno, pero se encuentra entre 3 KB y 7 KB, con la mayoría entre 3 KB y 4 KB. Nuevamente, los resultados se encuentran dentro de los límites establecidos.

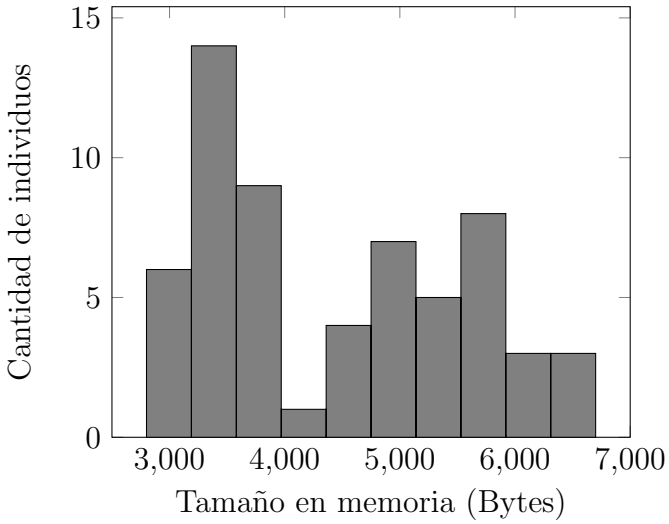


Figura 5.15: Histograma del tamaño promedio en memoria de los organismos después de 400 generaciones.

En conclusión, los resultados del estudio tanto de los tiempos de ejecución

como de evolución y el uso de memoria se encuentran dentro de lo esperado para que AGIO sea aplicable en proyectos fuera del ámbito académico. En todos los casos los resultados se encontraron dentro, y normalmente muy por debajo, de los límites establecidos en la subsección 4.3.5, que fueron pensados considerando los requerimientos de un juego AAA moderno.

Capítulo 6

Conclusiones y trabajo futuro

En este capítulo se plantean tanto conclusiones generales como posibles trabajos futuros, incluyendo mejoras al algoritmo y a la implementación, junto con posibles lineas de investigación para el futuro.

6.1. Conclusiones generales

La evaluación experimental mostró que el algoritmo presentado es capaz de generar diversas especies (en el orden de decenas) con relaciones entre ellas, formándose incluso cadenas de dependencias. El comportamiento de los individuos generados pudo competir favorablemente contra el comportamiento de un jugador humano, cuando la persona y el individuo tenían la misma información. Respecto a la implementación, el tiempo de evolución, de decisión y el uso de memoria se encontraron dentro de los rangos definidos pensando en la utilización de AGIO en el desarrollo de un videojuego AAA.

Este proyecto de grado presenta un algoritmo general para generar automáticamente diversos individuos con comportamientos interrelacionados, algo que no se había realizado anteriormente en la bibliografía investigada. Estos individuos son generados de manera automática, pero se permite al desarrollador que utiliza el algoritmo controlar como se pueden formar los individuos, restringiendo como se agrupan y seleccionan las partes que los componen, que pueden tener los individuos como entrada para decidir que acción ejecutar, y por último que es lo que hacen las acciones. El poder controlar la formación y el comportamiento de los individuos provee el aspecto de control de diseño que se encuentra faltante en los demás trabajos previos presentados, incrementando

la viabilidad de AGIO dentro de la industria de videojuegos.

6.2. Trabajo futuro

Si bien se lograron los resultados esperados, existen una variedad de mejoras y lineas de investigación sobre las que se puede continuar. Esta sección se divide en las mejoras de implementación, las mejoras al algoritmo y posibles lineas de investigación a futuro.

6.2.1. Mejoras de implementación

Un aspecto que afecta negativamente el tiempo de evolución de la implementación realizada es que la misma se realiza de manera serial, o sea, en un solo hilo de ejecución. Sin embargo, la simulación de la población, que para el experimento principal ocupa aproximadamente el 98 % del tiempo de evolución, consiste de múltiples batches independientes, que podrían ser paralelizados. Se propone duplicar el estado del entorno, teniendo una instancia por cada hilo en el que se quiera ejecutar el algoritmo, donde cada uno de estos hilos evaluara batches, con sus respectivas simulaciones, hasta que se todos los batches hayan sido simulados. Dado que no hay necesidad de sincronización entre los distintos hilos, la mejora obtenida debería ser aproximadamente lineal a la cantidad de núcleos de ejecución disponibles, mientras se tengan suficientes batches y suficiente memoria como para mantener simultáneamente el estado de cada hilo. Una vez realizada la paralelización, el siguiente paso es realizar una implementación nueva de la biblioteca de NEAT utilizada. La biblioteca utilizada de NEAT está escrita en un estilo de programación que no aprovecha todas las ventajas de las versiones modernas de C++, dependiendo mucho de la memoria dinámica y de indirecciones de memoria que incrementan el tiempo de ejecución. Además, no todas las funcionalidades de la biblioteca son utilizadas por AGIO, por lo que una reescritura podría reducir el tamaño del código y el uso de memoria.

6.2.2. Mejoras al algoritmo

Como se describe en el capítulo 3, las sub-poblaciones de NEAT utilizadas son independientes y tienen rangos de fitness potencialmente muy distintos entre ellas. Por lo tanto, es de esperarse que se puedan obtener mejores resultados

si se utilizan distintos valores para los parámetros en cada sub-población, ya sea seleccionando antes de la ejecución del algoritmo diferentes parámetros para distintos rangos de varianza y/o media de fitness (u otra métrica que pueda diferenciar las especies) o ajustando dinámicamente los parámetros. Respecto a la segunda opción, Aguston Eiben, Zbigniew Michalewicz, Marc Schoenauer y Jim Smith presentan en su articulo "Parameter control in evolutionary algorithms" [1] un resumen de los diversos mecanismos de ajuste de parámetros, incluyendo el ajuste durante la ejecución. Esa investigación puede servir de base para introducir esa capacidad en AGIO, no solo para las sub-poblaciones si no que para los parámetros de AGIO en si mismo.

6.2.3. Lineas de investigación para el futuro

El algoritmo presentado en este proyecto de grado realiza la evolución como un paso de pre-proceso, antes del programa donde se van a utilizar los individuos. Esto causa que una vez evolucionados los individuos, no puedan adaptarse a nuevos cambios en el entorno. Si se pudiera realizar la evolución en tiempo real, durante la misma simulación, se podría eliminar el tiempo de evolución y además permitir obtener individuos que respondan a cambios en el entorno, cambios que podrían ser causados por ejemplo por un jugador si se piensa en un videojuego. Esta linea de investigación fue tratada por los autores originales de NEAT en su artículo "Real-Time Neuroevolution in the NERO Video Game" [13], donde se presenta una modificación de NEAT para realizar la evolución en tiempo real. Las mismas ideas se podrían aplicar a AGIO para crear una versión en tiempo real. Una versión en tiempo real de AGIO tiene el atractivo práctico de reducir o eliminar los tiempos de evolución y de permitir nuevas interacciones entre jugadores e individuos. Adicionalmente, generar los individuos en tiempo real corresponde a una simulación de la vida más compleja y cercana a la naturaleza y por lo tanto AGIO podría utilizarse como una herramienta de investigación para el campo de la vida artificial.

Referencias bibliográficas

[1] Aguston Eiben, Zbigniew Michalewicz, M. S. J. S. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141.

[2] Bedau, M. (2003). Artificial life: organization, adaptation and complexity from the bottom up. *Trends in Cognitive Sciences*, 7(11):505–512.

[3] Ben Lorica, Mike Loukides (2018). What machine learning means for software development. Obtenido de *https://www.oreilly.com/ideas/what-machine-learning-means-for-software-development* el 28/03/2019.

[4] C. Srinivas, B. Ramgopal Reddy, K. R. R. N. (2014). Sensitivity analysis to determine the parameters of genetic algorithm for machine layout. *Procedia Materials Science*, 6:866–876.

[5] Daniel Jallov, Sebastian Risi, J. T. (2017). Evocommander: A novel game based on evolving and switching between artificial brains. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(2):181–191.

[6] Faustino Gomez, Jürgen Schmidhuber, R. M. (2008). Accelerated neural evolution through cooperatively coevolved synapses. *J. Mach. Learn. Res.*, 9:937–965.

[7] Gruau, F. (1997). *Neural Network Synthesis using Cellular Encoding and The Genetic Algorithm*. PhD thesis, L'universite Claude Bernard-Lyon.

[8] Joel Lehman, K. S. (2011a). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223.

[9] Joel Lehman, K. S. (2011b). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 211–218.

[10] Joel Lehman, K. S. (2011c). Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 211–218.

[11] Kalyanmoy Deb, Amrit Pratap, S. A. T. M. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

[12] Kenneth Stanley, R. M. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127.

[13] Kenneth Stanley, Bobby Bryant, R. M. (2005). Real-time neuroevolution in the nero video game. *IEEE Transactions on Evolutionary Computation*, 9(6):653–668.

[14] Krčah, P. (2007). Evolutionary development of robotic organisms. Master's thesis, Universidad Charles (Praga, República Checa).

[15] Langton, C. (1986). Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena*, 22(1):120–149.

[16] Langton, C. (1988). *Artificial Life*. Addison-Wesley.

[17] Leslie Kaelbling, Michael Littman, A. M. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4(1):237–285.

[18] Mahfoud, S. (1995). Niching methods for genetic algorithms. Master's thesis, Illinois Genetic Algorithms Laboratory (IlliGAL) Department of General Engineering University of Illinois at Urbana-Champaign.

[19] Marco Foco, Dmitry Korobchenko, A. E. (2017). Zoom, enhance, synthesize! magic upscaling and material synthesis using deep learning. Game Developers Conference 2017.

[20] Martin Mahner, M. K. (1997). What exactly are genomes, genotypes and phenotypes? and what about phenomes? *Journal of Theoretical Biology*, 186(1):55–63.

[21] Mayr, E. (2007). *What Makes Biology Unique?: Considerations on the Autonomy of a Scientific Discipline*. Cambridge University Press.

[22] Melanie Mitchell, S. F. (1994). Genetic algorithms and artificial life. *Artificial Life*, 1:267–289.

[23] Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. The MIT Press.

[24] Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

[25] Moriarty, D. (1997). *Symbiotic Evolution Of Neural Networks In Sequential Decision Tasks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin.

[26] Newzoo (2018). 2018 global games market report. Obtenido de *https://newzoo.com/insights/trend-reports* el 26/05/2019.

[27] Ralph D'Agostino, A. B. (1990). A suggestion for using powerful and informative tests of normality. *The American Statistician*, 44(4):316–321.

[28] Ray, T. (1991). An approach to the synthesis of life. In *Artificial Life II*, pages 371–408.

[29] Robert Wilson, F. K. (1999). *The MIT Encyclopedia of the Cognitive Sciences*. MIT Press.

[30] Samuel Shapiro, M. W. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611.

[31] Sebastian Echegaray, W. L. (2006). Simulation of animal behavior using neural networks. In *2006 IEEE Region 5 Conference*, pages 99–102.

[32] Sims, K. (1994). Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 15–22.

[33] Stanley, K. (2017). Neuroevolution: A different kind of deep learning. Obtenido de *https://www.oreilly.com/ideas/neuroevolution-a-different-kind-of-deep-learning* el 27/04/2019.

[34] Stephanie Forrest, Brenda Javornik, R. S. A. P. (1993). Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, 1(3):191–211.

[35] Theodore, S. (2018). Why have video game budgets skyrocketed in recent years. Obtenido de *www.quora.com/Why-have-video-game-budgets-skyrocketed-in-recent-years/answer/Steve-Theodore* el 11/01/2019.

[36] U.S. Department of Commerce (2017). 2017 top markets report media and entertainment sector snapshot. Obtenido de *https://www.trade.gov/topmarkets* el 26/05/2019.

Glosario

Mecánica de juego Regla de juego con una entrada y una salida que produce cambios en el sistema. 1

Sistema bottom-up Un sistema es diseñado de forma *bottom-up* cuando las partes individuales se diseñan con detalle y luego se enlazan para formar componentes más grandes, que a su vez se enlazan hasta que se forma el sistema completo. 4

Sistema top-down Un sistema es diseñado de forma *top-down* cuando se comienza su diseño con un resumen del sistema, sin especificar detalles. Iterativamente, cada parte del sistema se refina, diseñando con mayor detalle, hasta que la especificación completa es lo suficientemente detallada para validar el sistema. 4