

Grado Universitario de Ingeniería Informática

2021-2022

Trabajo Fin de Grado

**“Estudio y desarrollo del algoritmo
Neuroevolución de Topologías
Aumentadas en el juego Pac-man”**

Alejandro Hernández Artiles

Tutor/es

Jose Antonio Iglesias Martinez

Leganés, Octubre de 2022



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

RESUMEN

Las redes de neuronas artificiales son unos aproximadores de funciones universales muy populares actualmente, usadas en numerosos sistemas de Inteligencia Artificial. Una de sus particularidades es que para un mismo problema, la estructura de una red de neuronas puede cambiar drásticamente el resultado, por lo que se debe elegir cuidadosamente dicha estructura para optimizar la tarea a tratar.

Sin embargo, dicha estructura adecuada se suele buscar mediante una heurística de prueba y error, siendo un proceso costoso e inefficiente.

En este trabajo se ha querido experimentar con métodos que usan Neuroevolución, un tipo de aprendizaje automático que utiliza algoritmos genéticos para realizar búsquedas sobre el espacio de estructuras de una red neuronal, con el objetivo de encontrar la que optimice la tarea determinada. En concreto, en este trabajo se estudiará el algoritmo de Neuroevolución de Topologías Aumentadas (NEAT).

Este algoritmo será analizado y probado en una abstracción del juego de Pac-man, usando enemigos cuyos movimientos siguen un patrón no determinista. El objetivo será estudiar el rendimiento y los límites del algoritmo, mientras se trata de crear un agente IA controlado por una red de neuronas óptima que resuelva la tarea propuesta.

Tras una experimentación exhaustiva con diferentes parámetros, funciones de evaluación y formatos de información proporcionada al agente, se obtuvo un rendimiento bajo del algoritmo en un entorno no determinista, con puntuaciones por debajo del 40 % debido al inmenso ruido que el no determinismo proyectaba sobre la función de evaluación.

Por el contrario, en un entorno determinista, el algoritmo mostró su potencial llegando a superar el 90 % de puntuación.

De esta manera, el análisis de NEAT mostró algunas de sus desventajas, pero también sus ventajas, como la optimización de la topología de la red y la obtención de una estructura mínima.

Palabras clave: Neuroevolución, NEAT, RNA, Algoritmo genético, topología, videojuego

AGRADECIMIENTOS

En primer lugar doy las gracias a mi padre, mi madre y hermano, por todo el inmenso esfuerzo que han hecho para que yo esté aquí hoy. Haré todo lo que haga falta para ser digno de ese sudor y pagarlos con intereses. También agradecer el apoyo y ánimo constante.

También al resto de mi familia, por mostrarme su apoyo y ayudarme cuando lo he necesitado.

Mención especial para el equipo de reforma de mi ordenador, mi tío Alex y mi tío Pepe, por ayudarme cuando más lo necesitaba y hacer que mi ordenador sobreviviera todo este trabajo.

A mis amigos de Canarias y Madrid por alegrarme y hacer divertido todo este trayecto universitario.

A mi tutor Jose Antonio por aconsejarme y preocuparse por mí durante la duración de este estudio.

A Rita por ayudarme siempre a elegir, dentro del conjunto infinito de posibles caminos, el mejor.

ÍNDICE GENERAL

1. INTRODUCCIÓN	1
1.1. Motivación	1
1.2. Descripción del problema	2
1.3. Objetivos	3
1.4. Organización de la memoria	4
2. ESTADO DEL ARTE	5
2.1. Aprendizaje Automático	5
2.1.1. Aprendizaje Supervisado	6
2.1.2. Aprendizaje no Supervisado	6
2.1.3. Aprendizaje por refuerzo	6
2.2. Redes de Neuronas Artificiales	6
2.2.1. Introducción teórica	6
2.2.2. Usos de las RNAs	9
2.3. Algoritmos genéticos	10
2.3.1. Introducción teórica	10
2.3.2. Usos de los AGs	12
2.4. Neuroevolución	13
2.4.1. Aspectos a evolucionar	13
2.4.2. Codificación	15
2.4.3. Usos de la Neuroevolución	16
2.5. Neuroevolución de Topologías Aumentadas	17
2.5.1. Codificación	18
2.5.2. Números de innovación	19
2.5.3. Especiación	20
2.5.4. Incremento de una estructura mínima	22
2.5.5. Rendimiento en las pruebas	22
2.5.6. Usos de NEAT	23
2.6. Estudios sobre Pac-man	24

3. ANÁLISIS Y DISEÑO DEL SISTEMA	26
3.1. Requisitos del sistema	26
3.1.1. Requisitos funcionales	26
3.1.2. Requisitos no funcionales	28
3.1.3. Casos de uso	28
4. IMPLEMENTACIÓN DEL SISTEMA	32
4.1. Cambios en el código fuente	32
4.2. Entradas de la red	33
4.2.1. Entradas en formato de coordenadas	33
4.2.2. Entradas en formato distancia-pasillo	35
4.2.3. Entradas de apoyo	37
4.3. Salidas de la red	38
4.4. Función de evaluación	40
4.4.1. Score y tiempo de vida	40
4.4.2. Score	41
4.4.3. Media del individuo	42
4.5. Hiperparámetros del algoritmo	43
4.6. Flujo del sistema	44
5. EXPERIMENTACIÓN	46
5.1. Métricas	46
5.2. Experimentación con 8 fantasmas no deterministas	47
5.2.1. Conclusión de la experimentación con 8 fantasmas no deterministas	50
5.3. Experimentación progresiva con fantasmas no deterministas	53
5.3.1. Conclusión de la experimentación progresiva con fantasmas no deterministas	61
5.4. Experimentación con fantasmas deterministas	61
5.4.1. Conclusión de la experimentación con fantasmas deterministas	69
6. GESTIÓN DEL PROYECTO	70
6.1. Metodología	70
6.2. Planificación	70

6.3. Presupuesto	71
6.3.1. Costes de personal	71
6.3.2. Costes de material	72
6.3.3. Costes totales	73
6.4. Impacto socio-económico	74
6.5. Marco regulador	74
6.6. Gestión de riesgos	75
7. CONCLUSIONES Y TRABAJOS FUTUROS	77
7.1. Conclusiones	77
7.2. Trabajos Futuros	78
7.2.1. Mitigación de los efectos no deterministas a la función de evaluación	78
7.2.2. Variación de entorno	80
7.2.3. Variación del algoritmo	80
BIBLIOGRAFÍA	81

ÍNDICE DE FIGURAS

1.1	Implementación original de Azuelo	3
2.1	Ejemplo de RNA	8
2.2	Arquitectura de AlphaStar	10
2.3	Esquema básico de la población de un AG	11
2.4	Ejemplo de fitness landscape	12
2.5	Evolución de pesos contra Backpropagation (Montana, Davis)	14
2.6	Codificación de NEAT	19
2.7	Cruce de dos genotipos en NEAT	20
2.8	Dependencias de las propiedades de NEAT	23
3.1	Diagrama de casos de uso	31
4.1	Paso del mapa original al modificado	33
4.2	Numeración de los pasillos	36
4.3	Ejemplo de diagrama de red totalmente conectada sin neuronas ocultas . .	39
4.4	Diagrama de flujo del sistema	45
5.1	Diagrama cronológico de la experimentación	46
5.2	Evolución del fitness del modelo 5	49
5.3	Evolución del fitness del modelo 4	50
5.4	Evolución del fitness del modelo 7	50
5.5	Registro de las especies del modelo 4	51
5.6	Red ganadora del modelo 7	52
5.7	Evolución de fitness del modelo Dis-pas sin fantasmas	54
5.8	Evolución del fitness del modelo dist-pas con una fantasma	55
5.9	Evolución del fitness del modelo x-y con un fantasma	56
5.10	Evolución del fitness del modelo dis-pas con dos fantasmas	56
5.11	Evolución del fitness del modelo x-y con dos fantasmas	57
5.12	Red ganadora del modelo x-y con dos fantasmas	58

5.13 Evolución del fitness del modelo x-y con dos fantasmas y 300 generaciones extra	59
5.14 Red ganadora del modelo x-y con dos fantasmas y 300 generaciones más	60
5.15 Evolución del fitness del modelo x-y con dos fantasmas deterministas	62
5.16 Evolución del fitness del modelo dis-pas-det con cinco fantasmas deterministas	63
5.17 Evolución del fitness del modelo dis-pas-det con cinco fantasmas deterministas	64
5.18 Red ganadora del modelo x-y-det con cinco fantasmas	65
5.19 Evolución del fitness del modelo x-y-det-wmi con ocho fantasmas deterministas	67
5.20 Red ganadora del modelo x-y-det-wmi con 8 fantasmas	68
6.1 Metodología del proyecto en cascada	70
6.2 Diagrama de Gantt del estudio	71

ÍNDICE DE TABLAS

3.1	Formato de requisitos	26
3.2	Requisitos funcionales	27
3.3	Requisitos no funcionales	28
3.4	Formato caso de uso	29
3.5	Caso de uso - CU - 01	29
3.6	Caso de uso - CU - 02	29
3.7	Caso de uso - CU - 03	30
3.8	Caso de uso - CU - 04	30
5.1	Tabla de modelos de la experimentación con 8 fantasmas no deterministas	48
5.2	Tabla de resultados de la experimentación con 8 fantasmas no deterministas	49
5.3	Tabla de modelos de la experimentación progresiva con fantasmas no deterministas	53
5.4	Tabla de resultados de la experimentación progresiva con fantasmas no deterministas	54
5.5	Tabla de modelos de la experimentación con fantasmas deterministas . . .	62
5.6	Tabla de resultados de la experimentación del modelo x-y-det y x-y-det-wmi con 8 fantasmas deterministas	66
6.1	Planificación del proyecto	71
6.2	Costes de personal	72
6.3	Costes de material	73
6.4	Costes Totales	73
6.5	Gestión de riesgos	76

1. INTRODUCCIÓN

Si se reiniciara la civilización de nuevo, desde la aparición de las primeros homínidos, ¿se llegaría en algún momento al estado actual, con la misma tecnología y comprensión del universo? ¿Sería Newton quien describiera las leyes gravitatorias y Einstein quien elaborara la teoría relativista? ¿Se hubiera llegado siquiera a las matemáticas como abstracción más completa de lo que nos rodea? No se sabrá si una segunda simulación hubiera convergido en el mismo punto de la historia actual, o hubiera encontrado diferentes soluciones, mejores o peores. Extendiendo la idea al infinito, y por consiguiente, infinitos reinicios de la civilización, cuesta imaginar si nuestro rumbo como sociedad es el único o uno de infinitos, dirigido por una función de evaluación desconocida.

Cómo el ser humano aprende y encuentra soluciones a los diversos problemas del entorno sigue siendo un tema capital en la neurociencia, ya sea para problemas cotidianos o más complejos, como tratar de entender nuestro propio funcionamiento o el del universo.

1.1. Motivación

En el campo de las ciencias informáticas se ha tratado humildemente de imitar el proceso de aprendizaje de diversas formas. Una de ellas es inspirándose en las estructuras naturales que lo permiten, como el cerebro o la genética. Es lo que se conoce como Inteligencia Artificial de Inspiración Biológica (Floreano y Mattiussi, 2008). Dentro de estos métodos existen dos paradigmas principales la genética y las redes neuronales artificiales (RNA) (Kubat, 1999). Tanto los algoritmos genéticos como las RNA son técnicas que actualmente tienen mucha utilidad en diversos campos. En el caso de estas últimas, su capacidad para resolver problemas no lineales y su tolerancia al ruido las hace una técnica muy poderosa para aprender cierto tipo de tareas o patrones. Por ejemplo, han sido utilizadas para meteorología (Maqsood et al., 2004) o para clasificación de imágenes (Sultana et al., 2018) entre muchas otras.

Sin embargo, uno de los problemas de las RNA es que para poder resolver un problema con eficacia, su diseño debe ser el adecuado, es decir, para un problema específico, el diseño de la red puede variar drásticamente el resultado. Por tanto, se debe determinar los parámetros de la red, como el tipo de red o su arquitectura (topología y pesos de las conexiones) lo mejor posible para obtener la solución más completa.

Normalmente, la elección de los parámetros se hace mediante prueba y error. Se prueba la red con diversas configuraciones y se obtiene para cada una de ellas un error de entrenamiento y un error de validación, que indicará qué red generaliza mejor para el problema dado. Este proceso es ineficiente y costoso. Aparte podría descartarse alguna configuración de parámetros adecuada debido al enorme espacio de soluciones que

comprende todas las posibles combinaciones de parámetros de una red (número de neuronas de entrada, número de neuronas de salida, número de capas ocultas, qué cantidad de neuronas ocultas habrá por cada capa oculta, qué conexiones habrá, pesos de dichas conexiones, etc.).

Uno de los métodos más interesantes, debido a su alta inspiración biológica, para tratar de abordar esta desventaja es la Neuroevolución (Floreano et al., 2008). Esta forma de inteligencia artificial usa algoritmos genéticos (AG) para generar distintas RNAs con diferentes configuraciones, hasta encontrar la que mejor se adapte a la tarea propuesta bajo la medida de una función de evaluación.

En este estudio se ha elegido mayoritariamente como objeto de las investigaciones el algoritmo de Neuroevolución de topologías aumentadas (K. O. Stanley y Miikkulainen, 2002), conocido como NEAT por sus siglas en inglés de "NeuroEvolution of Augmenting Topologies". Esta decisión ha sido tomada debido a que es un algoritmo paradigmático que puede servir para entender la mayoría de conceptos propios de la Neuroevolución, además de contar con interesantes particularidades y buenos resultados que lo hacen uno de los métodos más famosos en este campo.

1.2. Descripción del problema

Para poder probar el funcionamiento de NEAT se ha elegido utilizar una abstracción del famoso juego Pac-man. Se refiere a abstracción porque se usarán diversos conceptos del juego original, pero modificados para poder analizar otros comportamientos de NEAT. Las principales modificaciones hechas al juego original son:

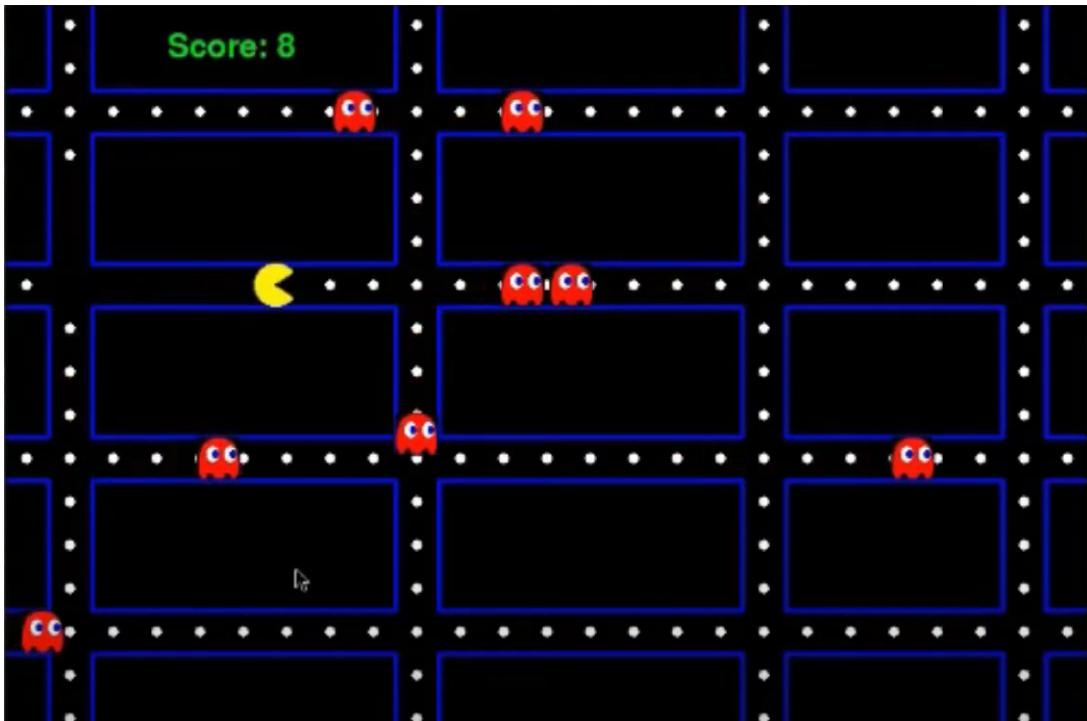
- Enemigos: En vez de 4 fantasmas habrá 8.
- Movimientos de los enemigos: Los movimientos de los fantasmas no serán deterministas, es decir, elegirán sus movimientos de manera aleatoria. Esto será un cambio crítico para el correcto aprendizaje del algoritmo, como se verá ulteriormente en el estudio.
- Muerte de los enemigos: No habrá Power-Pellets, por lo que Pac-man no puede de ninguna manera comerse a los fantasmas.

Como se aprecia observando los cambios en las reglas, el juego de Pac-man ha sido complejizado hasta convertirlo en una tarea de esquivar numerosos enemigos cuyo patrón de movimiento es impredecible. Esto hace que el algoritmo no pueda definir una ruta más óptima para esquivar los fantasmas, sino que deberá aprender un comportamiento más abstracto como es el de esquivar y esperar los movimientos de sus enemigos, a la vez que maximiza su puntuación. Por tanto, el objetivo de Pac-man es comerse la máxima cantidad de puntos del mapa sin ser comido por uno de los ocho fantasmas.

El código fuente de Pac-man que se ha utilizado de base para implementar el algoritmo y las normas es el subido al portal SourceCodeHero por Glenn Azuelo (Azuelo, 2022). En una futura sección se detallará los cambios efectuados y las características de esta implementación.

Figura 1.1

Mapa de la implementación original de Azuelo



Fuente: Azuelo, 2022

1.3. Objetivos

Una vez sintetizado el problema, el objetivo consistirá en implementar un sistema que pueda aprender a maximizar la puntuación de Pac-man sin entrar en contacto con los fantasmas.

Este sistema será un modelo de una RNA que habrá sido optimizado por el algoritmo NEAT, de manera que pueda desenvolverse adecuadamente en la tarea de Pac-man.

En este estudio se han propuesto 3 objetivos principales, que tratan, aparte de obtener un buen resultado en el juego modificado de Pac-man, ganar conocimiento y comprender las profundidades de NEAT y de la Neuroevolución en general. A continuación se listan dichos objetivos:

- Obtener una configuración de RNA a través del algoritmo NEAT adecuada para jugar a la abstracción de Pac-man propuesta.

- Analizar y mostrar la evolución del algoritmo y de las RNAs obtenidas durante la ejecución.
- Sacar conclusiones sobre la relación entre las configuraciones de las RNAs y su desempeño posterior en el juego.

1.4. Organización de la memoria

Para finalizar esta introducción al estudio, se dispone a describir brevemente los contenidos de los capítulos siguientes.

Con el objetivo de dar una visión general de los temas que se tratan en este trabajo, en el capítulo 2 se revisa la literatura disponible sobre Inteligencia Artificial, Neuroevolución, sus diferentes conceptos y sus usos.

En el capítulo 3 se describe el análisis y el diseño del sistema. En concreto, se propondrán un conjunto de requisitos que el sistema debe cumplir, además de sus casos de uso.

En el capítulo 4 se presenta la implementación utilizada para abordar el problema de Pac-man. Se tratarán diferentes cuestiones dentro del algoritmo como la función de evaluación, los hiperparámetros del algoritmo genético o cambios en el código fuente. En este capítulo se detallarán los diferentes procesos que se han llevado a cabo para tratar de cumplir el primer objetivo principal del estudio.

En el capítulo 5 se relata la experimentación con los diferentes diseños y sus resultados. Se estudiarán cuestiones como los efectos de las diferentes implementaciones del algoritmo. A su vez, se abordarán el segundo y tercer objetivo del estudio.

En el capítulo 6 se exponen los aspectos relativos a la gestión del proyecto. Específicamente, se explica la metodología usada y el desglose del presupuesto, se analiza el impacto socio-económico del trabajo y se sitúa el estudio en un marco regulador.

Finalmente, en el capítulo 7 se detallan las conclusiones generales del estudio y los posibles trabajos y modificaciones futuras.

2. ESTADO DEL ARTE

Históricamente la Inteligencia Artificial (IA) ha seguido cuatro caminos diferentes, que se han juntado y ayudado en diversas ocasiones. De estos caminos, dos miden el éxito de la IA en base a su fidelidad con el humano, y las otros dos miden el éxito en base a un ideal de racionalidad (Russell y Norvig, 2009). Para este estudio se tomará la perspectiva de un agente IA por aquel que actúa racionalmente. Se entiende que un agente es capaz de recibir información del entorno, procesar esta información, y luego actuar en base a ella.

Las IA cuentan con diversos métodos para obtener resultados que cumplan con los requisitos de la racionalidad. Por ejemplo, cuenta con el razonamiento lógico o con tener programadas respuestas determinadas para entradas específicas, sin embargo para la comprensión de este estudio se tendrán en cuenta dos de los métodos.

- Técnicas de búsqueda, específicamente los algoritmos genéticos, tratando de emular las reglas genéticas de la naturaleza.
- Aproximadores de funciones, concretamente las redes de neuronas artificiales, tratando de emular el funcionamiento del cerebro.

En este capítulo se detallan las investigaciones relevantes sobre los temas tratados en este estudio. En el apartado 2.1 se presenta el aprendizaje automático y sus distintos tipos. En el apartado 2.2 se explican en detalle las RNAs. En el 2.3 se habla sobre los AG, para en el 2.4 introducir y detallar las investigaciones en Neuroevolución. Posteriormente, en el apartado 2.5 se explicará la técnica neuroevolutiva implementada y analizada en este estudio (NEAT). Finalmente, se presentan trabajos e investigaciones relacionados con el juego Pac-man en el apartado 2.6.

2.1. Aprendizaje Automático

El aprendizaje automático es una disciplina de la Inteligencia Artificial que estudia el aprendizaje de las máquinas, es decir, investiga diferentes técnicas y métodos para que los computadoras sepan resolver un problema a través de datos del mismo.

Actualmente el aprendizaje automático es una herramienta muy utilizada en muchos y muy diversos campos, como en el de la medicina (Kononenko, 2001), la biología con el nuevo modelo Alphafold para predecir la estructura 3D de todas las proteínas (Ruff y Pappu, 2021), reconocimiento de posturas (Parvathy et al., 2021), reconocimiento facial (J. Chen y Jenkins, 2017), videojuegos (Skinner y Walmsley, 2019) o economía (Athey, 2018).

Los métodos de aprendizaje automático se pueden clasificar en tres tipos principales, dependiendo de su salida y la forma de aprender.

2.1.1. Aprendizaje Supervisado

El aprendizaje supervisado trata de aprender encontrando un modelo que relacione las entradas y las salidas, entrenando con ejemplos ya clasificados es decir, con una clase asignada a priori, para poder predecir la salida de ejemplos no vistos anteriormente. Por tanto, para aprender el algoritmo compara su salida actual con la clase del ejemplo, para luego realizar los cambios que sean necesarios (Cunningham et al., 2008).

Este es el tipo más usado de aprendizaje automático. Suele ser utilizado con RNAs, que modifican sus pesos en función del error al tratar de predecir la clase.

2.1.2. Aprendizaje no Supervisado

En el aprendizaje no supervisado, las entradas para entrenar no están clasificadas, por lo que el algoritmo no tiene conocimiento sobre lo que tiene que aprender a clasificar. Por tanto, detectará patrones dentro del conjunto de ejemplos, clasificando él mismo las diferentes entradas. Internamente intenta encontrar redundancias y rasgos significativos para agrupar los datos.

Se suelen utilizar algoritmos como K-medias (Ahmed et al., 2020) o también RNAs (Flinton y Sejnowski, 1999).

2.1.3. Aprendizaje por refuerzo

En el aprendizaje por refuerzo el algoritmo aprende recibiendo información del entorno. Las entradas que recibe es la retroalimentación que obtiene del entorno al ejecutar una determinada acción. Por ende, aprende mediante ensayo y error, actuando y esperando la respuesta positiva o negativa tras esa acción.

El propósito del algoritmo es encontrar una función que maximice las respuestas positivas o recompensas. Es utilizado por ejemplo en agentes de ajedrez u otro tipo de juegos (Littman, 1994).

2.2. Redes de Neuronas Artificiales

2.2.1. Introducción teórica

Una red de neuronas artificial trata de emular el comportamiento del cerebro humano mediante la abstracción de las neuronas y sus conexiones a un ámbito computacional.

Las neuronas biológicas poseen tres partes:

- Las dendritas, que funcionan como las entradas de información a la neurona.
- El cuerpo de la neurona, que procesa la información que llega de las dendritas.
- El axón que lleva la salida de la neurona. Está conectado a las dendritas de otras neuronas.

Cómo funciona exactamente el cerebro sigue siendo un tema desconocido, pero en rasgos generales, una neurona transmite su salida a otra a través de su axón. La neurona que recibe la información recoge todas las salidas de las neuronas conectadas a ella, para luego procesarla, sumando las salidas excitadoras (positivas) e inhibidoras (negativas). Finalmente, envía la señal resultante, excitadora o inhibidora, a través de su axón. Además, se conoce que el cerebro cuenta con 10^{11} neuronas y cada una de ellas está conectada aproximadamente con otras 10.000. Actualmente es imposible crear un sistema con un tamaño y conectividad similar, sin embargo, las redes de neuronas artificiales, a pesar de no llegar al poder computacional del cerebro, son realmente útiles para diversas tareas.

Una RNA consiste en un sistema interconectado de neuronas artificiales. Una neurona artificial cuenta con un conjunto de entradas y de salidas, y un estado, llamado nivel de activación. El estado puede ser cualquier valor dentro de un conjunto de valores posibles. Por ejemplo, el conjunto de estados S podría ser $S = \{0,1\}$, o un conjunto con más valores $S = \{1,2,3,4,\dots,n\}$, o un intervalo continuo $S = [0, 1]$.

El nivel de activación depende del valor de las entradas y de los correspondientes pesos de las conexiones de las entradas a la neurona. Cada valor de entrada se multiplica por su respectivo peso, siendo el nivel de activación la suma de todos los resultados. Es decir, siendo un conjunto de entradas $x_1, x_2, x_3, \dots, x_n$ representadas por un vector X, y un conjunto de pesos $w_1, w_2, w_3, \dots, w_n$ representadas por un vector W, el nivel de activación vendría dado por:

$$E = \sum_{i=1}^n x_i w_i \quad (2.1)$$

O lo que es lo mismo:

$$E = X^T W \quad (2.2)$$

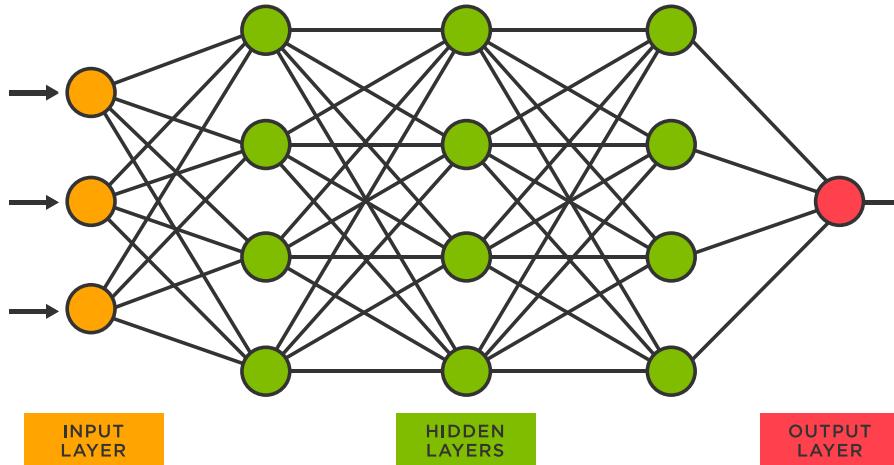
Finalmente, para calcular la salida de la neurona, el nivel de activación E pasa por una función de activación, que suele servir para acotar la salida de la neurona. Se suele utilizar la función sigmoidal para acotar los valores en el intervalo $[0,1]$, la relu para evitar valores negativos en la salida, o la tangente hiperbólica para obtener valores dentro del intervalo $[-1,1]$.

A continuación, tras haber entendido el funcionamiento de una neurona artificial, se explica el funcionamiento de la propia red de neuronas. La red se divide en diferentes capas. Cada capa contiene un determinado número de neuronas, y una neurona solo puede pertenecer a una capa. Existen tres tipos de capas:

- Capa de entrada: Estrictamente es una capa que no está formada por neuronas, ya que no lleva a cabo ningún procesamiento. Simplemente introduce los datos de entrada a la red.
- Capa oculta: Puede haber de 0 a n capas ocultas en una RNA, donde $n \in N$. Estas capas están formadas por neuronas cuya información proviene de la capa anterior, y cuya salida es dirigida a la capa posterior.
- Capa de salida: La última capa de la red. Es aquella que aporta la salida definitiva de la RNA.

En la siguiente figura se puede observar una RNA con una capa de entrada con tres neuronas, tres capas ocultas con cuatro neuronas cada una, y una capa de salida con una sola neurona.

Figura 2.1
Ejemplo de RNA



Fuente: (TIBCO, [s.f.](#))

En la figura también se distingue que cada neurona de una capa i tiene conexiones a todas las neuronas de la capa $i+1$. A las redes con este tipo de conectividad se les llama redes totalmente conectadas. A una red que no esté totalmente conectada se le llamará parcialmente conectada.

Tras entender el funcionamiento básico de las RNAs, se puede introducir que existen diversos modelos dependiendo de su diseño, arquitectura, o reglas internas. Una primera clasificación, y en cuanto al estudio es la que se necesita conocer, se basa en la dirección del recorrido de la información a través de la red, distinguiéndose redes alimentadas hacia delante y las redes con retro-alimentación (Viñuela y León, 2004).

La principal diferencia entre las RNAs alimentadas hacia delante o feed-forward networks en inglés, y las RNAs con retro-alimentación o recurrentes, es que las feed-forward tienen una restricción fuerte que evita que una neurona pueda tener conexiones consigo

misma, mientras que las recurrentes no cuentan con esta restricción, lo que provoca que puedan existir neuronas con una conexión en sí misma formando un ciclo. A nivel práctico, este diseño permite a las RNAs recurrentes tener memoria a largo y corto plazo, al poder guardar estados y reconocer sucesos anteriores. Esto les permite poder tratar grandes secuencias de datos conectados entre sí, en vez de únicamente datos puntuales (Hochreiter y Schmidhuber, 1997).

2.2.2. Usos de las RNAs

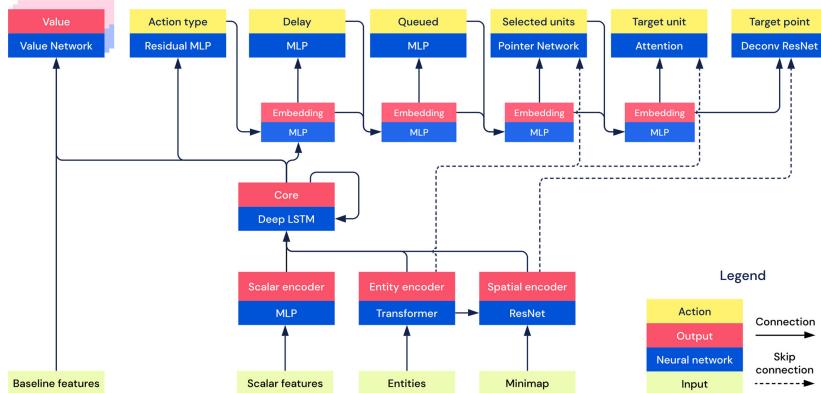
Las RNAs tienen una gran variedad de usos debido a las características anteriormente comentadas. Uno de los principales es la detección de patrones, como pueden ser desde reconocimiento del habla a expresiones faciales (Basu et al., 2010).

También están siendo actualmente muy populares para funcionar como clasificadores de imágenes. Por ejemplo, en (Li et al., 2014) se propone un modelo de RNA capaz de detectar enfermedades pulmonares intersticiales observando fotos de los pulmones de los pacientes. En (Liang et al., 2018) se combinan redes convolucionales y redes recurrentes para detectar enfermedades de la sangre a partir de las células de la misma. La inclusión de redes recurrentes permitían un ajuste más efectivo de los pesos de la red convolucional. En (Xin y Wang, 2019) se propone un nuevo sistema de cálculo del error basado en el algoritmo de aprendizaje de retropropagación, mejorando los resultados en benchmarks de clasificación como el CIFAR-10 o MNIST.

Finalmente, las RNAs, los dos tipos que interesan en este estudio, alimentadas hacia delante y las recurrentes, ya han sido utilizadas para servir de agentes IA para videojuegos de diferentes tipos. Existen ejemplos como el famoso AlphaGo de DeepMind, capaz de dominar el complejo juego Go (Silver et al., 2016). Este juego tiene un espacio de estados gigantesco, por lo que se necesitaba un algoritmo muy potente. AlphaGo utiliza una RNA de gran tamaño para aprender de las partidas de los humanos. Una vez AlphaGo fue capaz de ganar a grandes maestros de Go, tuvo que jugar millones de partidas contra sí mismo para poder avanzar, siguiendo un esquema de aprendizaje por refuerzo (Granter et al., 2017).

De la misma empresa DeepMind se encuentra AlphaStar, capaz de jugar al videojuego de Starcraft II apoyándose en RNAs recurrentes (Stacy, 2020). Específicamente, AlphaStar es un sistema de gran tamaño que comprende técnicas de RNAs, aprendizaje por refuerzo, aprendizaje con múltiples agentes, y aprendizaje supervisado (Vinyals et al., 2019). Primero, el modelo se entrena con aprendizaje supervisado enseñándole partidas de humanos, para que aprenda las bases y conceptos iniciales del juego, para luego pasar a un aprendizaje por refuerzo mediante el juego de partidas. El modelo es penalizado cuando pierde y recompensado cuando gana. Además, como se ha comentado el modelo tiene una red recurrente en su núcleo, específicamente una LSTM (Long short-term memory network) capaz de tratar las largas secuencias de datos a lo largo de la partida.

Figura 2.2
Arquitectura de AlphaStar



Fuente: (Vinyals et al., 2019)

2.3. Algoritmos genéticos

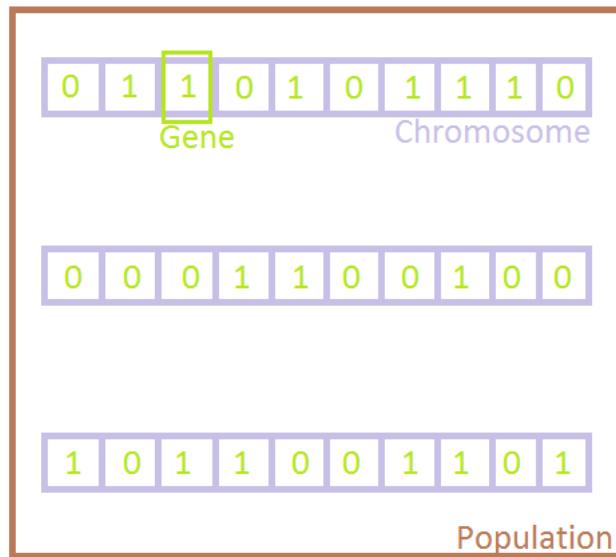
2.3.1. Introducción teórica

Un algoritmo genético (AG) es una técnica de búsqueda basada en los principios naturales de la evolución, propuestos por Charles Darwin (Darwin, 1909). Entonces, según los principios de selección natural, los individuos más fuertes de una generación serán los que tengan más probabilidades de sobrevivir y reproducirse, generando individuos con características similares a ellos, haciendo avanzar a la especie.

Computacionalmente, este proceso se representa como una población que va cambiando de generación en generación. Una población es un conjunto de individuos, mientras que un individuo es una solución al problema. Un individuo está representado por una cadena de caracteres (letras o números), similar al modelo de cadenas de cromosomas. Por ende, la cadena entera que representa al individuo se suele llamar genotipo o cromosoma, dependiendo de la literatura, que a su vez está formado por genes, subcadenas que comprenden cierto atributo del individuo (Mitchell, 1998).

Figura 2.3

Esquema básico de la población de un AG



Fuente: (Gour, 2019)

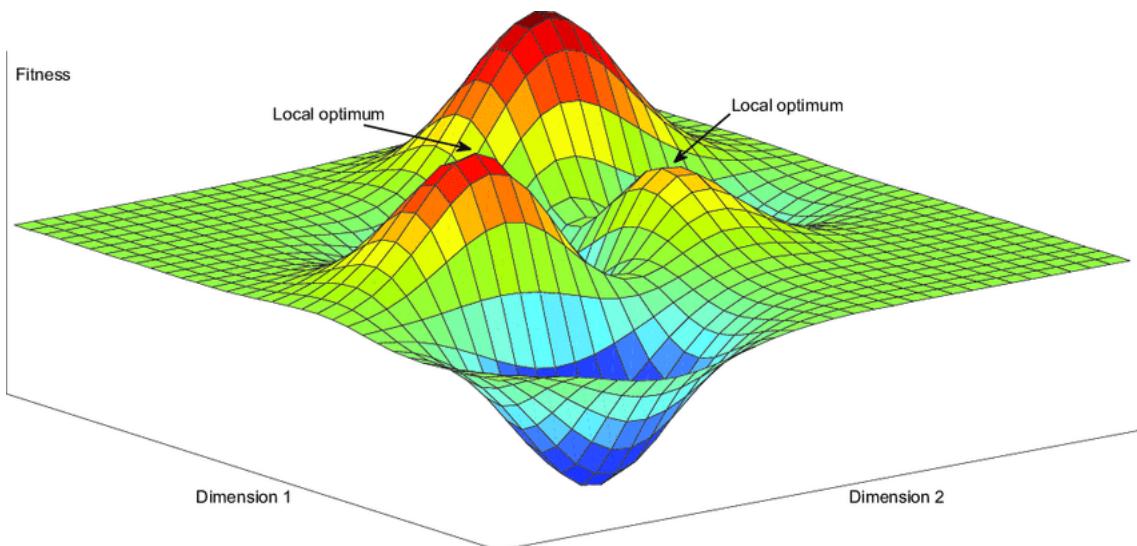
La aptitud de cada uno de ellos se evalúa con una función de evaluación. Por tanto, un AG es un programa que recibe de entrada un conjunto de soluciones (los individuos) para una tarea, y devuelve un nuevo conjunto evolucionado. Para poder evolucionar los individuos de una generación, y por ende, potencialmente optimizar la búsqueda, los AG cuentan con ciertos operadores:

- Selección: Tras conocer el nivel de aptitud de los individuos de una generación, se selecciona cuáles sobrevivirán o serán cruzados en la siguiente generación. Generalmente, aquellos con mayor aptitud tienen más probabilidad de ser seleccionados.
- Cruce: Representa la reproducción sexual. Opera sobre dos individuos a la vez para generar dos descendientes donde se combinan las características de ambos padres.
- Mutación: Modifica al azar parte de la cadena de cromosomas de los individuos.

De esta manera, iterando el proceso base del AG de generación en generación, las poblaciones irán convergiendo hacia el valor de aptitud óptimo dada la función de evaluación.

Un paso importante para entender mejor este proceso de búsqueda y optimización es introduciendo el concepto de paisaje de adecuación (o fitness landscape en inglés). Originalmente fue definido por el biólogo Sewell Wright en 1931 (Wright et al., 1932). Se trata de una representación gráfica de todos los posibles genotipos, asociados a una función de evaluación.

Figura 2.4
Ejemplo de fitness landscape



Fuente: (Nygaard, 2015)

La representación de la adecuación puede formar picos y valles, obteniendo así el nombre de paisaje. Según la publicación de Wright, la evolución fuerza a las poblaciones a dirigirse a los picos (ya sean máximos locales o un máximo global). Desde el punto de vista de un AG, los operadores son los encargados de esta fuerza. El operador de cruce se encarga de una búsqueda local, obteniendo nuevos individuos dentro de un dominio local, mientras que la mutación, al usar una búsqueda aleatoria, permite búsquedas globales a través del paisaje (Zhai et al., 1996). La selección de los genotipos más fuertes finalmente es la que empuja, tras estas búsquedas, la población a subir los picos.

En la naturaleza, los fitness landscapes no suelen ser estáticos, como los que se suelen dar en los AG. Por el contrario, suelen ser dinámicos, y cambios en los genotipos de la población pueden hacer que el paisaje cambie radicalmente (Mitchell, 1998).

2.3.2. Usos de los AGs

Como se ha introducido en la sección anterior, los AGs son métodos de búsqueda y optimización capaces de funcionar en espacios no diferenciables y muy grandes, por lo que tiene muchos usos en diversos campos.

Por ejemplo, se han usado AGs para el diseño de los armazones de aviones. Tradicionalmente, el diseño se hacía con métodos que consideraban el espacio de búsqueda como continuo, lo que no es realista debido a que los materiales disponibles tienen restricciones de tamaño y forma. Por ende, un AG es capaz de optimizar la estructura de manera que se cumplan los requisitos de disponibilidad de los materiales, haciendo una búsqueda por un espacio discreto (Coello Coello et al., 1994).

También han sido utilizados para el diseño de la suspensión de vehículos, el cual es un problema de optimización debido a su alta no-linealidad y a su gran cantidad de mínimos locales. Los métodos de optimización numéricos suelen sufrir ante este tipo de problemas, pero este estudio demuestra que un AG puede mejorar los resultados debido a su capacidad de optimizar en espacios no lineales (J. Zhang et al., 2006).

Para problemas de gestión de tareas es normal aplicar AGs, de nuevo por su capacidad de optimizar espacios no diferenciables. En (Monnier et al., 1998) se estudia un AG capaz de optimizar la gestión de tareas en sistemas distribuidos. Un sistema distribuido se comunica a través de un bus y requiere la asignación y programación de tareas, así como la consideración de los retrasos en la comunicación. Ya que hay plazos para las macrotareas, el problema de encontrar una organización de tareas factible es crítico, y un AG es un método adecuado para optimizar estos espacios.

Para finalizar la introducción a los AGs, éstos no han triunfado individualmente creando agentes IA de videojuegos, al ser un método de búsqueda y optimización. Sin embargo, pueden servir para optimizar ciertas tareas a la hora de crear un juego, como el diseño de ciudades en el juego FreeCiv (Watson et al., 2009) o detectar bugs en el RPG *Journey to the Center of Hawkthorn* y en *Zabuyaki* (Ahumada y Bergel, 2020). Como se verá en la siguiente sección, los AGs destacan más en los videojuegos cuando se usan en compañía de otras técnicas, en este caso, de las RNAs.

2.4. Neuroevolución

La neuroevolución es una forma de IA que usa AGs para generar configuraciones de RNAs. Se le puede incluir dentro del paradigma del aprendizaje por refuerzo, ya que no hace uso de ejemplos correctos o incorrectos para ir aprendiendo, sino que se basa en un sistema cuya evaluación se hace mediante una función de aptitud que recompensa o penaliza las diferentes acciones.

Al igual que las RNAs o los AGs, la neuroevolución cuenta con muchos y muy diversos métodos. Éstos se pueden clasificar fácilmente por los aspectos de la RNA que son evolucionados y por la codificación de la propia red, entre otras taxonomías.

2.4.1. Aspectos a evolucionar

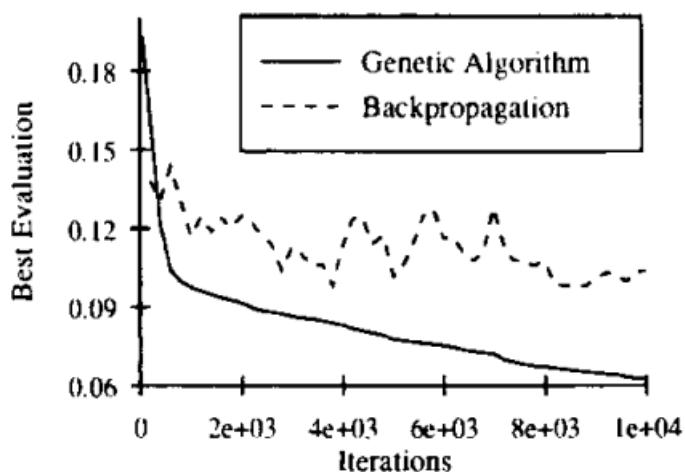
Cada método selecciona qué parámetros de las RNAs optimizará mediante el AG que use, sin embargo, los dos principales aspectos a evolucionar son los valores de los pesos y la topología de la red.

Evolución de pesos

El primer acercamiento es el más antiguo, a la vez que el más básico. Se considera el proceso de aprendizaje, y por tanto, el ajuste de los pesos de la red, como una búsqueda. Los AG , al ser buenos métodos de búsqueda en espacios largos y complejos, pueden obtener resultados adecuados en un espacio como el de los posibles pesos de las conexiones de la red. Numerosos estudios sobre este tema han sido publicados como (Whitley y Hanson, 1989) o (Montana y Davis, 1989) donde se demuestra que el aprendizaje de una red feed-forward mediante evolución de pesos podía mejorar el aprendizaje mediante Retropropagación (Backpropagation), el método de aprendizaje para RNAs más utilizado, para ciertos datos complejos.

Figura 2.5

Evolución de pesos contra Backpropagation (Montana, Davis)



Fuente: (Montana y Davis, 1989)

Sin embargo, aunque pueda parecer que un aprendizaje basado en la evolución de pesos es la opción más atractiva, tiene una gran desventaja. Requiere un coste computacional excesivo, siendo más lento que el método de Retropropagación, basado en el descenso del gradiente. Esto es demostrado por Kitano (Kitano, 1990b) mostrando que los AG consumen más tiempo a la hora de converger en comparación con el algoritmo de Retropropagación.

Evolución de la topología

El siguiente aspecto a evolucionar es la topología de la red, es decir, la cantidad de capas, la cantidad de neuronas por capa y las conexiones existentes. Al contrario que la configuración de los pesos, es común que la topología sea elegida por expertos mediante prueba y error. Sin embargo, diversos métodos han sido desarrollados para evitar esta medida, por ejemplo usando optimización bayesiana para encontrar una configuración

adecuada (White et al., 2019) o utilizando hill-climbing para seleccionar la arquitectura de una red convolucional (Elsken et al., 2017).

Atendiendo a los métodos que nos interesan en este estudio, los evolutivos, ya en 1989 en (Miller et al., 1989) se describen razones de por qué el uso de AGs es un candidato muy adecuado para la búsqueda de topologías.

Entre las más importantes se encuentra el factor de que el espacio de búsqueda de la arquitectura de una red es infinito, ya que puede contar con un número ilimitado de capas y de neuronas por capa. Además, el espacio de búsqueda es no diferenciable, ya que los cambios en el número de capas, conexiones y neuronas es discreto. Por otro lado, hay que tener cuidado con las arquitecturas similares, porque pequeños cambios en ellas pueden producir grandes cambios en el rendimiento de la red.

Este espacio de búsqueda aventaja a los AGs respecto a otros métodos de optimización de topologías, produciendo diversos estudios sobre el tema como (Kitano, 1990a), (Chaves y Chang, 2008) o (Donate et al., 2012) donde se mezcla la evolución de topologías y el método de aprendizaje de Retropropagación para la predicción de series temporales.

TWEANNs

Tras analizar los dos aspectos que suelen ser evolucionados en los métodos de neuroevolución, se presentan las TWEANNs (Topology and Weight Evolving Artificial Neural Networks) sobre las que se centra el método de neuroevolución principal de este estudio. Una TWEANN es una RNA cuyos pesos y topología son evolucionadas. Diversos trabajos han sido publicados sobre estas redes como (Braun y Weisbrod, 1993), (Gruau et al., 1996), (Angeline et al., 1994), (Fullmer y Miikkulainen, 1991). A la hora de trabajar con TWEANNs uno de los factores más determinantes es cómo se va a abordar la codificación de la red, para que el AG pueda optimizar el espacio de todas las posibles redes de la manera más eficiente posible.

2.4.2. Codificación

Generalmente, las TWEANNs pueden ser clasificadas entre aquellas que usan una codificación directa y las que usan una codificación indirecta.

Codificación directa

La mayoría de las TWEANNs utilizan esta codificación. Se les llama directas porque son una representación explícita de la red, por tanto, cada nodo y conexión de la red está representado en el genotipo.

Por ejemplo, una de las codificaciones directas más sencillas es la binaria, usada en (Dasgupta y McGregor, 1992). La representación de las diferentes RNAs es una matriz

binaria que indica qué neuronas tienen conexiones con otras. Sin embargo, este tipo de codificación tiene problemas graves de eficiencia. Primero, el tamaño de la matriz será el cuadrado del número de neuronas de la red, y segundo, el tamaño de la matriz es fijo, por lo que un número máximo de neuronas deberá ser designado por un humano a priori.

Debido a que una representación binaria no es la más natural para diseñar una RNA, la mayoría de las TWEANNs usan una codificación gráfica, representando las RNAs como un grafo. Un ejemplo es la codificación dual en (Pujol y Poli, 2004), donde se representa la red mediante dos codificaciones. Una codifica la red como un grafo y la otra codifica las conexiones mediante una representación lineal. De esta manera, la codificación gráfica se usa para cruces y mutaciones en la topología de la red, y la lineal para los cruces y las mutaciones en la conectividad y los pesos.

Codificación indirecta

La codificación indirecta suele ser la aproximación más compleja. Al contrario que la codificación directa, que representa explícitamente la RNA, la indirecta representa las reglas que hacen falta para construir la RNA a la que se refieren, como en (Mandischer, 1994).

Algunas acercamientos interesantes a la codificación indirecta son usando codificación de reglas de desarrollo, lo que permite mayor escalabilidad, ya que el tamaño de la red no significa mayor número de reglas (Kitano, 1990a), o por ejemplo utilizando una representación fractal de la conectividad (Merrill y Port, 1991).

2.4.3. Usos de la Neuroevolución

La neuroevolución, en conclusión, es un método poderoso para ciertos contextos, debido a su flexibilidad y facilidad de adaptación al problema al que se enfrenta, ya que no necesita apenas información del mismo. Es una técnica adecuada para problemas no-lineales (Gomez, 2003), poco informados, o procesos no-markovianos como el sistema ESP (Gomez, Miikkulainen et al., 1999).

Además es un método que al estar altamente inspirado en procesos biológicos, permiten analizar abstractamente y deshilvanar algunos entresijos de la naturaleza del aprendizaje de los seres vivos. Por ejemplo, en (Channon y Damper, 1998) se crea un *mundo* artificial llamado *Geb* donde, sin ningún tipo de función de adecuación explícita, se encuentran unos organismos cuyas acciones son decididas por la salida de la red neuronal que tiene asignada cada uno de ellos. Las posibles acciones son:

- Reproducirse con el individuo de enfrente
- Matar al individuo de enfrente
- Girar en sentido horario

- Girar en sentido antihorario
- Moverse hacia delante

Las redes neuronales iban evolucionando mediante la reproducción de los organismos y su muerte, generando organismos con diferentes estrategias que formaban un ecosistema con sus propias normas. Por ejemplo, se formó un grupo de organismos depredadores que giraban en un círculo y mataban a todo organismo que se encontraran, otros que aprendían a huir de éstos, y una minoría que con el tiempo aprendía a calcular cómo matar a los depredadores.

Hablando de nuevo de AlphaStar, el modelo de IA capaz de dominar el Starcraft II, utiliza métodos evolutivos para optimizar los parámetros de la RNA (Arulkumaran et al., 2019). Específicamente se combina el algoritmo de aprendizaje de Retropropagación para entrenar a la red y hacer una búsqueda local efectiva, a la vez que con un AG se hace una exploración global.

La neuroevolución ha demostrado además que a la hora de entrenar RNAs profundas (redes de larga escala) puede ser un método a tener en cuenta. En (Such et al., 2017) se demuestra que un método neuroevolutivo para entrenar una RNA sobrepasa al método de Retropropagación en ciertas tareas de aprendizaje por refuerzo. El AG evolucionó con éxito redes con más de cuatro millones de parámetros, las mayores redes neuronales jamás evolucionadas con un algoritmo evolutivo tradicional, consiguiendo resolver juegos de Atari y otras tareas de aprendizaje por refuerzo con tiempos y resultados satisfactorios.

Tras este primer acercamiento a los conceptos básicos de la Neuroevolución, se detallará el funcionamiento del método neuroevolutivo implementado y analizado en este estudio, Neuroevolución de Topologías Aumentadas (NEAT).

2.5. Neuroevolución de Topologías Aumentadas

La Neuroevolución de Topologías Aumentadas (NEAT) es una técnica neuroevolutiva ideada por Kenneth O. Stanley y Risto Miikkulainen en 2002 basada en la optimización de TWEANNs. Esta técnica trata de sobreponer a los métodos neuroevolutivos anteriores mediante la mejora de tres aspectos principales:

- **Resolución del Competing Conventions Problem:** Este problema, presente en todo método neuroevolutivo, se da cuando hay más de una manera de expresar una solución para un problema de optimización de estructura para una RNA. Cuando los genotipos que representan la misma solución no tienen la misma codificación, es probable que el cruce produzca una descendencia defectuosa. (Montana y Davis, 1989). Para las TWEANNs este problema es más grave, ya que RNAs con diferentes topologías pueden tener resultados similares.

- **Proteger la innovación:** En las TWEANNs, cuando se añaden nueva estructura a una red, es común que el valor de adecuación para dicha red baje, ya sea por añadir una nueva neurona, introduciendo un comportamiento no lineal donde previamente era lineal, o al añadir una nueva conexión que aún no ha podido optimizar el valor de su peso. NEAT idea un método basado en especies para preservar esas nuevas RNAs que aún no han tenido tiempo para llegar al máximo de su potencial, evitando que mueran prematuramente.
- **Configuración de la población inicial:** Normalmente, la población inicial de los métodos neuroevolutivos basados en TWEANNs es un conjunto de topologías aleatorias. Sin embargo, esta aproximación tiene varios problemas graves. Primero, se pueden dar topologías que desde el principio estarán destinadas al fracaso, como por ejemplo aquellas para las que no existe un camino de las neuronas de entrada a las salidas. Estas redes tardarán en ser eliminadas del proceso de evolución, causando ineficiencias. Segundo, al empezar con una población aleatoria se hace mucho más difícil llegar a la RNA mínima que soluciona el problema. Es decir, al empezar con topologías que tienen neuronas y conexiones ya establecidas inútiles para la solución final, se incrementa el espacio de búsqueda y por ende se ralentiza el proceso, a la vez que puede dar como solución una red con mayor tamaño del necesario para el problema. Se ha tratado de contrarrestar esto incluyendo en el genotipo el tamaño de la red para penalizar a aquellas que son más grandes (B.-T. Zhang, Muhlenbein et al., 1993). Sin embargo, no es una manera general de solucionar el problema, debido a que la concepción de lo que es una red grande o pequeña puede variar dependiendo del problema y de la complejidad de éste. En NEAT se opta por un desarrollo desde la mínima expresión, reduciendo el espacio de búsqueda y pudiendo hallar la topología mínima.

Tras introducir los tres aspectos principales a tener en cuenta, se detalla el funcionamiento de NEAT y cómo lida con ellos.

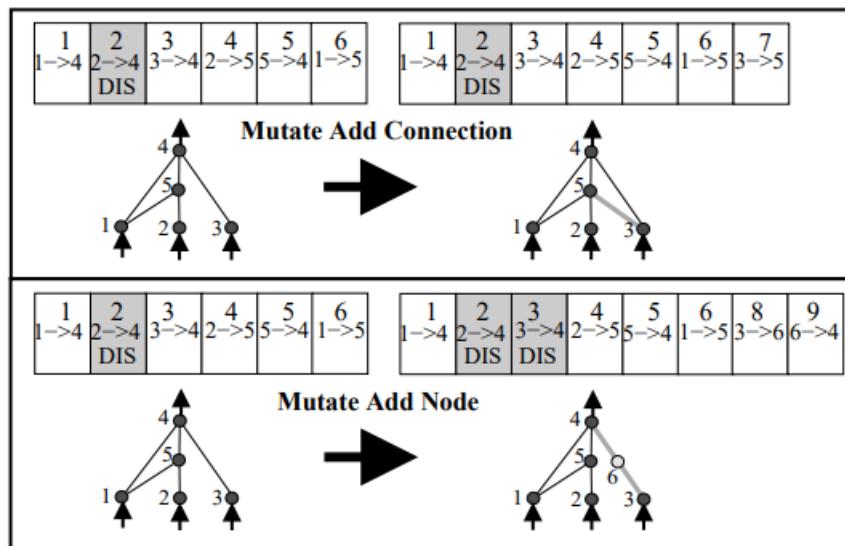
2.5.1. Codificación

Primero, NEAT usa una codificación directa. Cada genotipo es una lista de genes de conexión. Cada gen de conexión representa dos neuronas (de entrada, oculta o de salida) que están conectadas. Cada gen indica la neurona de entrada y de salida de la conexión, su peso, si la conexión está habilitada o deshabilitada, y un número de innovación, cuya función se explicará más adelante.

La mutación, como en todas las TWEANNs, puede cambiar las conexiones, los pesos de las conexiones, y el número de neuronas de la red. Las mutaciones de la estructura de la red funcionan de dos maneras. Si se añade una nueva conexión a la red, al genotipo se le añade un nuevo gen que represente la nueva conexión. Por otro lado, si a la red se le añade una nueva neurona, la conexión donde se añade se divide a la mitad, conectando una parte

de la conexión a la entrada de la neurona, y la otra parte como salida. La conexión que entra en la neurona recibe un peso inicial de 1, y la que sale de ella, el peso de la conexión que fue dividida. De esta manera, se minimizan los efectos en el rendimiento de la red al añadir una nueva neurona. En la Figura 2.6 se aprecian los dos tipos de mutaciones, así como la representación del genotipo.

Figura 2.6
Codificación de NEAT



(K. O. Stanley y Miikkulainen, 2002)

Por tanto, a medida que se vayan sucediendo las mutaciones, los genotipos serán más grandes, al poder crecer sin restricciones. Esto puede ser un problema a la hora de cruzar individuos que tengan un tamaño diferente. Posteriormente se verá cómo se trata esta cuestión.

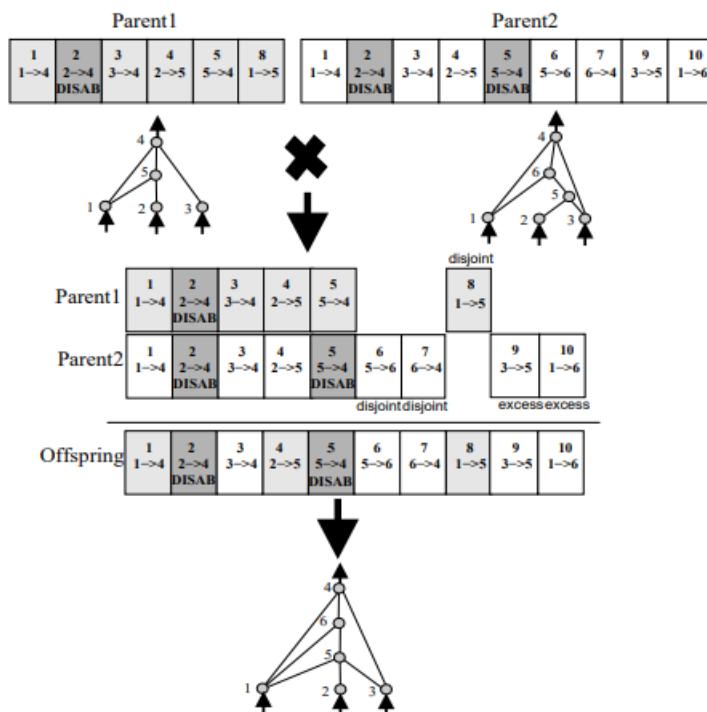
2.5.2. Números de innovación

Para resolver el problema de los cruces, se necesita saber si las dos redes que se van a cruzar tienen una estructura compatible. Esta información se puede encontrar analizando los números de innovación de los diferentes genes de los genotipos. Cuando un nuevo gen aparece, un número de innovación global aumenta en uno y se le asigna al gen. Por ende, el número de innovación representa el orden en el que las diferentes neuronas o conexiones han aparecido. Entonces, dos genes con los mismos números de innovación deben representar la misma estructura. Usando los números de innovación de los genes, se pueden cruzar los genotipos compatibles fácilmente. Por ejemplo, si un gen tiene un número de innovación de 4, cuando se cruce con otro gen, solo podrá cruzarse con ese gen que también tiene el mismo número de innovación, manteniéndose el número de

innovación constante, es lo que se llama un gen compatible. Gracias a que los números de innovación son constantes para cada genotipo, siempre se podrá saber la cronología de la evolución del mismo. Si dos genotipos se cruzan y no coinciden en algunos números de innovación, se dice que tienen genes incompatibles, ya sea exceso o disjunto, dependiendo de si se encuentra fuera o dentro del rango de tamaño del menor genotipo que se cruza. En la Figura 2.7 se observa el cruce entre dos genotipos con genes compatibles, y con excesos y disjuntos.

Figura 2.7

Cruce de dos genotipos en NEAT



(K. O. Stanley y Miikkulainen, 2002)

Esta estrategia resuelve el competing convention problem, además de aportar una manera satisfactoria de cruzar genotipos diferentes y formar una población diversa. El problema de esta diversidad, como se comentaba al principio de la sección, es que las redes más recientes y grandes tienen pocas probabilidades de sobrevivir, debido a que las redes pequeñas convergen más rápido, y que al incluir nuevas estructuras a la red, inicialmente su rendimiento baja, necesitando tiempo para optimizarse.

2.5.3. Especiación

Para proteger a las nuevas redes generadas, la población se divide en especies, donde los individuos podrán competir primero dentro de su propia especie. De esta manera, la innovación topológica es preservada dándole tiempo a cada individuo para optimizarse

compitiendo dentro de un nicho de la población. Los métodos de especiación o nichos ya habían sido utilizados en AGs (Mahfoud, 1995), pero eran muy raros en métodos de Neuroevolución. NEAT implementa este concepto al resultar extremadamente útil para garantizar una buena y justa optimización de sus individuos.

Para que este método de especiación funcione, se necesita dividir a la población en especies donde todos los individuos tengan una topología similar. Para ello, se hace uso de una función que determine lo similar que es un individuo de otro. Usando los números de innovación recientemente explicados, NEAT utiliza una función que calcula la *distancia de compatibilidad* entre los individuos para calcular la especie a la que pertenece. Luego, la distancia de compatibilidad δ vendrá dada por la cantidad de estructura no similar entre dos genotipos. Esto se puede calcular observando los genes en exceso y disjuntos entre cada genotipo, y las diferencias entre los pesos de los genes que sí son compatibles. Siendo N el número de genes del mayor genotipo, E el número de genes en exceso, D el número de genes disjuntos, y P la media de la diferencia entre los pesos de los genes compatibles entre dos genotipos:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 P \quad (2.3)$$

c_1, c_2 y c_3 son constantes que permiten ajustar el peso en el cálculo de la distancia de compatibilidad para cada una de las variables.

Para determinar si un genotipo pertenece a una especie, su distancia de compatibilidad al individuo que representa a esa especie debe de estar por debajo de cierto umbral determinado por el usuario, el umbral de compatibilidad. Cada genotipo se clasifica en la primera especie con la que es compatible.

Para la reproducción NEAT utiliza una función de fitness sharing, donde los individuos de una especie deben compartir su aptitud con el nicho. Así se evita que una especie se haga demasiado grande y acabe agrupando a toda la población. Para calcular el fitness ajustado de un individuo i respecto a su especie f'_i , se obtiene su valor de aptitud f_i entre el número de individuos restantes de la especie:

$$f'_i = \frac{f_i}{N_s - 1} \quad (2.4)$$

Las especies entonces se reproducen primero eliminando aquellos individuos con un bajo valor de fitness ajustado, para luego sustituir la población por la descendencia de los individuos en cada especie. Si alguna especie se estanca, es decir, no mejora su rendimiento dentro de un número de generaciones dado por el usuario, es eliminada de la población.

En definitiva, al dividir la población en especies, se protege la innovación topológica, a la vez que se mejora la eficiencia de la búsqueda de una solución. Esto es posible gracias a la reducción del espacio de búsqueda que ocasiona la división de la población.

2.5.4. Incremento de una estructura mínima

Como se explicó al principio de esta sección, la mayoría de métodos neuroevolutivos empiezan con una población aleatoria. Por el contrario, NEAT empieza con una población uniforme, lo más simple posible. Las nuevas estructuras que se vayan añadiendo a la población son aquellas que hayan sobrevivido y nada más, haciendo que los cambios en las estructuras estén siempre justificados. A su vez, al empezar de una estructura mínima el espacio de búsqueda de NEAT comparado con otros sistemas de neuroevolución es menor, lo que provoca importantes mejoras en el rendimiento.

2.5.5. Rendimiento en las pruebas

Tras la explicación de las cuatro claves para la comprensión de NEAT, se da un breve resumen de sus resultados en las pruebas del estudio en (K. O. Stanley y Miikkulainen, 2002).

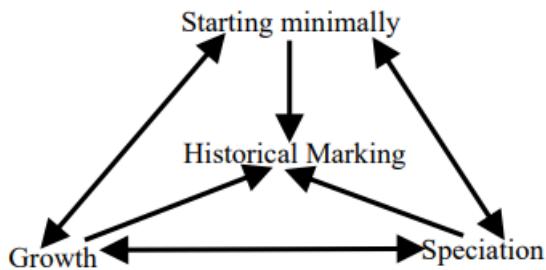
Para probar sistemas neuroevolutivos es normal usar como prueba inicial el problema de XOR ya que no es linealmente separable, y por ende requiere una red con al menos una neurona oculta. NEAT halla una solución en 32 generaciones de media, usando 1 o 2 neuronas ocultas, lo cual muestra como NEAT, al crecer desde una estructura mínima, encuentra soluciones minimizadas.

Luego, se probó en situaciones más difíciles como el *Doble Pole Balancing*, otro benchmark utilizado en los métodos de neuroevolución. Consiste en balancear dos barras situadas encima de un sistema móvil, donde se considera balanceado una barra que esté entre -36° y 36° de la vertical. NEAT fue capaz de resolver la prueba en 24 generaciones, sólo superado por el método neuroevolutivo ESP (Gomez, Miikkulainen et al., 1999). Sin embargo, la prueba se puede complicar si no se le administra al sistema información sobre la velocidad del móvil, convirtiendo esta tarea en una tarea no-markoviana, donde el sistema debe calcular la velocidad por sí mismo y adaptarse a los cambios de la misma. En esta tarea, NEAT vence al sistema ESP usando 286 generaciones frente a 289, pero donde realmente se ve la dominancia de NEAT es en el número de redes evaluadas, donde ESP evaluó 169.466 redes y NEAT 33.184.

La experimentación demuestra que NEAT es un método neuroevolutivo que goza de los beneficios de los métodos neuroevolutivos anteriores, a la vez que mejora en eficiencia y rendimiento haciendo uso de los elementos clave explicados anteriormente, que minimizan la búsqueda y diversifican la población. En la Figura 2.8 se resume cómo las propiedades claves de NEAT se relacionan entre ellas.

Figura 2.8

Dependencias de las propiedades de NEAT



(K. O. Stanley y Miikkulainen, 2002)

2.5.6. Usos de NEAT

NEAT, como método utilizado popularmente para tareas de aprendizaje por refuerzo, ha sido usado en repetidas ocasiones para generar agentes IA para videojuegos. Por ejemplo, un agente para jugar al juego Flappy Bird (Techwithtim, 2019), uno para jugar al Super Mario (wts42, 2018) u otro para jugar al Pong contra sí mismo (John-Trager, 2022).

Por otro lado, se ha usado en aprendizaje supervisado, mezclando NEAT con retropropagación (L. Chen y Alahakoon, 2006). Específicamente, se propone un sistema, L-NEAT, que combina el algoritmo de retropropagación para optimizar los pesos de las conexiones de las RNAs, y NEAT para la búsqueda de una topología adecuada.

NEAT ha sido utilizado para sistemas de aviso de colisiones para vehículos (K. Stanley et al., 2005). En este sistema, primero se evolucionó un modelo usando NEAT que fuera capaz de conducir un coche en un entorno simulado, obteniendo un modelo pseudoracional, que condujera adecuadamente pero que tuviera alguna colisión ocasional. Luego, usando este modelo se entreno un sistema que detectará y avisara de situaciones de riesgo con alta probabilidad de choque.

Además, con el tiempo los principios de NEAT han sido evolucionados y adaptados a los nuevos avances en IA. Estas adaptaciones, entre otras, ha dado lugar al sistema HyperNEAT, que usa las bases de NEAT para evolucionar RNAs de gran tamaño (K. O. Stanley et al., 2009); o al sistema CoDeepNEAT que utiliza NEAT para evolucionar arquitecturas de deep learning (Miikkulainen et al., 2017). Estos sistemas presentan avances importantes en la tecnología y la capacidad de NEAT, sin embargo, no serán comentados más allá de esta breve mención, ya que se alejan del foco de este estudio.

2.6. Estudios sobre Pac-man

Pac-man es un famoso videojuego arcade nacido a principios de los años 80. Por su naturaleza y diseño, suele ser objeto de numerosas investigaciones relacionadas con agentes de IA. En esta sección se detallan algunas de ellas:

Automatic Controller of Ms. Pac-Man and Its Performance: Winner of the IEEE CEC 2009 Software Agent Ms. Pac-Man Competition

Este agente divide en dos la tarea de Pac-man, teniendo un sistema encargado de procesar las diferentes imágenes del juego para obtener la información relevante, y otro para decidir los movimientos de Pac-man. El primer sistema, para agilizar la obtención de la información, en la primera iteración detecta los objetos no-móviles del tablero, para luego solo centrarse en los objetos móviles en las posteriores iteraciones. El sistema de movimiento de Pac-man se basa en reglas de decisión que tratan de optimizar la puntuación y la supervivencia de Pac-man (Thawonmas y Matsumoto, 2009).

Learning to Play Pac-Man: An Evolutionary, Rule-based Approach

En este estudio se trata de resolver una versión simplificada de Pac-man desde un enfoque basado en reglas, pero con la inclusión de un algoritmo basado en PBIL (Population-based incremental learning) para evolucionar las reglas de comportamiento (Gallagher y Ryan, 2003). El algoritmo de aprendizaje automático de PBIL es un AG que en vez de codificar mediante diferentes genotipos la población entera, solo optimiza un genotipo que comprende a la población entera, siendo este genotipo un vector de probabilidades (Baluja, 1994).

Interesting Games through Stage Complexity and Topology

En este trabajo se trata de mejorar un agente ya desarrollado que controla los fantasmas de Pac-man (Yannakakis y Hallam, 2004a). En él, se prueba el modelo creado en diferentes mapas con dificultad ascendente, dando un dato importante para este estudio. La complejidad para atrapar a Pac-man es inversamente proporcional al largo de los pasillos del mapa, por ende, mientras más cortos sean los pasillos, más fácil será para Pac-man escapar (Yannakakis y Hallam, 2004b).

Real-Time Monte Carlo Tree Search in Ms Pac-Man

Esta investigación estudia un método basado en Morte Carlo con árboles de búsqueda para maximizar la puntuación y el tiempo de vida de Ms Pac-man (La contraparte femenina de Pac-man). En este estudio, los fantasmas tienen un comportamiento *greedy*,

es decir, tratan de maximizar las probabilidades de cazar a Ms Pac-man con su siguiente movimiento. El algoritmo dio muy buenos resultados, sobreviviendo tanto contra pocos fantasmas como contra un gran número de ellos, demostrando que los algoritmos de árboles de búsqueda mezclado con reglas de decisión podían tener un adecuado desempeño jugando a Pac-man (Pepels et al., 2014).

Estudio del algoritmo NEAT aplicado al videojuego Pac-Man

Es un TFG elaborado en la Universidad Politécnica de Cataluña por Darío Blasco. El trabajo trata el mismo tema que este estudio, pero desde un punto de vista distinto, ya que el mapa que utiliza y las reglas del juego son diferentes a las propuestas en éste (los fantasmas son deterministas excepto uno). Sin embargo, sus investigaciones han ayudado para fijar ciertos hiperparámetros de NEAT. Por ejemplo, en su estudio se hacen pruebas con diferentes funciones de activación para las RNAs, dando la sigmoide como una de las más apropiadas para la tarea, por ende, en este trabajo no se tratará ese tema, y las RNAs llevarán por defecto la función sigmoide. Lamentablemente, su trabajo concluyó con resultados poco satisfactorios, debido al gran tiempo de entrenamiento, y el comportamiento no determinista de uno de los fantasmas (Blasco Millan, 2019). En el presente estudio, se tratará, entre otros, de probar los límites de NEAT con diferentes características y experimentar con nuevos entornos.

3. ANÁLISIS Y DISEÑO DEL SISTEMA

En esta sección se describe el análisis y las decisiones de diseño del sistema propuesto. En concreto, se propondrán un conjunto de requisitos que el sistema final deberá cumplir, además de los casos de uso que relacionan estos requisitos.

3.1. Requisitos del sistema

En esta sección se presentan los requisitos propuestos, que deberá cumplir el sistema para asegurar un buen funcionamiento. Los requisitos se han clasificado entre funcionales y no funcionales, indicando la prioridad y necesidad de implementación de cada uno. Los requisitos seguirán el formato de la tabla 3.1.

Donde:

- **Id** representa un identificador único para cada requisito donde Z puede ser F o NF, dependiendo si el requisito es funcional o no funcional respectivamente, y donde X es un número cualquiera.
- **Descripción** proporciona una descripción del requisito.
- **Prioridad** indica el nivel de prioridad del requisito, que puede ser Baja, Media o Alta.
- **Necesidad** indica el nivel de necesidad del requisito, que puede ser Opcional, Desirable o Obligatorio.

3.1.1. Requisitos funcionales

Los requisitos funcionales son aquellos que establecen detalles técnicos que describen la funcionalidad del sistema. En la tabla 3.2 se muestran los requisitos funcionales del sistema.

Id	Descripción	Prioridad	Necesidad
RZ - XX	Descripción del requisito RZ - XX	Prioridad del requisito RZ - XX	Necesidad del requisito RZ - XX

Tabla 3.1

Formato de requisitos

Id	Descripción	Prioridad	Necesidad
RF - 01	El sistema es capaz de obtener la posición de los fantasmas	Alta	Obligatorio
RF - 02	El sistema es capaz de obtener la posición del jugador	Alta	Obligatorio
RF - 03	El sistema es capaz de obtener la posición de los puntos de comida	Alta	Obligatorio
RF - 04	El sistema es capaz de cambiar la dirección de movimiento del jugador	Alta	Obligatorio
RF - 05	El sistema permite añadir o eliminar fantasmas	Media	Deseable
RF - 06	El sistema permite la ejecución del algoritmo NEAT	Alta	Obligatoria
RF - 07	El sistema muestra la aptitud del mejor individuo de una generación	Media	Deseable
RF - 08	El sistema muestra la aptitud de todas las especies de una generación	Media	Deseable
RF - 09	El sistema proporciona una gráfica que muestre la evolución de la aptitud al finalizar la ejecución del algoritmo	Alta	Obligatorio
RF - 10	El sistema proporciona un diagrama que muestre la red con mayor aptitud obtenida a finalizar la ejecución del algoritmo	Alta	Obligatorio
RF - 11	El sistema proporciona una gráfica que muestre la evolución de las especies al finalizar la ejecución del algoritmo	Alta	Obligatorio
RF - 12	El sistema es capaz de leer el archivo de configuración	Alta	Obligatorio
RF - 13	El sistema es capaz de aplicar los valores del archivo de configuración como hiperparámetros en el algoritmo NEAT	Alta	Obligatorio
RF - 14	El sistema proporciona la red del mejor individuo al finalizar la ejecución del algoritmo	Alta	Obligatorio

Tabla 3.2
Requisitos funcionales

Id	Descripción	Prioridad	Necesidad
RNF - 01	El sistema muestra la aptitud del mejor individuo de una generación por el STANDARD_OUTPUT	Media	Deseable
RNF - 02	El sistema muestra la aptitud de todas las especies de una generación por el STANDARD_OUTPUT	Media	Deseable
RNF - 03	El sistema proporciona la gráfica de evolución de aptitud en formato .svg	Media	Deseable
RNF - 04	El sistema proporciona el diagrama de la red del mejor individuo en formato .svg	Media	Deseable
RNF - 05	El sistema proporciona la gráfica de evolución de especies en formato .svg	Media	Deseable
RNF - 06	El sistema proporciona la red del mejor individuo en formato .pickle	Alta	Obligatorio
RNF - 07	El sistema será desarrollado en Python	Alta	Deseable

Tabla 3.3

Requisitos no funcionales

3.1.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen las características de la aplicación, es decir, especifican cómo se cumplen los requisitos funcionales. En la tabla 3.3 se muestran los requisitos no funcionales del sistema.

3.1.3. Casos de uso

En esta sección se incluyen los casos de uso extraídos a partir de los requisitos del sistema. El formato utilizado viene dado por la tabla 3.4:

Una vez definido el formato, a continuación se definen los casos de uso considerados.

Finalmente, en la Figura 3.1, se muestra un diagrama con los casos de uso considerados.

CU - XX	
Nombre	Nombre del caso de uso
Fuente	Requisitos relacionados
Precondiciones	Postcondiciones
Precondiciones del caso de uso	Postcondiciones del caso de uso
Descripción	
Descripción del caso de uso	

Tabla 3.4

Formato caso de uso

CU - 01	
Nombre	Iniciar ejecución de NEAT
Fuente	RF - 05, RF - 06, RF - 12, RF - 13
Precondiciones	Postcondiciones
- Definir valores del archivo de configuración - Definir número de fantasma - Establecer ruta donde se guardarán las gráficas	- El algoritmo NEAT se inicia correctamente, con los hiperparámetros y número de fantasmas definido
Descripción	
Se inicia el algoritmo NEAT para los parámetros dados	

Tabla 3.5

Caso de uso - CU - 01

CU - 02	
Nombre	Desarrollo de NEAT
Fuente	RF - 01, RF - 02, RF - 03, RF - 04, RF - 07, RF - 08
Precondiciones	Postcondiciones
- NEAT está iniciado con hiperparámetros válidos - El mapa ha sido generado correctamente	- El algoritmo NEAT se desarrolla correctamente, mostrando información sobre el mejor individuo y las especies de cada generación en pantalla
Descripción	
Se desarrolla el algoritmo NEAT con normalidad	

Tabla 3.6

Caso de uso - CU - 02

CU - 03	
Nombre	Obtención del mejor individuo
Fuente	RF - 14, RF - 02, RNF - 06
Precondiciones	Postcondiciones
- NEAT ha acabado su ejecución sin errores	- Se proporciona la red que representa al mejor individuo en formato .pickle
Descripción	
Finaliza el algoritmo NEAT y se guarda la red ganadora en formato .pickle	

Tabla 3.7

Caso de uso - CU - 03

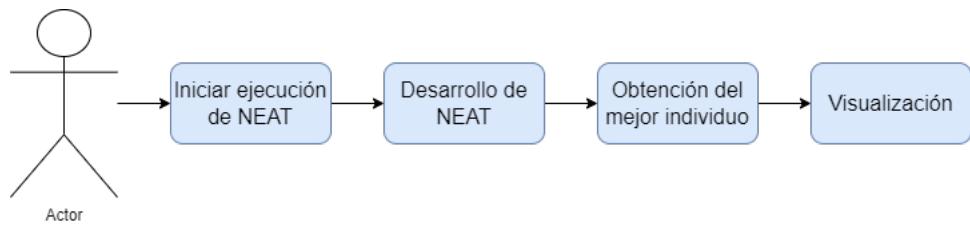
CU - 04	
Nombre	Visualización
Fuente	RF - 09, RF - 10, RF - 11, RNF - 03, RNF - 04, RNF - 05
Precondiciones	Postcondiciones
- NEAT ha acabado su ejecución sin errores y la red ganadora ha sido guardada	- Se proporciona la gráfica de evaluación de la aptitud, la evolución de las especies, y el diagrama de la red ganadora en formato .svg
Descripción	
Se guardan las gráficas del algoritmo al finalizar	

Tabla 3.8

Caso de uso - CU - 04

Figura 3.1

Diagrama de casos de uso



4. IMPLEMENTACIÓN DEL SISTEMA

En este capítulo se detalla cómo se ha implementado el sistema y el algoritmo con sus distintos elementos. También, se explicarán los cambios hechos en el código fuente del juego para adaptar el sistema a la tarea.

4.1. Cambios en el código fuente

La librería utilizada para aplicar NEAT a Pac-man fue *NEAT - python* (McIntyre et al., s.f.), la cual sufrió algunos pequeños cambios que serán mencionados ulteriormente.

Por otro lado, el código fuente de Pac-man sufrió varios cambios de menor y mayor importancia relacionados con su eficiencia y su mapa. Los cambios efectuados han sido:

- **Eliminación del menú de inicio:** Para que el entrenamiento pudiera ser automatizado fácilmente, se eliminó el menú principal, donde se podía seleccionar si salir del juego o empezar una nueva partida. Tras el cambio ejecutado, la partida comienza y cuando alcanza un estado de terminación, el proceso se cierra directamente, sin pasar por el menú principal. Esto permite la automatización de partidas concadenadas de una manera más eficiente.
- **Estados de terminación:** En la versión original, la partida sólo termina cuando Pac-man colisiona contra un fantasma, volviendo al menú principal. Para poder agilizar el proceso de entrenamiento y que no se quede estancado, se han añadido dos nuevas condiciones de término. Entonces, la partida concluirá si:
 - Pac-man colisiona contra un fantasma.
 - Pac-man se come todos los puntos del mapa, ergo alcanza la puntuación máxima.
 - Si la condición de término 1 o 2 no han sido cumplidas, la partida finaliza si se supera un tiempo límite marcado por el usuario.
- **Subida de la velocidad de juego:** La versión original tenía un límite impuesto al transcurso del juego de 30 imágenes por segundo, frames per second en inglés (FPS), para poder ser asimilado por la visión y los reflejos de un ser humano. Al automatizar la tarea y cambiar al jugador por una RNA, 30 FPS ralentiza extremadamente el tiempo por generación. Para hacer frente a este problema, se puso un límite de 600 FPS al juego. Por lo general, debido a la potencia de la computadora donde se ejecuta el entrenamiento, los FPS rondan 300 - 400, acelerando por más de 10 veces el ritmo del entrenamiento.

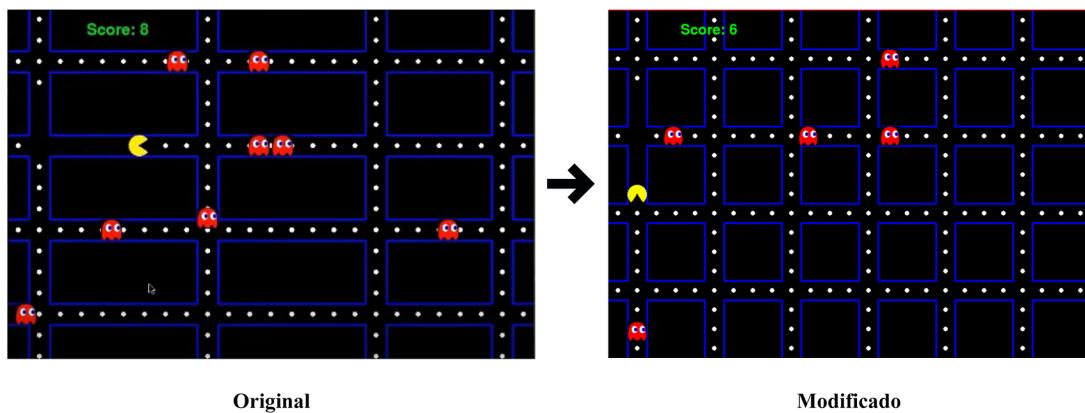
- **Cambio en el mapa inicial:** Como se comentó en la sección 2.6, específicamente en (Yannakakis y Hallam, 2004b), la complejidad de capturar a Pac-man es inversamente proporcional a la longitud de los pasillos del mapa. Por ende, en el caso de este estudio, la complejidad del mapa vendrá dada por:

$$C = M\{P\} \quad (4.1)$$

siendo $M\{P\}$ la longitud media de los pasillos del mapa. Bajo esta medida de la complejidad, el mapa original era bastante complejo, lo que se podía comprobar simplemente jugando en él con los controles. Para simplificar la tarea inicial, se añadieron nuevos pasillos en el mapa, recortando la longitud de los anteriores. En caso de que el modelo obtuviera buenos resultados con esta complejidad, se volvería al mapa original.

Figura 4.1

Paso del mapa original al modificado



4.2. Entradas de la red

Las RNAs pertenecientes a la población deben recibir un conjunto de entradas para poder dar una salida razonable. El formato y el significado de las entradas es de vital importancia, ya que mientras más fielmente representen e informen al agente del entorno, mejor podrá aprender y tomar decisiones. Por ende, tras diversos análisis y pruebas se han obtenido los siguientes formatos de entrada:

4.2.1. Entradas en formato de coordenadas

En este formato, la información que es proporcionada a Pac-man son las coordenadas x e y de los fantasmas, de sí mismo y del dot más cercano. Por tanto, cada una de las

entradas individuales vendría asociada a una neurona de entrada de la red. Las entradas, junto con su abreviatura en los posteriores diagramas, serían:

- Coordenada X del fantasma 1 (GHOST1 X)
- Coordenada Y del fantasma 1 (GHOST1 Y)
- Coordenada X del fantasma 2 (GHOST2 X)
- Coordenada Y del fantasma 2 (GHOST2 Y)
- Coordenada X del fantasma 3 (GHOST3 X)
- Coordenada Y del fantasma 3 (GHOST3 Y)
- Coordenada X del fantasma 4 (GHOST4 X)
- Coordenada Y del fantasma 4 (GHOST4 Y)
- Coordenada X del fantasma 5 (GHOST5 X)
- Coordenada Y del fantasma 5 (GHOST5 Y)
- Coordenada X del fantasma 6 (GHOST6 X)
- Coordenada Y del fantasma 6 (GHOST6 Y)
- Coordenada X del fantasma 7 (GHOST7 X)
- Coordenada Y del fantasma 7 (GHOST7 Y)
- Coordenada X del fantasma 8 (GHOST8 X)
- Coordenada Y del fantasma 8 (GHOST8 Y)
- Coordenada X de Pac-man (PACMAN X)
- Coordenada Y de Pac-man (PACMAN Y)
- Coordenada X del dot más cercano (ND X)
- Coordenada Y del dot más cercano (ND Y)

Entonces, las redes que vayan a recibir este formato de entradas deberán tener 20 neuronas de entrada. Durante el estudio, se eliminaron fantasmas para probar modelos con dificultad creciente. En caso de que se hubieran eliminado fantasmas, las neuronas de entrada respectivas a sus coordenadas recibían una entrada de -1, indicando que ese fantasma faltaba.

Para este formato se probó pasar las entradas sin normalizar y normalizadas, dando mejores resultados cuando las coordenadas estaban normalizadas, ya que los valores de

los pesos de las conexiones no debían sufrir tantas mutaciones para ser viables, y por ende, el algoritmo aprendía y avanzaba más rápido.

Para normalizar las coordenadas, se dividió cada uno de los componentes de la coordenada entre el valor más alto que podía obtener, el ancho y el alto del mapa, unas variables que podían ser modificadas por el usuario. Durante todo el estudio se utilizó un mapa de 800×576 . Por tanto:

$$\text{coordenada}_{\text{norm}} = \left(\frac{x}{800}, \frac{y}{576} \right) \quad (4.2)$$

donde cada valor de la $\text{coordenada}_{\text{norm}}$ sería enviado a su respectiva neurona de entrada.

4.2.2. Entradas en formato distancia-pasillo

En un intento de aportar al agente una visión del entorno más informada y sencilla de interpretar que la abstracción de un conjunto de coordenadas, se diseñó un sistema que informara al agente en función de la distancia (euclídea o manhattan) a Pac-man y el pasillo en el que se encontraban los elementos del mapa. De esta manera, el agente puede aprender a esquivar a los fantasmas sin tener que lidiar con la gestión de coordenadas, lo cual puede ser más laborioso ya que Pac-man deberá aprender las relaciones entre un gran espacio de puntos, sino que deberá aprender las relaciones entre pasillos y las distancias de los fantasmas a él. Por ejemplo, para esquivar a un fantasma usando estas entradas, idealmente se debería comprobar si Pac-man y el fantasma a esquivar están en el mismo pasillo y si la distancia entre ellos es de riesgo. En conclusión, el agente recibe, para cada fantasma y para el dot más cercano a Pac-man, la distancia a la que se encuentra del jugador y el pasillo en el que se halla, siendo las entradas, junto a su abreviatura para los diagramas:

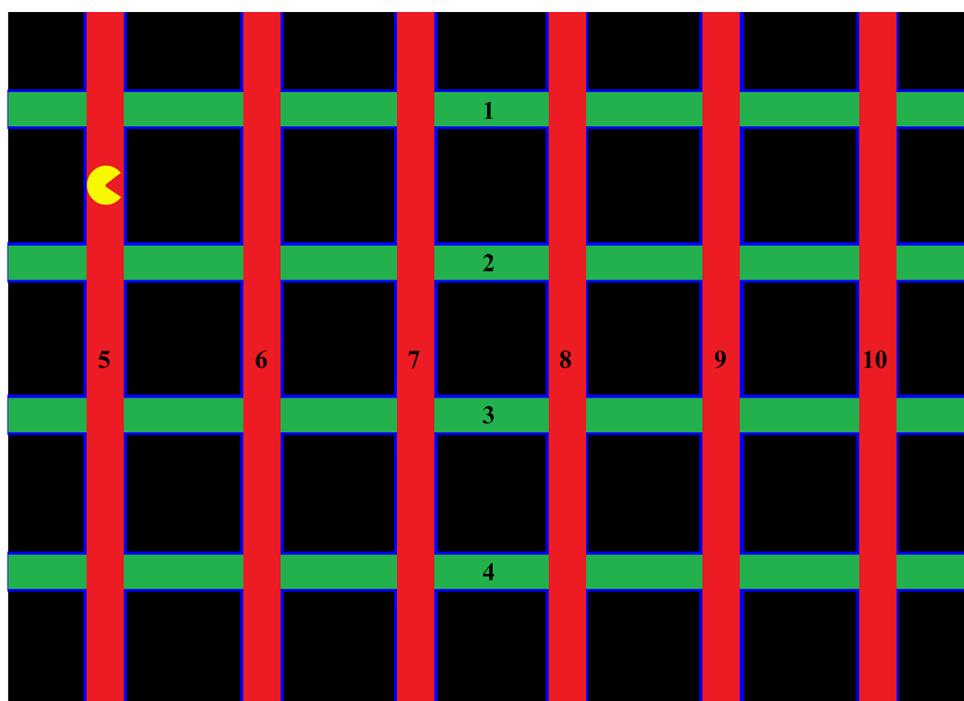
- Distancia al fantasma 1 (GHOST1 DIST)
- Distancia al fantasma 2 (GHOST2 DIST)
- Distancia al fantasma 3 (GHOST3 DIST)
- Distancia al fantasma 4 (GHOST4 DIST)
- Distancia al fantasma 5 (GHOST5 DIST)
- Distancia al fantasma 6 (GHOST6 DIST)
- Distancia al fantasma 7 (GHOST7 DIST)
- Distancia al fantasma 8 (GHOST8 DIST)
- Pasillo del fantasma 1 (GHOST1 CORR)
- Pasillo del fantasma 2 (GHOST2 CORR)

- Pasillo del fantasma 3 (GHOST3 CORR)
- Pasillo del fantasma 4 (GHOST4 CORR)
- Pasillo del fantasma 5 (GHOST5 CORR)
- Pasillo del fantasma 6 (GHOST6 CORR)
- Pasillo del fantasma 7 (GHOST7 CORR)
- Pasillo del fantasma 8 (GHOST8 CORR)
- Pasillo de Pac-man (PACMAN CORR)
- Pasillo del dot más cercano (ND CORR)
- Distancia al dot más cercano (ND DIST)

Por tanto, las redes que vayan a recibir este formato de entradas deberán tener 19 neuronas de entrada.

Para implementar este formato de entradas, se tuvo que idear una enumeración para los distintos pasillos del mapa, que contaba en su versión modificada con 10 pasillos.

Figura 4.2
Numeración de los pasillos



Durante el estudio, se eliminaron fantasmas para probar modelos con dificultad creciente. En caso de que se hubiera eliminado algún fantasma, las neuronas de entrada respectivas a su pasillo recibían un valor de 0, y la de su distancia a Pac-man, un valor de 9999999999, indicando que ese fantasma faltaba.

Para normalizar las entradas, solo se modificó las neuronas relacionadas con la distancia a Pac-man, ya que los pasillos tienen números pequeños fáciles de manejar. La distancia se normaliza dividiendo la distancia a Pac-man de un elemento entre la máxima distancia que permite el mapa, esa es la distancia euclídea entre la coordenada (0,0) y la (800,576), los valores constantes que se han definido como medidas del mapa. Por ende, la distancia normalizada es:

$$distancia_{norm} = \frac{distancia_{original}}{dist((0, 0), (800, 576))} \quad (4.3)$$

siendo *dist* una función que calcula la distancia euclídea entre dos coordenadas.

4.2.3. Entradas de apoyo

A los dos conjuntos de entradas disponibles, se les añade unas entradas extra de apoyo que ayudan al agente a ser sensible a su propio movimiento. Específicamente las entradas indican, de manera binaria, en qué dirección se está moviendo Pac-man y si éste está en una intersección, dado que es el lugar en el mapa que permite ejecutar un cambio de movimiento horizontal a vertical o viceversa. También se ideó una entrada, RECENT_SCORED, que indicaba si el agente había actualizado su puntuación en el anterior movimiento, para motivar al fantasma a cambiar de dirección si este valor era nulo durante mucho tiempo. Sin embargo, ninguna de las mejores redes de prueba utilizaban esta entrada y por tanto se descartó. De esta manera, las entradas de apoyo finales serán:

- Indica si Pac-man se mueve hacia arriba (MOVING UP)
- Indica si Pac-man se mueve hacia abajo (MOVING DOWN)
- Indica si Pac-man se mueve hacia la izquierda (MOVING LEFT)
- Indica si Pac-man se mueve hacia la derecha (MOVING RIGHT)
- Indica si Pac-man se encuentra en una intersección (IN_INTERS)

Estas entradas de apoyo son comunes a los dos conjuntos de entradas vistos anteriormente. Por tanto, estas 5 nuevas neuronas se suman a las neuronas de entrada de los conjuntos anteriores. Entonces, para el conjunto de entradas de coordenadas hará falta una red de 25 neuronas de entrada, y para el conjunto de entradas distancia-pasillo se necesitará una red de 24 neuronas de entrada.

Se debe recordar también que estas redes sufrirán cambios a medida que el algoritmo NEAT se ejecute. Como se ha explicado anteriormente, NEAT trata de encontrar la estructura mínima de la red que es óptima para resolver el problema, por lo que si alguna de las entradas no aporta información o no ayuda a la red a mejorar su rendimiento en la tarea, podría ser descartada. Si hubiera alguna entrada con estas características, tras generaciones del algoritmo NEAT se comprobaría que las redes que utilizan esa determinada entrada tendrían peor aptitud que las que no la usan, haciendo que en generaciones posteriores prevalezcan las redes que omiten esa entrada. Esto permite poder analizar diversos factores de los distintos conjuntos de entradas, como qué entradas son prioritarias, cuáles son innecesarias o incluso qué relaciones tienen entre ellas.

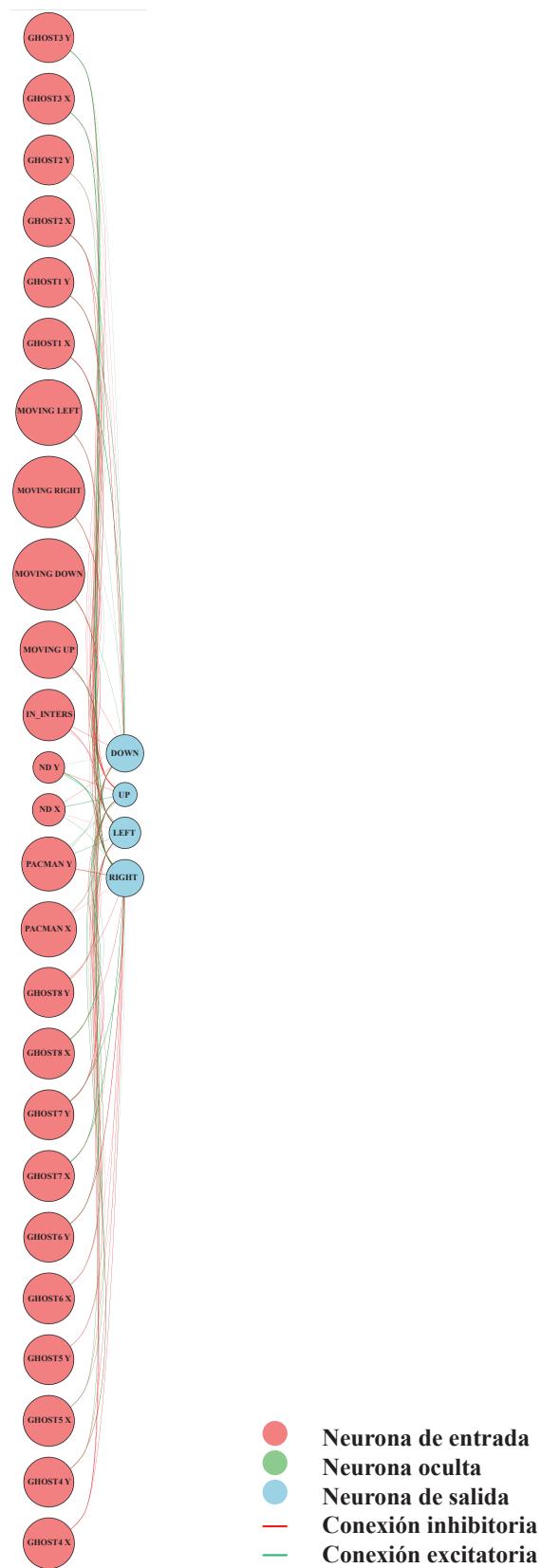
4.3. Salidas de la red

Las salidas de la red decidirán los movimientos de Pac-man, que puede moverse hacia arriba, hacia abajo, hacia la izquierda o hacia la derecha. Por tanto, las redes del algoritmo tendrán 4 neuronas de salida, una para cada posible movimiento de Pac-man. El movimiento elegido se corresponderá con la neurona de salida con mayor valor.

En la Figura 4.3 se muestra el diagrama de una red sin neuronas ocultas totalmente conectada, siguiendo el formato de entradas de coordenadas. Todas las redes presentadas en este estudio se mostrarán en este tipo de diagramas.

Figura 4.3

Ejemplo de diagrama de red totalmente conectada sin neuronas ocultas



4.4. Función de evaluación

La función de evaluación determinará la calidad de una determinada solución, por lo que guiará la búsqueda del algoritmo NEAT hacia una solución óptima. La correcta definición de la función de evaluación, entonces, es crucial para que el algoritmo funcione adecuadamente. La función debe distinguir claramente una buena solución de una mala y debe de medir cuán lejos está la solución propuesta de la solución ideal.

A lo largo de este estudio se han desarrollado varias funciones de evaluación, algunas más exitosas que otras. A continuación se describen las más importantes y las que más han influido en el estudio.

4.4.1. Score y tiempo de vida

La primera función de evaluación creada. Tiene en cuenta el score (número de dots comidos por Pac-man) y el tiempo que Pac-man se mantuvo con vida. En caso de que obtenga la puntuación máxima para el mapa, se guarda el tiempo que ha tardado en completar la tarea.

A la hora de interpretar el tiempo de vida de Pac-man, o lo que será la duración del juego, hay que tener en cuenta que el entrenamiento se ejecuta a unos FPS variables, como máximo 600. Esto fue explicado en las modificaciones del código fuente del juego. Por tanto, dependiendo de los FPS para esa ejecución, el movimiento de Pac-man y los fantasmas puede variar su velocidad, lo que podría aumentar o acortar la duración de un individuo respecto a los otros, a pesar de haber hecho el mismo número de pasos. Para abordar este problema, a la hora de calcular la duración de un individuo se tiene en cuenta los FPS a los que ha ido el juego. En particular, se usa la función de la librería pygame, `get_fps()`, que devuelve la media de los 10 últimos datos de FPS («Pygame documentation», [s.f.](#)). Una vez obtenido los FPS se divide entre una constante. En el caso de este estudio será 30, debido a que 30 FPS es la tasa de imágenes por segundo por defecto en el juego. Por tanto, la duración de un individuo se calcula de la siguiente forma:

$$duration_{individuo} = duration * (FPS/30) \quad (4.4)$$

donde *duration* es el tiempo total registrado de ejecución de la partida, FPS es la salida de la función `get_fps()` y *duration_{individuo}* el tiempo transformado, que es el equivalente al tiempo que hubiera tardado el individuo si se moviera a 30 FPS.

Una vez se calcula la duración del individuo, se puede hallar su valor de aptitud mediante la función de evaluación. En las primeras pruebas se utilizó una función de evaluación que obedecía la siguiente fórmula:

$$f = score - duration_{individuo} \quad (4.5)$$

De esta manera, se trataba de maximizar la puntuación de Pac-man, pero a su vez, penalizar a aquellos individuos que tardaran demasiado con el fin de encontrar un individuo

que tardara poco tiempo en comer todos los dots del mapa. Sin embargo, esta definición de la función de evaluación tenía un fallo grave. Un individuo que hubiera obtenido 18 puntos y hubiera tardado 3 segundos tendría más aptitud que un individuo que hubiera obtenido la máxima puntuación n pero tardando $n - 3$ segundos por ejemplo. Esto hizo que el algoritmo se estancara en un mínimo local que consistía en una estrategia basada en moverse hacia arriba desde el inicio sin parar, obteniendo la mayoría de las veces 18 puntos, la cantidad de dots en la primera vertical (la elección de 18 puntos en el ejemplo no fue casual), hasta morir. Mientras más rápido morían, más aptitud.

Por tanto, se ideó una simple reforma a (4.5). Disminuir el peso que la duración de un individuo debía de tener a la hora de evaluar su aptitud. De esta manera, un agente lento pero con puntuaciones altas prevalecería en la población con la posibilidad de reproducirse y sería mejor evaluado que un agente que consigue poca puntuación pero en muy poco tiempo. Finalmente, la función de evaluación modificada tiene la forma:

$$f = score - \alpha * duration_{individuo} \quad (4.6)$$

siendo α el peso de la duración en la evaluación. Se fijó $\alpha = 0,2$ hallado mediante prueba y error. Aún así α podría variar dependiendo de la tarea y del objetivo propuesto.

4.4.2. Score

Esta es la función de evaluación más sencilla de todas, pero la que en conclusión aporta información relevante para la distinción de individuos. La función sigue la sencilla fórmula.

$$f = score \quad (4.7)$$

Las anteriores funciones de evaluación ralentizaban un entrenamiento ya de por sí bastante lento, debido a que había individuos con estrategias malas pero grandes consumidoras de tiempo, como quedarse quietos o solo moverse en una dirección. Por ello, se decidió no usar el tiempo dentro del cálculo de la evaluación, al menos de manera explícita. Se implementó un nuevo contador, que cronometraba cuánto tiempo llevaba el agente sin puntuar, es decir, sin comerse ningún dot. Si traspasaba cierto límite de tiempo marcado por el usuario, la ejecución del individuo se frenaba y se pasaba a calcular su aptitud mediante (4.7), usando la puntuación que había obtenido antes de la cancelación. De esta manera, se mejoraba la eficiencia del proceso de entrenamiento, descartando estrategias malas y costosas temporalmente, a la vez que se obtenía una aptitud fiel al desempeño del individuo para esa instancia y no se premiaba en ningún momento acciones que acabaran con la vida de Pac-man para acortar tiempo. El límite de tiempo sin puntuar fue variando de 0,75 segundos, a 1,5 segundos o a 3 segundos, dependiendo de la dificultad de la tarea. Nótese que el juego corre por encima de los 300 FPS, por lo que en 0,75 segundos el agente tiene tiempo de sobra para recorrer el mapa en línea recta dos veces.

4.4.3. Media del individuo

Dada la naturaleza no determinista de los movimientos de los fantasmas, un individuo siguiendo una sucesión de pasos específica podría alcanzar más o menos puntuación dependiendo de si un fantasma se cruza aleatoriamente en su camino. Esto añade ruido a la evaluación de los individuos y hace la función de evaluación menos consistente, ya que un mismo individuo en la misma generación puede obtener valores distintos en la función de evaluación. De esta manera, se pierde precisión a la hora de seleccionar el mejor individuo de cada generación.

Para abordar este problema, se tuvieron que añadir pequeñas modificaciones al código fuente de NEAT para permitir contabilizar las veces que un individuo había sido evaluado, es decir, había pasado por la función de evaluación. Con esto, se pretendía calcular el nivel de aptitud de un individuo como la media de las aptitudes anteriores. Por tanto, la función de evaluación usada seguía la siguiente fórmula, donde f' es (4.7) y n el número de evaluaciones del individuo.

$$f = \frac{\sum_{i=1}^n f'_i}{n} \quad (4.8)$$

De esta manera se trataba de atenuar el ruido del no determinismo de los fantasmas, premiando a los individuos que desarrollaran una estrategia con buenos resultados en general, y no a individuos cuya estrategia había sido beneficiada por los movimientos aleatorios de los fantasmas en un momento puntual.

Sin embargo, esta estrategia no mejoró los resultados anteriormente recogidos, debido a que el tiempo de vida de un individuo era en su mayoría muy limitado, de unas dos o tres generaciones. Esto era provocado a su vez, por el propio efecto de los operadores genéticos. A través de una mutación o un cruce se crea un individuo nuevo, con características heredadas de individuos anteriores. Estos individuos nuevos inicialmente tienen un valor de aptitud más alto que individuos que llevan más tiempo, por lo que los antiguos son descartados rápidamente y así en bucle, dando resultados muy similares a usar una función de evaluación sin media.

Por ejemplo, si se tiene el individuo A que lleva tres generaciones en la población con un fitness de 56 en su primera evaluación, 44 en la segunda y 40 en la tercera, es decir 46,66 de nivel de aptitud, y en la siguiente generación se crea un individuo B que consigue una aptitud de 55, lo que lo convierte en su nivel de aptitud definitivo al tener solo una evaluación, mientras que el individuo A vuelve a conseguir 40, es posible que el individuo A sea desechar por el algoritmo, al tener que competir contra individuos nuevos con ventaja ya que su aptitud tiene mayor impacto directo en la aptitud de la generación.

Se trató de abordar penalizando las primeras evaluaciones, bajando su aptitud un 25 % si era la primera vez que se evaluaba el individuo. Luego para la segunda evaluación, se

calcularía la media de las dos evaluaciones con su aptitud original (4.9).

$$f = \begin{cases} f'_i \times 0,25 & n = 1 \\ \frac{\sum_{i=1}^n f'_i}{n} & n > 1 \end{cases} \quad (4.9)$$

Sin embargo, esta función de evaluación acabó siendo sustituida por un criterio de evaluación de media en las especies, como se verá más adelante.

4.5. Hiperparámetros del algoritmo

El archivo de configuración es la herramienta que se utilizará para definir los distintos hiperparámetros del algoritmo NEAT (McIntyre et al., [s.f.](#)). En el apéndice se puede observar el formato del archivo de configuración en su totalidad. En esta sección se detallarán los principales hiperparámetros que se han variado e investigado en este estudio.

■ Configuración general NEAT

- **fitness_criterion:** Puede ser max, min o mean, indicando el objetivo de la optimización.
- **fitness_threshold:** Si la aptitud de un individuo llega alcanza este valor, la ejecución del algoritmo NEAT termina.
- **pop_size:** Indica el número de individuos de la población.
- **reset_on_extinction:** Puede ser verdadero o falso. Si es verdadero, si todas las especies del algoritmo se estancan, reinicia de nuevo el algoritmo NEAT.

■ Configuración Especies

- **species_fitness_func:** Puede ser max, min o mean, indicando cómo se evaluará la aptitud de una especie.
- **max_stagnation:** Define el número máximo de generaciones que una especie puede estar sin mejorar su aptitud. Si una especie traspasa este valor es eliminada.
- **species_elitism:** El número de especies que son protegidas del estancamiento. Si este valor es 2, por ejemplo, las dos mejores especies serán preservadas aunque estén estancadas.
- **compatibility_threshold:** Los individuos cuya distancia de compatibilidad sea menor que este valor, serán de la misma especie.
- **compatibility_disjoint_coefficient:** Indica la aportación de los genes en exceso o en defecto a la distancia de compatibilidad.
- **compatibility_weight_coefficient:** Indica la aportación de la diferencia de pesos en a la distancia de compatibilidad.

■ Configuración Reproducción

- **elitism:** El número de mejores individuos que pasan a la siguiente generación sin ser modificados.
- **survival_threshold:** La fracción de individuos por especie que tiene permitido reproducirse.

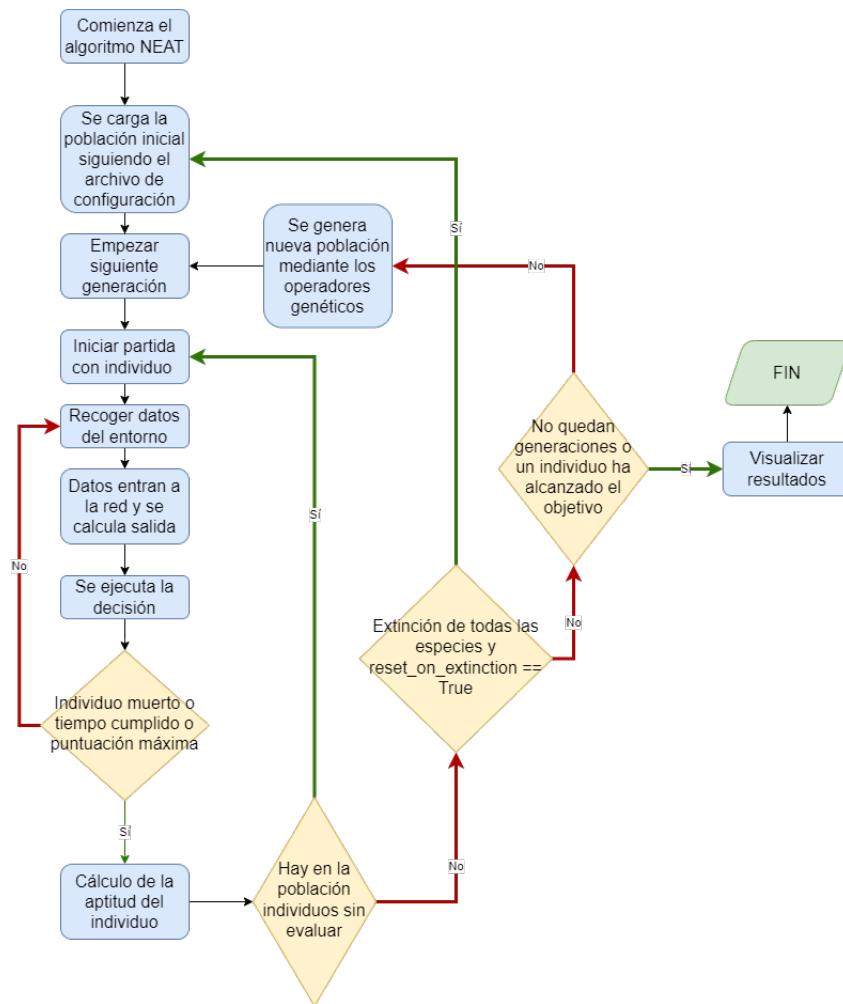
■ Configuración Genotipo

- **conn_add_prob:** Indica la probabilidad de añadir una conexión entre dos neuronas.
- **conn_delete_prob:** Indica la probabilidad de eliminar una conexión entre dos neuronas.
- **node_add_prob:** Indica la probabilidad de añadir una nueva neurona oculta a la red.
- **node_delete_prob:** Indica la probabilidad de eliminar una neurona oculta de la red.
- **num_hidden:** Número de neuronas ocultas iniciales de la población inicial.
- **num_inputs:** Número de neuronas de entrada para las redes.
- **num_outputs:** Número de neuronas de salida para las redes.
- **feed_forward:** Puede ser verdadero o falso. Si es verdadero, NEAT no permitirá conexiones recurrentes, en cambio, si es falso se podría dar el caso de mutaciones en conexiones de neuronas consigo mismas.
- **initial_connection:** Establece la conectividad inicial de la población. Puede ser *unconnected*, sin conexiones iniciales, *full_direct*, cada neurona de entrada está conectada a todas las neuronas ocultas y de salida, y cada neurona oculta a todas las de salida o *partial_direct X*, como *full_direct* pero X es la probabilidad de que cada conexión esté presente o no. Hay más tipos de conectividades pero este estudio se ha centrado en las tres anteriores al ser las más generales.

4.6. Flujo del sistema

Una vez explicadas las partes claves del sistema, se pasa a resumir su funcionamiento conjunto, el flujo del sistema. El algoritmo NEAT sigue una ejecución por iteraciones, sucediéndose generaciones donde los individuos van cambiando mediante los operadores genéticos del algoritmo hasta que el objetivo de la tarea se ha cumplido o ha pasado un número determinado de generaciones. En la Figura 4.4 se puede apreciar un diagrama de flujo simplificado resumiendo el orden de ejecución y la lógica interna del sistema.

Figura 4.4
Diagrama de flujo del sistema



5. EXPERIMENTACIÓN

En este capítulo se detallará la experimentación que se ha llevado a cabo para detectar y analizar los mejores modelos. La experimentación ha seguido el proceso que se muestra en el diagrama de la Figura 5.1. La decisión de esta cronología de experimentación será desarrollada más adelante.

Figura 5.1
Diagrama cronológico de la experimentación



5.1. Métricas

Las métricas elegidas para evaluar los diferentes modelos dependen de si el entorno en el que se prueba es o no determinista. Para el entorno no determinista, donde los movimientos de los fantasmas son de naturaleza aleatoria, se tomarán en cuenta las siguientes métricas:

- La puntuación del mejor individuo (Porcentaje de puntuación obtenida sobre la puntuación máxima del mapa).
- La puntuación media más alta de la población.
- La puntuación media en 100 partidas del mejor individuo.

Para el aprendizaje en un entorno determinista, dado que los fantasmas siempre responderán con los mismos movimientos a las acciones de Pacman, la tarea se convierte en buscar

un camino secuencial que maximice la función de evaluación, por lo que a las métricas serán más simples, siendo éstas:

- La puntuación del mejor individuo (Porcentaje de puntuación obtenida sobre la puntuación máxima que ha obtenido el mejor individuo).
- La puntuación media más alta de la población.

Una vez ya se tienen las métricas definidas, se pasa a relatar los diferentes procesos de experimentación.

5.2. Experimentación con 8 fantasmas no deterministas

Al principio del estudio, se dispuso a probar inicialmente las diferentes configuraciones de parámetros, funciones de evaluación y formato de entradas en el entorno modificado del que se hablaba en el capítulo 4.

Se decidió elaborar unos 3 modelos iniciales para probar las configuraciones básicas usando el formato de entradas de coordenadas, donde luego surgieron otros 3 modelos más al innovar con el formato de entradas de distancia-pasillo. En la tabla 5.1 se detalla el número del modelo, si permite conexiones recurrentes, su conectividad inicial, su formato de entradas y su función de evaluación.

Los modelos son variaciones, además, del archivo de configuración por defecto que se puede ver en el anexo. Todos los parámetros que no fueron nombrados en la sección Archivo de configuración no han sido variados en el estudio.

Todos los modelos de esta sección han sido probados con una población de 150 individuos y 200 generaciones (menos el modelo 7 con 300 al usar la media como fitness), lo que a tiempo de entrenamiento se traduce aproximadamente a 24 horas por modelo. Cada modelo, aparte de las particularidades descritas en la tabla, tiene variaciones en los parámetros de configuración que lo diferencian del resto. En el Modelo 1 se ha subido la importancia de la diferencias entre los pesos son clave para diferenciar entre especies, mientras que en el modelo 2 se le ha dado más importancia a la diferencia de estructura para la distinción de especies. En el modelo 3 la importancia de la diferencia de pesos y estructuras vuelve a su valor por defecto,

En la tabla 5.2 se muestran los resultados de las diferentes métricas cuantitativas de la experimentación. Observando la tabla 5.2 se aprecia que una configuración inicial con la mitad de neuronas conectadas da mejor resultados que las que están conectadas completamente. Al contrario de lo que parece, esta mejora no es porque las redes permitan conexiones recurrentes en ellas, ya que todos los mejores individuos de estos modelos no mantienen conexiones recurrentes. Además, se aprecia que la diferencia de rendimiento entre el formato de entradas de coordenadas y distancia-pasillo no cambia demasiado. Se podría determinar que el mejor modelo es el 4, ya que tiene una puntuación media de

Modelo	Conexiones recurrentes	Conectividad inicial	Formato de entradas	Función de evaluación
1	No	Full direct	Coordenadas	Score y duración
2	No	Full nodirect	Coordenadas	Score y duración
3	Sí	Partial direct 0.5	Coordenadas	Score y duración
4	Sí	Partial direct 0.5	Dist-Pasillo (dist. euclídea)	Score y duración
5	Sí	Partial direct 0.5	Dist-Pasillo (dist. manhattan)	Score y duración
6	No	Full direct	Dist-Pasillo (dist. euclídea)	Score y duración
7	No	Full direct	Coordenadas	Media de individuo

Tabla 5.1

Tabla de modelos de la experimentación con 8 fantasmas no deterministas

su mejor individuo más alta, aunque la puntuación máxima de este no haya sido la más alta, siendo superado por el modelo 7 en 18 puntos. Sin embargo, se puede analizar más profundamente los resultados.

Es muy notable el cambio tan drástico de puntuación obtenida entre la mejor puntuación del mejor individuo y su puntuación media en 100 partidas, pudiendo haber hasta 92 puntos de diferencia entre uno y otro, lo que hace evidente la inmensa cantidad de ruido que aportan 8 fantasmas aleatorios. Esto hace sospechar que los modelos están guiándose por una búsqueda aleatoria, donde dependiendo de la *suerte* hay individuos que usando una estrategia estática consiguen en una partida una puntuación alta, pero no mantienen el nivel. Esto se puede ver reflejado en la Figura 5.2, que representa la evolución del fitness durante las generaciones. En ella se aprecia que el modelo consigue la puntuación más alta al principio, para luego seguir oscilando sin ningún patrón. La media del fitness de la población es un gran indicativo de que la búsqueda que se está produciendo es aleatoria, ya que permanece constante durante todo el entrenamiento en todos los modelos, como se aprecia en la Figura 5.2 y 5.3. En la Figura 5.4 se aprecia que al utilizar la función de evaluación que premia al individuo con mejor media, la media de la población entera sube levemente más que el resto, aunque esto no asegura que su mejor modelo pueda tener una media alta tras 100 partidas como refleja la tabla 5.2.

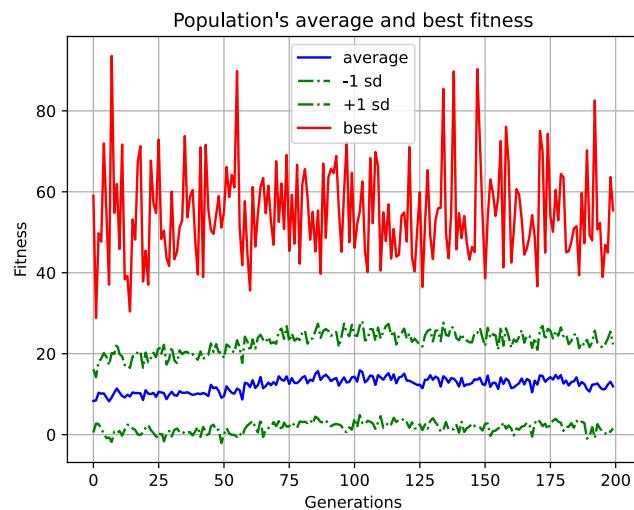
Modelo	Punt. mejor individuo	Punt. media población	Punt. media mejor individuo
1	70 (38 %)	11,5	10,38
2	76 (41,3 %)	13,74	9,56
3	104 (56,52 %)	14,45	12,83
4	102 (55,43 %)	19,21	18,4
5	98 (53,26 %)	18,33	10,51
6	98 (53,26 %)	19,92	13,41
7	119 (64,67 %)	20,4	14,95

Tabla 5.2

Tabla de resultados de la experimentación con 8 fantasmas no deterministas

Figura 5.2

Evolución del fitness del modelo 5



En la Figura 5.3 se observa la evolución del modelo 4, aparentemente el mejor modelo, donde se aprecia grandes indicios de búsqueda aleatoria.

Figura 5.3
Evolución del fitness del modelo 4

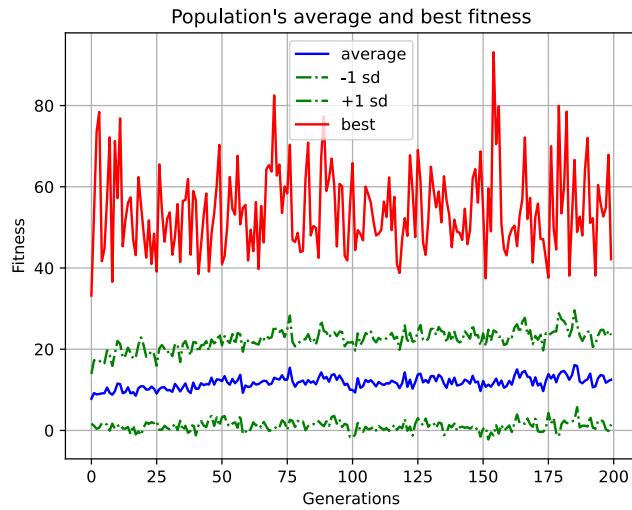
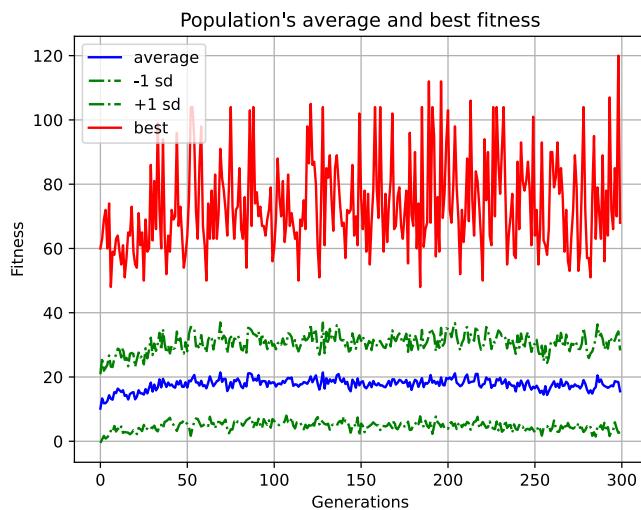


Figura 5.4
Evolución del fitness del modelo 7

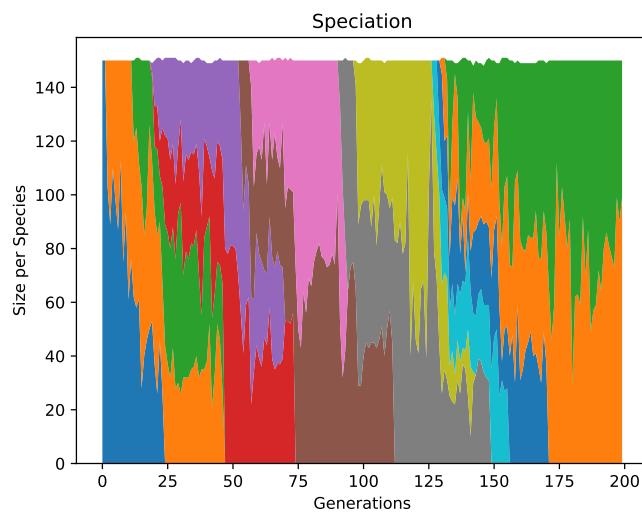


5.2.1. Conclusión de la experimentación con 8 fantasmas no deterministas

En esta primera sección de experimentación se ha establecido un contacto inicial con el problema, demostrando que el método actual no es satisfactorio para resolverlo correctamente. El algoritmo, debido a la naturaleza aleatoria del problema, sufre de mucho ruido en las evaluaciones. La función de evaluación no es capaz de medir qué individuos sirven y cuáles no, debido a que el paisaje de adecuación, el fitness landscape explicado en el estado del arte en la sección de AGs, es altamente dinámico, y en cada partida es diferente provocando que una estrategia prometedora en una primera partida, no lo sea

en la segunda ni en la tercera. Esto causa que la función de evaluación no sea consistente y que el algoritmo ejecute una búsqueda a ciegas. Como se aprecia en la Figura 5.5, no hay especies predominantes o longevas, sino que todas nacen y mueren rápidamente al no encontrar soluciones y estancarse.

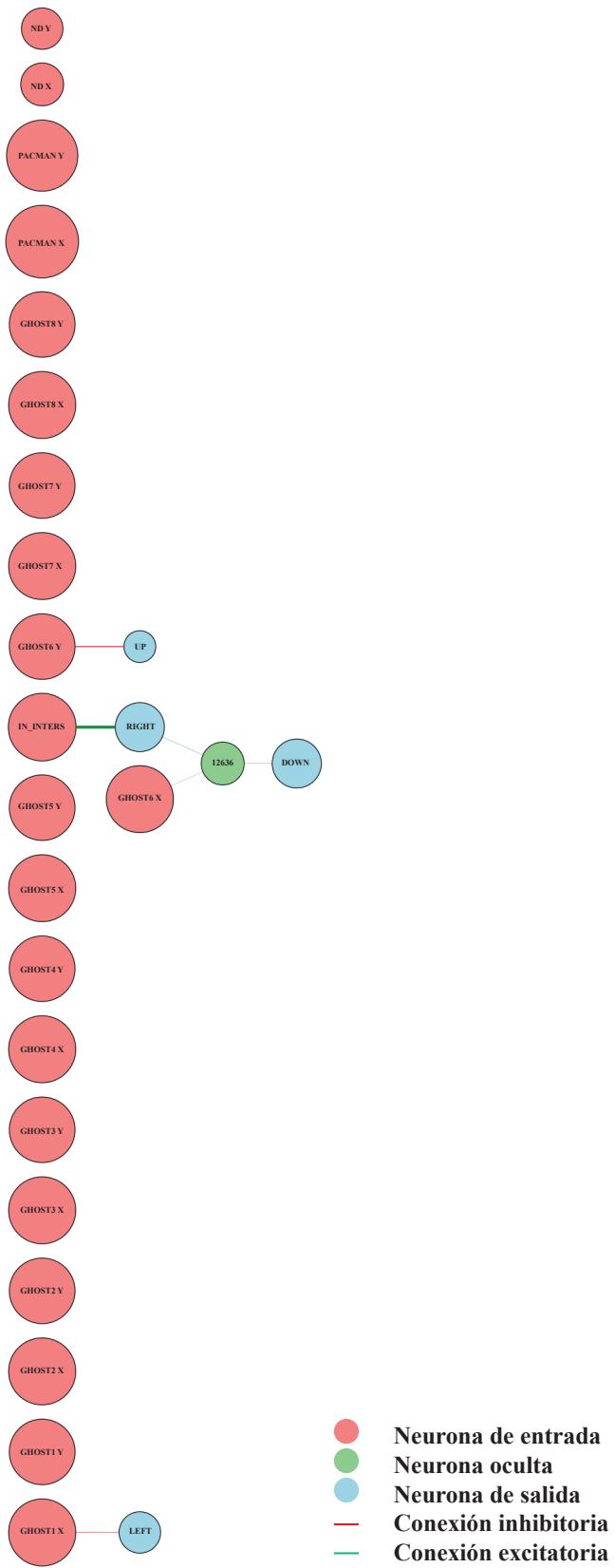
Figura 5.5
Registro de las especies del modelo 4



A parte de la búsqueda aleatoria, se aprecia que los modelos con menos conexiones iniciales normalmente sobrepasan a los que empiezan totalmente conectados. Como se ha observado durante el proceso, a medida que surgen individuos los que sobreviven son aquellos con una estructura más simple, debido a que permiten hacer estrategias más sencillas que por probabilidad funcionan más a menudo al centrarse en menos entradas. Por ende, los que empiezan totalmente conectados y tienen en cuenta todos los fantasmas tardan más en llegar a este tipo de estrategias, premiando a los más sencillos, aunque en este estudio lo ideal sería que los modelos usaran el conocimiento total que tienen de la ubicación de los fantasmas para triunfar.

Por ejemplo, en la Figura 5.6 se aprecia cómo el modelo 7 deja casi la totalidad de las entradas sin participar en la decisión, y además se ven decisiones cuestionables como que la coordenada x del primer fantasma sea lo único que influye en girar a la izquierda, seguramente para esquivarlo al principio ya que es el más cercano a Pac-man al inicio. También, cada vez que esté en una intersección girará a la derecha debido a la fuerte conexión exitatoria entre IN_INTERS y la salida RIGHT. Esto demuestra lo que se comentaba anteriormente, el sistema busca estrategias estáticas simples con menos probabilidad de muerte, en vez de aquellas dinámicas que reaccionen al entorno en su completitud, ya que éstas necesitan un crecimiento de la estructura que al principio es menos efectiva que las estrategias que aportan las redes más básicas.

Figura 5.6
Red ganadora del modelo 7



Modelo	Conexiones recurrentes	Conectividad inicial	Formato de entradas	Función de evaluación
Dist-pas	No	Unconnected	Dist-pasillo	Score
x-y	No	Unconnected	Coordenadas	Score

Tabla 5.3

Tabla de modelos de la experimentación progresiva con fantasmas no deterministas

5.3. Experimentación progresiva con fantasmas no deterministas

Tras los resultados anteriores, se decidió cambiar el proceso de entrenamiento y experimentación en general. Se planteó que 8 fantasmas a la vez era demasiado ruido al principio que no dejaba al algoritmo desarrollar redes con un sistema de aprendizaje correcto. Entonces, el sistema se entrenaría primero sin fantasmas, para que el agente encontrara la estrategia óptima para conseguir la mejor puntuación, y a partir de ahí, el mismo modelo se entrenaría con un fantasma, para luego añadir dos y así consecutivamente. En esta sección de la experimentación se añadió el sistema del cronómetro que mide cuánto tiempo lleva sin puntuar un individuo, referido en la explicación de la función de evaluación *Score*, para poder desechar a los menos fructíferos y optimizar el tiempo de entrenamiento.

En esta experimentación se desarrollaron dos modelos con los mismos hiperparámetros, pero con distinto formato de entradas. Se decidió esto para centrar la búsqueda en encontrar la mejor manera de representar el entorno y hacer una justa comparación. Se eligió un archivo de configuración muy similar al por defecto debido al enorme espacio de estados que posee. Se podría dedicar un trabajo entero a la variación y estudio de cada uno de estos hiperparámetros. Los modelos creados se muestran en la tabla 5.3.

Primero, se propone una conectividad inicial nula, para que poco a poco las redes vayan cogiendo las entradas que necesiten para cada número de fantasmas. Los modelos tienen una probabilidad de borrar o crear una conexión del 0,5 y de crear o borrar neuronas de un 0,2. Las especies pueden estar 20 generaciones como máximo sin mejorar su puntuación, y cada generación tiene un elitismo de 2 individuos, que pasarán indemnes a la siguiente generación. En cada especie, el 20 % de los individuos tiene posibilidad de reproducirse. Finalmente, una de las medidas más cruciales es que el rendimiento de una especie se valorará por la media del fitness de sus individuos, en vez de usar el fitness del mejor individuo de la especie. Así, la especie se estancará si sus individuos no mejoran globalmente, impidiendo que un individuo con *suerte* haga mantenerse una especie no prometedora. De esta manera, se sustituye la función de evaluación de las medias de un individuo (4.9) por este sistema que además afecta de mejor manera al desarrollo de las especies.

Inicialmente los dos modelos llegan a 184 puntos (100 %) relativamente rápido, antes de las 100 generaciones, con una estrategia similar que consiste en empezar a bajar y en

Modelo	Punt. mejor individuo	Media población	Punt. media mejor individuo	Límite sin puntuar	Número de fantasmas
Dist-pas	184 (100 %)	70,3	100,08	1,5s	1
x-y	184 (100 %)	60,8	88,76	1,5s	1
Dist-pas	163 (88,58 %)	43,9	47,38	3s	2
x-y	183 (99,45 %)	42,88	53,07	3s	2

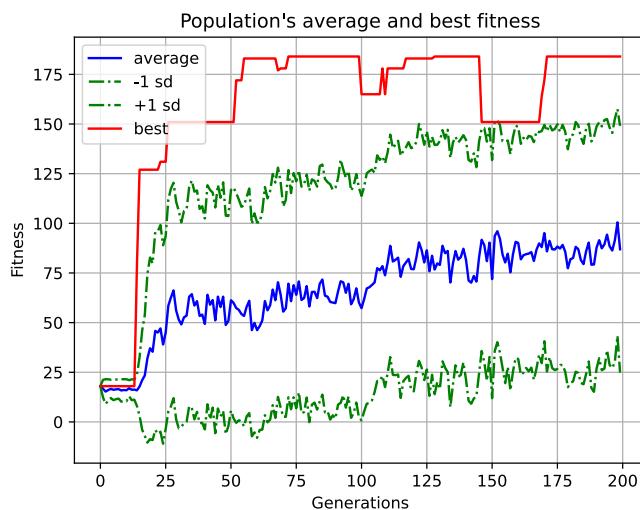
Tabla 5.4

Tabla de resultados de la experimentación progresiva con fantasmas no deterministas

cada intersección girar a la derecha. En la Figura 5.7 se puede ver la evolución del fitness del modelo que usa el formato distancia-pasillo. Se aprecia además cómo llega al máximo para luego bajar y volver a subir. Eso es debido a que, al llegar a 184 puntos, el fitness de la especie que tenía el individuo de 184 puntos no subió la media y se eliminó. Esta decisión de prolongar el entrenamiento a 200 generaciones a pesar de que un individuo había llegado ya a la puntuación máxima se debe a que se quería aumentar la cantidad de individuos con máxima puntuación, para luego a la hora de añadir fantasmas hubiera más variedad de estrategias óptimas. En la gráfica se demuestra que se consiguió este objetivo consiguiendo una media de fitness de la población de más de 100 puntos.

Figura 5.7

Evolución de fitness del modelo Dis-pas sin fantasmas



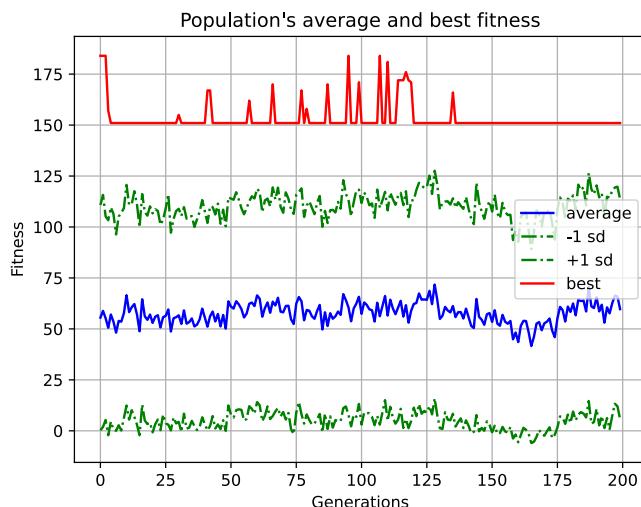
Tras entrenar los modelos sin fantasmas, se prueba a añadir 1 y 2 fantasmas, para luego analizar los resultados y guiar el siguiente paso de la experimentación.

En la tabla 5.4 se pueden observar los resultados de los diferentes modelos. Se aprecia como la media cae al cambiar de uno a dos fantasmas en ambos modelos casi 20 puntos, y la puntuación media de los mejores modelos es capaz de caer 50 puntos, un 27,17 % de la puntuación total del mapa.

Observando la tabla se ve que el modelo con un formato de entradas de coordenadas parece desenvolverse lvemente mejor, a pesar de que para un fantasma el modelo dist-pas sea mejor, es debido a que su mejor individuo es un individuo que en la experimentación sin fantasmas consiguió la puntuación máxima y con un fantasma consiguió con la misma estrategia estática esquivarlo una vez, aunque luego el peso del no determinismo hizo que se eliminara y que el rendimiento del modelo se estabilizara en 151 puntos excepto por subidas infrecuentes, como se puede apreciar en la Figura 5.8. Este estancamiento se produjo debido a la supremacía de un individuo que se expandió, haciendo que los individuos siguientes adoptaran la misma estrategia, dejando muy poca innovación en la población.

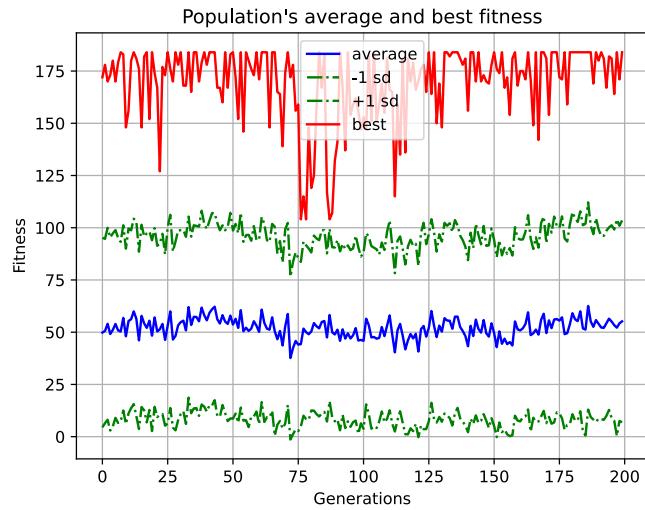
Figura 5.8

Evolución del fitness del modelo dist-pas con una fantasma



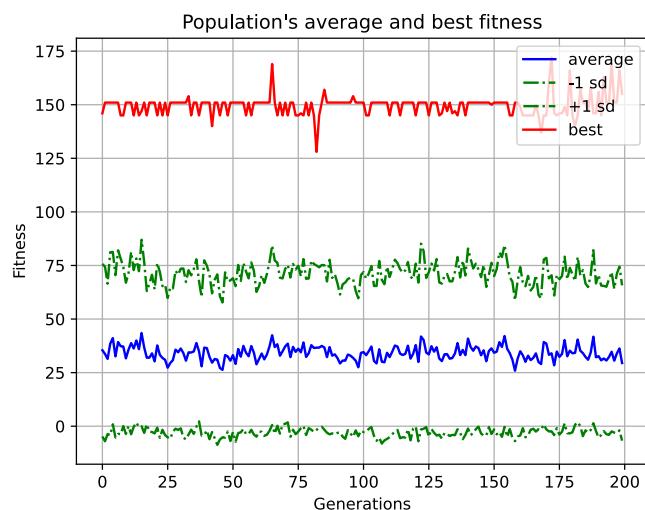
En cambio, el modelo de coordenadas cedió a lo que parece ser una búsqueda aleatoria, pero consiguiendo mejor puntuación en conjunto, llegando a la puntuación máxima varias veces durante la ejecución.

Figura 5.9
Evolución del fitness del modelo x-y con un fantasma



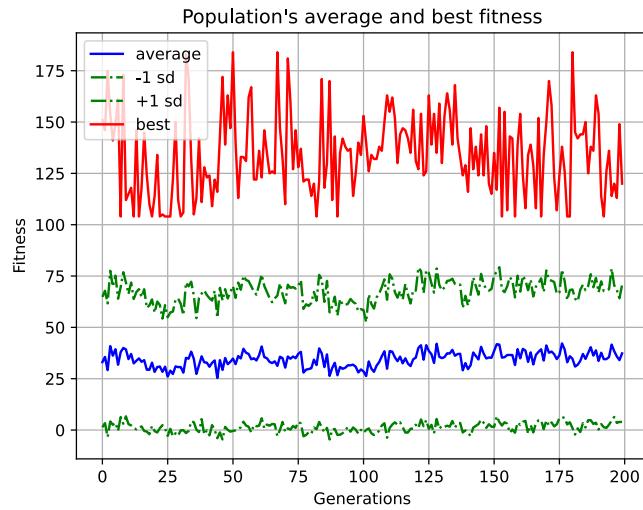
Tras añadir el segundo fantasma, el modelo dis-pas siguió comportándose de la misma manera, con menos innovación todavía. Su problema principal es que su estancamiento con muchos individuos de 151 de puntuación, luego no aseguraba que éstos fueran capaces de mantener esa puntuación jugando varias partidas seguidas, sino que su rendimiento caía.

Figura 5.10
Evolución del fitness del modelo dis-pas con dos fantasmas



Mientras, el modelo x-y siguió con un comportamiento parecido al que tenía con un fantasma, pero bajando la media de sus individuos.

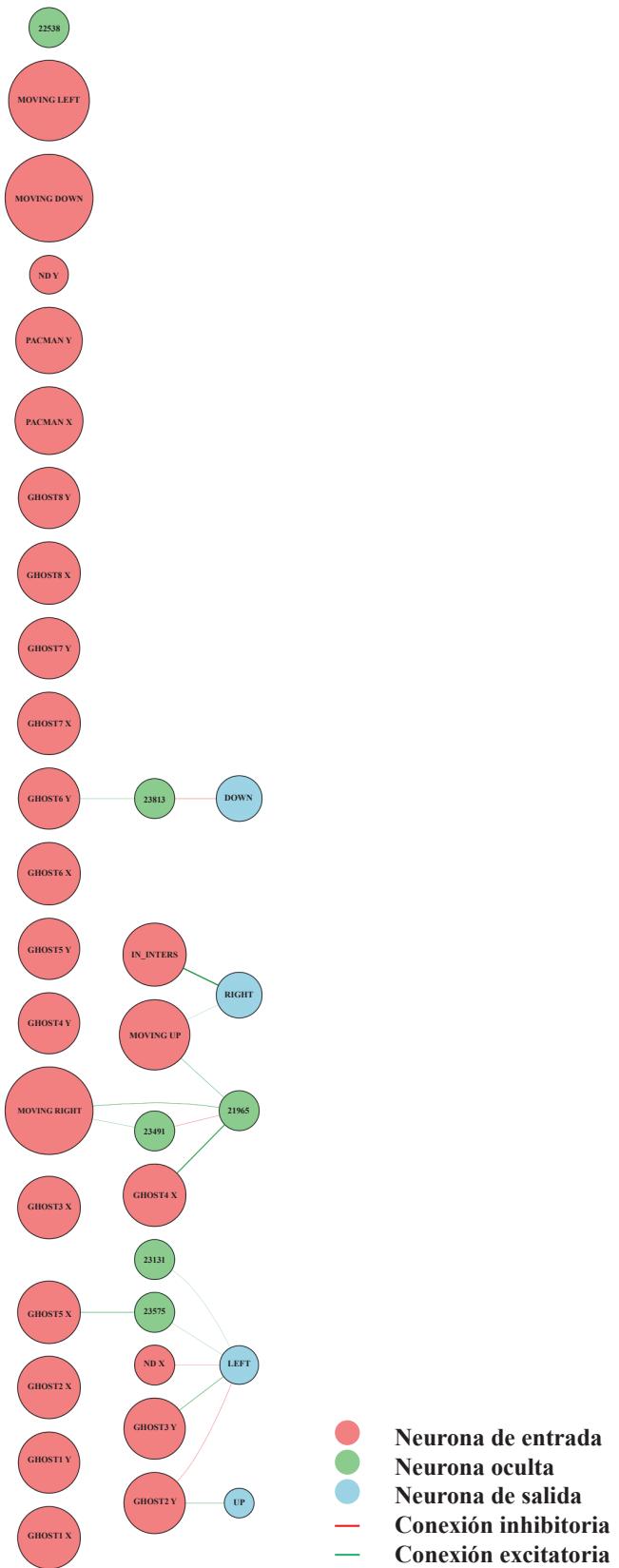
Figura 5.11
Evolución del fitness del modelo x-y con dos fantasmas



La red resultante del modelo x-y con dos fantasmas se muestra a continuación en la Figura 5.12. Se puede ver como no utiliza los datos de ninguno de los dos fantasmas, el 1 y el 2, excepto la coordenada Y del fantasma 2. Predomina el uso de la coordenada X del dot más cercano, y utiliza el conocimiento sobre si se está moviendo hacia arriba y hacia la derecha, junto con si está en una intersección, para activar el movimiento de girar a la derecha. Esto tiene sentido ya que en ejecución, el agente va hacia arriba y cuando encuentra una intersección va a la derecha, para luego en la siguiente intersección seguir subiendo y así repetidamente, subiendo en forma de escalera, excepto en muy pocas ocasiones que gira a la izquierda para comer un dot debido a la influencia de la conexión con la entrada de la coordenada X del dot más cercano. Esta red destaca también por tener conexiones inútiles, como aquella que une la entrada GHOST6 Y del fantasma 6, no presente, con la neurona de salida DOWN. La inutilidad de esta conexión se demuestra al ver que el agente nunca va hacia abajo. Estas conexiones inútiles se deben a que el modelo aún no ha tenido tiempo de encontrar una estructura mínima.

Figura 5.12

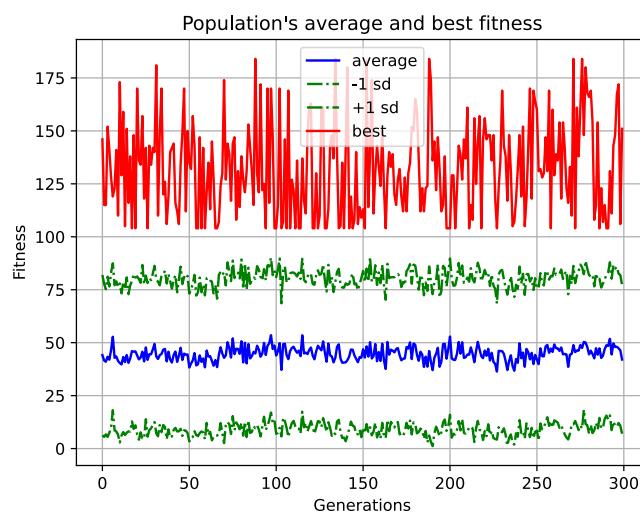
Red ganadora del modelo x-y con dos fantasmas



Tras estos dos modelos, se decidió seguir entrenando el modelo x-y con dos fantasmas debido a sus mejores resultados, tratando de conseguir el comportamiento que se busca en este estudio, el de un agente que es capaz de reaccionar a los movimientos de los fantasmas en tiempo real, sin una estrategia predefinida. Para ello, se entreno durante 300 generaciones más. Este modelo consiguió subir la media levemente de la población hasta llegar a alcanzar los 52 puntos de media, aunque el ruido de la función de fitness seguía mermando el rendimiento.

Figura 5.13

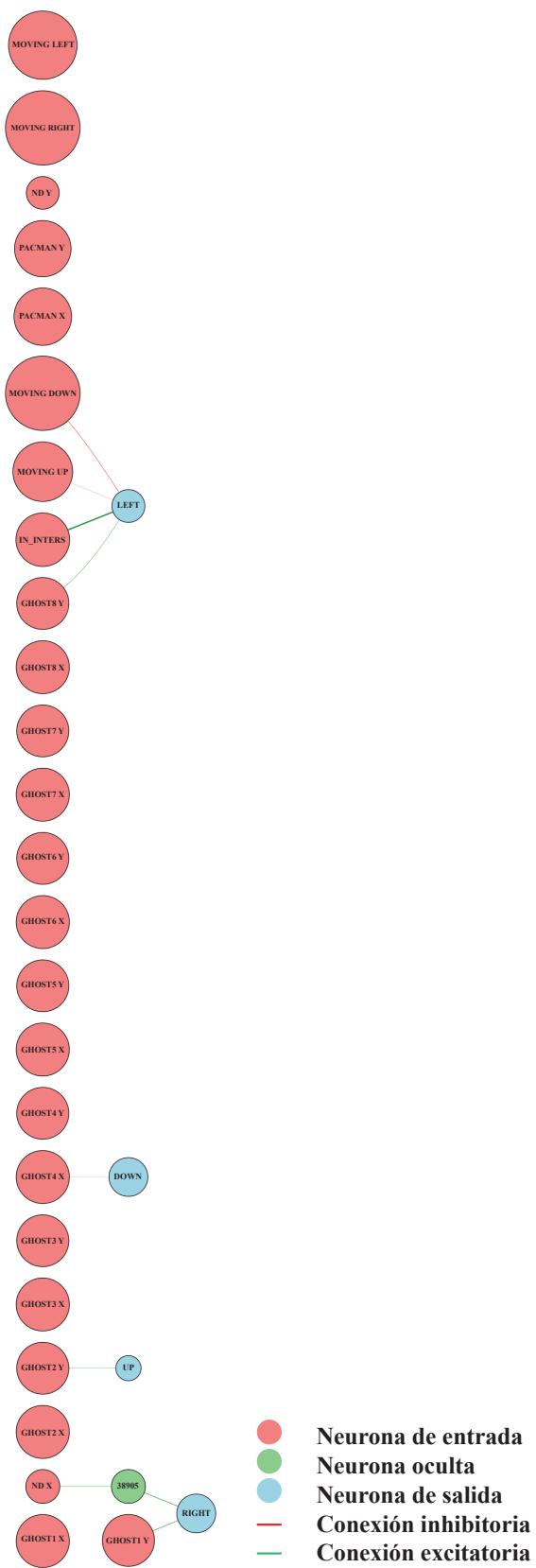
Evolución del fitness del modelo x-y con dos fantasmas y 300 generaciones extra



Además, el rendimiento medio de su mejor individuo también mejoró, consiguiendo una media de 58,8 puntos en 100 partidas, aproximadamente completando un 32 % del mapa. La red creada tiene todavía una estructura más simple que la anterior, demostrando la tendencia de NEAT de encontrar la mínima estructura óptima. Esta red utiliza la coordenada Y del primer fantasma y la coordenada X del dot más cercano para determinar si gira a la derecha y la coordenada Y del fantasma 2 para determinar si va hacia arriba, dotando al agente de cierta capacidad de reacción. Sin embargo, las conexiones más fuertes se sitúan entre IN_INTERS y la neurona de salida LEFT mediante una conexión fuertemente excitatoria que provoca que en cada intersección el agente gire a la izquierda, excepto en determinadas ocasiones. La cantidad de conexiones inútiles ha disminuido desde la red de la Figura 5.12.

Figura 5.14

Red ganadora del modelo x-y con dos fantasmas y 300 generaciones más



5.3.1. Conclusión de la experimentación progresiva con fantasmas no deterministas

En esta experimentación se ha visto que el problema existente en la experimentación con 8 fantasmas persiste, viendo cómo el rendimiento se ve gravemente afectado por la inclusión de solo 2 fantasmas al mapa. Estos dos fantasmas ya son capaces de aportar ruido suficiente a la función de evaluación para ralentizar mucho la búsqueda. Además, el algoritmo sigue tratando de encontrar estrategias simples que sean capaces de lograr una puntuación decente la mayoría de las veces, sin embargo, la última red generada ha sido lo más cercano a un agente con conciencia sobre los fantasmas, a pesar de no usar sus coordenadas completas y por ende no ser capaz de esquivarlos en todos los contextos. Aún así, los mejores individuos son los que hacen buen uso de las entradas de apoyo, IN_INTERS, MOVING_UP, MOVING_RIGHT, etc, ya que permiten al agente crear rutas estáticas con baja probabilidad de muerte, aunque siguen estando lejos de ser agentes adecuados, no alcanzando ni el 40 % de la puntuación del mapa.

Tras estos resultados con baja puntuación y rendimiento, pero que servían para analizar y entender el interesante comportamiento de NEAT, se decidió que un entorno no determinista necesitaba de otro tipo de entrenamiento y funciones de evaluación centrado en varias partidas para cada individuo en cada generación, lo que supondría un incremento del coste temporal del entrenamiento excesivo para este estudio. Por ello, se decidió seguir analizando el algoritmo de NEAT en un entorno determinista.

5.4. Experimentación con fantasmas deterministas

Para crear un entorno determinista, se modificó el código de la toma de decisiones de los fantasmas. Ahora, en vez de ser aleatoria, seguirá un algoritmo codicioso. Cada vez que un fantasma llegara a una intersección, eligirá la dirección que reduce la distancia, euclídea o de manhattan, entre él y Pac-man. A su vez, para dotar de más variedad al entorno, se crearon dos tipos de fantasmas. Unos (EU) perseguirían a Pac-man haciendo uso de ese algoritmo codicioso con distancia euclídea y otros (MH) tomarían la decisión contraria que el algoritmo con distancia de manhattan sugiere. De esta manera, habrá fantasmas que persigan a Pac-man y otros que irán en la dirección contraria a él. Esto permite un entorno variado, donde los fantasmas van en diferentes direcciones. Al implementar un entorno donde todos los fantasmas perseguían a Pac-man, era sencillo esquivarlos y hacer que estuvieran en una captura infinita detrás del jugador. En conclusión, mientras más variedad de movimientos, más difícil para el agente encontrar una estrategia óptima, siendo la máxima expresión de dificultad los movimientos no deterministas.

Tras tener el entorno, se pasó a hacer una prueba inicial con dos fantasmas deterministas, un EU y un MH. Básicamente, al ser los fantasmas deterministas la tarea se puede reducir a encontrar el camino más óptimo para solucionar un laberinto. Esta vez la función de evaluación sí será consistente y un individuo siempre tendrá la misma aptitud independientemente del número de partidas que se jueguen.

Modelo	Conexiones recurrentes	Conectividad inicial	Formato de entradas	Función de evaluación
Dist-pas-det	No	Full direct	Dist-pasillo	Score
x-y-det	No	Full direct	Coordenadas	Score

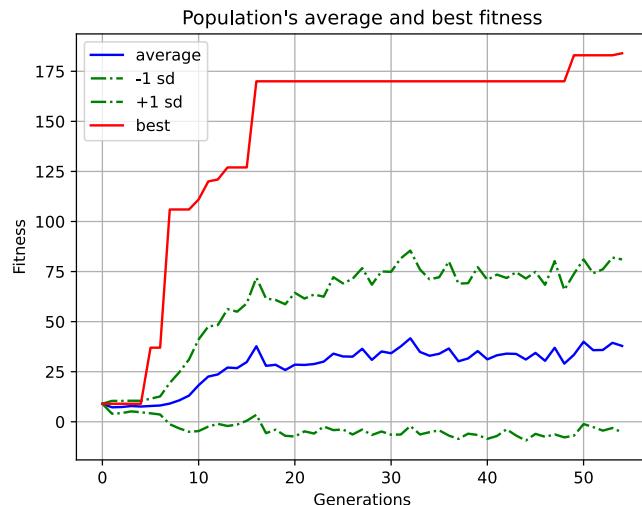
Tabla 5.5

Tabla de modelos de la experimentación con fantasmas deterministas

Se probó con el mismo modelo x-y anterior, con la misma función de evaluación e hiperparámetros. Rápidamente se solucionó la tarea usando menos de 60 generaciones para llegar a la puntuación máxima. El objetivo se consiguió tan rápido que la mayoría de individuos se quedaron atrás en puntuación, provocando una media baja de la población en general.

Figura 5.15

Evolución del fitness del modelo x-y con dos fantasmas deterministas



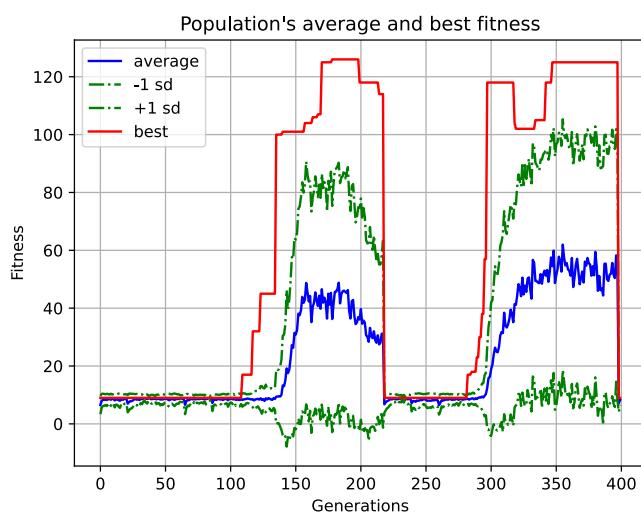
Tras encontrar tan rápido una solución para dos fantasmas se pasan a probar dos modelos más con 5 fantasmas, 3 EU y 2 MH. Los modelos son descritos en la tabla 5.5. El archivo de configuración es muy similar al usado en el anterior, cambiando el porcentaje de individuos que se reproducen de un 20 % a un 40 % para promover las búsquedas locales. Además se subieron levemente las probabilidades de una mutación en la estructura para también promover búsquedas globales sobre el fitness landscape. Como se ve en la tabla 5.5 se puso una conectividad inicial completa, ya que, al haber 5 fantasmas, deberá usar más conexiones para tenerlos en cuenta, por lo que así se optimiza la búsqueda.

Resulta interesante la evolución del fitness del modelo dis-pas-det, en la Figura 5.16, que alcanza 125 puntos para luego extinguirse toda la población. Tras la extinción se crea una nueva población que resulta en un suceso similar, se traspasa la barrera de los 120

puntos y se vuelve a extinguir. Esto es debido a que el formato de entradas distancia-pasillo tiene una gran desventaja a la hora de esquivar fantasmas. Tiene puntos ciegos. Como se ve en la Figura 4.2, los pasillos verticales en las intersecciones se superponen a los horizontales, por lo que si un fantasma se mueve recto horizontalmente, en las intersecciones su asignación de pasillo cambia, para luego volver al anterior pasillo al cruzarla. Esto hace que el agente tenga puntos ciegos en las intersecciones, donde es vulnerable a los fantasmas al no tener un conocimiento preciso de su pasillo.

Figura 5.16

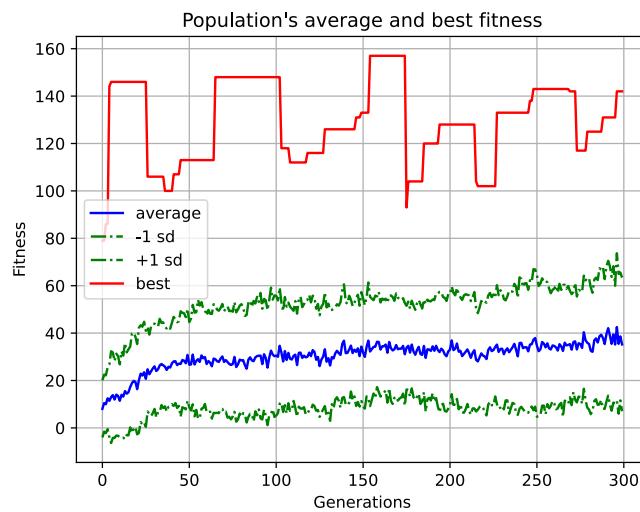
Evolución del fitness del modelo dis-pas-det con cinco fantasmas deterministas



Esto contrasta con el rendimiento de el formato de coordenadas, en la Figura 5.17, con un conocimiento preciso de la posición de los fantasmas en todo momento, llegando a alcanzar un 85,32 % de puntuación sobre el total y con una puntuación media de la población ascendente.

Figura 5.17

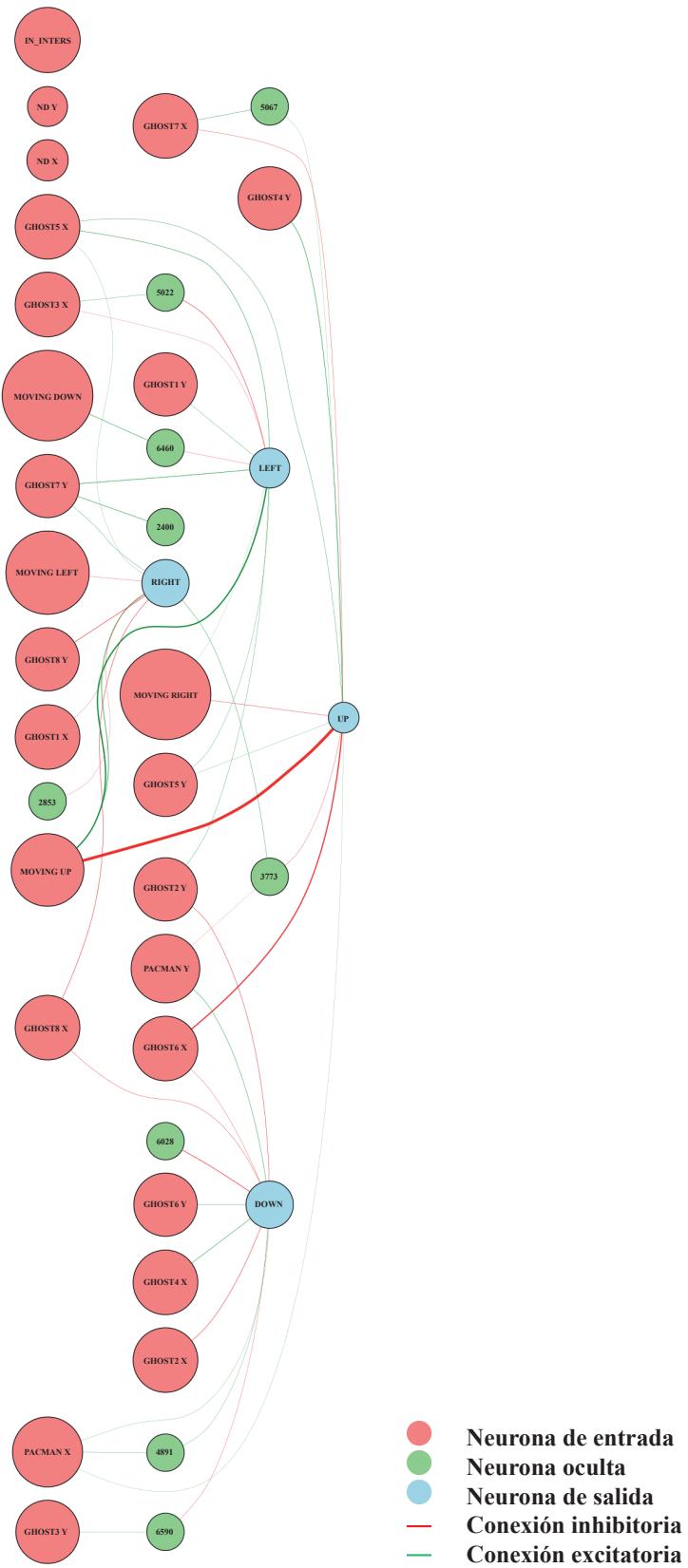
Evolución del fitness del modelo dis-pas-det con cinco fantasmas deterministas



La red que representa al mejor individuo del modelo x-y-det es más próximo al objetivo del estudio, siendo el modelo más fructífero por ahora. Un agente que tenga en cuenta los movimientos de los fantasmas para elaborar estrategias más elaboradas y adaptativas al entorno, aunque al hacer los fantasmas deterministas se pierde la adaptatividad ya que el agente sólo funciona con esos fantasmas con sus determinadas estrategias de movimiento. En la red se observa que todas las entradas participan en la decisión menos la posición del dot más cercano e IN_INTERS. Por tanto, el agente va a ciegas respecto a los dots y cuando puede girar. Si se analiza la red más profundamente, se ve que las conexiones más fuertes son aquellas relacionadas con la entrada MOVING_UP que dirige la mayoría de giros a la izquierda y los movimientos hacia arriba, dejando una influencia más débil a las posiciones de los fantasmas. Esto supone que el agente se mueve más guiado por sus movimientos anteriores que por los movimientos de los fantasmas a su alrededor.

Figura 5.18

Red ganadora del modelo x-y-det con cinco fantasmas



- Neurona de entrada
- Neurona oculta
- Neurona de salida
- Conexión inhibitoria
- Conexión excitatoria

Modelo	Punt. mejor individuo	Media población
x-y-det	128 (69,56 %)	51
x-y-det-wmi	147 (79,89 %)	37

Tabla 5.6

Tabla de resultados de la experimentación del modelo x-y-det y x-y-det-wmi con 8 fantasmas deterministas

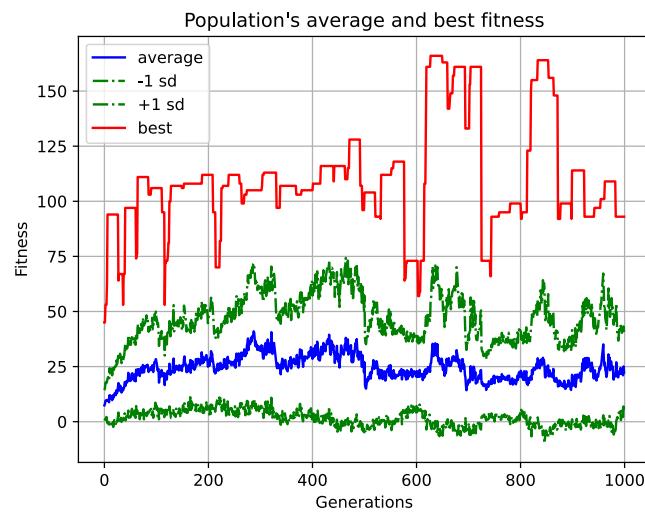
Viendo como las entradas de apoyo han servido a los agentes para memorizar secuencias de movimiento y utilizarlas en vez del conocimiento sobre los fantasmas, se experimentó con dos modelos. El x-y-det anterior y el modelo x-y-det-wmi (wmi - without movement information) que carece de las entradas de apoyo que aportan información al agente de los movimientos de Pac-man. Se probarán estos dos modelos con 8 fantasmas, como en la versión original, pero deterministas, 4 EU y 4 MH.

Cada uno será sometido a 200 generaciones y el que mejor resultados tenga, será elegido como el mejor modelo para recibir un entrenamiento de 1000 generaciones y tratar de obtener la máxima puntuación posible. La tabla 5.6 refleja los resultados de la experimentación, siendo el modelo x-y-det-wmi el que mejor puntuación ha obtenido, y por ende, siendo el mejor a pesar de tener una media más baja. Al ser un entorno determinista se valora más los resultados altos individuales que la media de la población, ya que los resultados son constantes.

Finalmente se ejecuta el algoritmo NEAT con 1000 generaciones para el modelo x-y-det-wmi teniendo un tiempo de entrenamiento de dos días y medio. Como se ve en la Figura 5.19 los resultados han oscilado junto a la creación y la eliminación de especies, hasta obtener el mejor individuo capaz de obtener 166 puntos, 90,21 % puntos sobre el total.

Figura 5.19

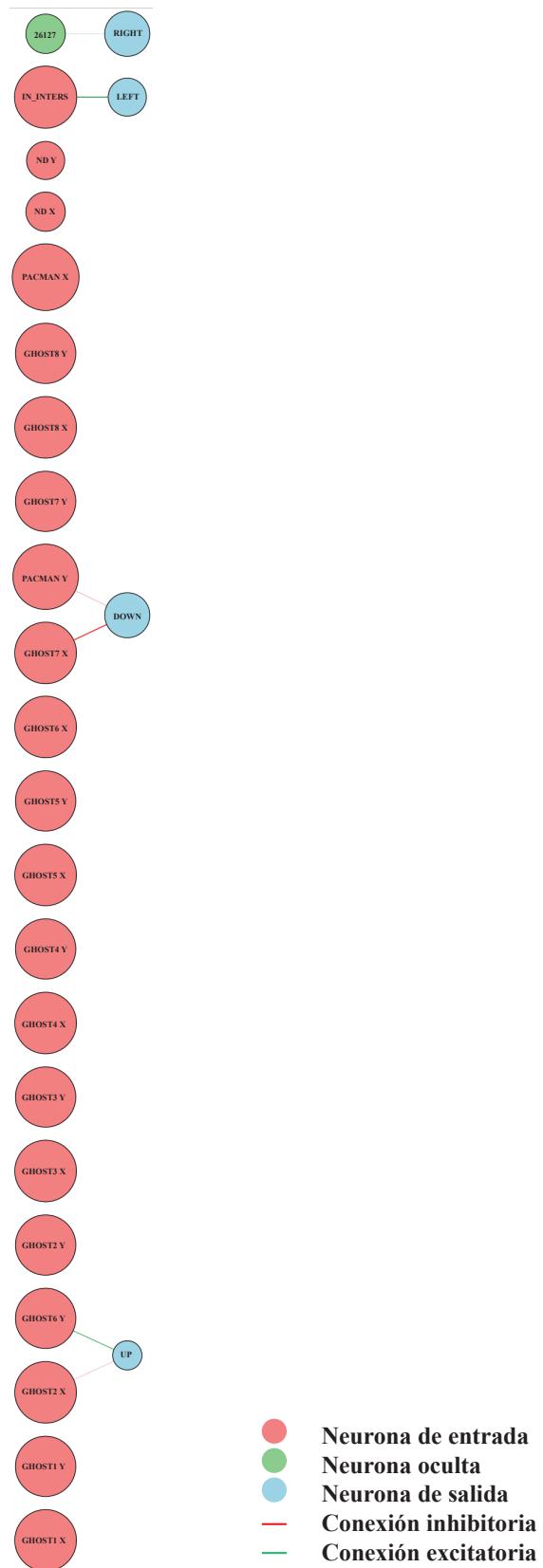
Evolución del fitness del modelo x-y-det-wmi con ocho fantasmas deterministas



Lo más destacable del resultado es la red subyacente al mejor individuo. Es la red más simple de todas las que se han visto en el estudio, sin embargo es la más efectiva en la tarea, al menos la tarea determinista. La red es capaz de conseguir el 90 % del mapa solo usando la coordenada X del segundo fantasma para alejarse de él con una conexión inhibitoria y la coordenada Y del sexto fantasma para determinar su dirección hacia arriba, la posición X del séptimo fantasma y su propia posición Y para determinar si baja, y finalmente IN_INTECS acaba conectado a la decisión de girar a la izquierda. La decisión de girar a la derecha es tomada por una constante en forma de umbral.

Figura 5.20

Red ganadora del modelo x-y-det-wmi con ocho fantasmas



5.4.1. Conclusión de la experimentación con fantasmas deterministas

En un entorno determinista la tarea se simplifica y se elimina el enorme ruido provocado por la aleatoriedad, por lo que es coherente que los resultados hayan mejorado drásticamente. Los resultados de la experimentación con 8 fantasmas deterministas y no deterministas se diferencian en más de 150 puntos, un 81,52 % del mapa.

En este entorno el algoritmo NEAT ha mostrado mejor su potencial, sobre todo su capacidad de optimizar la búsqueda, a la vez que optimiza su estructura. Demostrado en la Figura 5.20, NEAT es capaz de solucionar una tarea compleja como la resolución del mapa al 90 % con 8 fantasmas deterministas, usando solo 6 conexiones y 6 neuronas, siendo más simple que un modelo creado para solucionar el mapa con dos fantasmas o superando a modelos anteriores que habían necesitado más de 70 conexiones y más de 20 neuronas para obtener un porcentaje menor.

NEAT, con la red ganadora, ha demostrado que no hace falta conocer la posición de todos los fantasmas para solucionar el problema, al menos el determinista. Ha sido capaz de encontrar una relación casi invisible para el humano entre los movimientos de los pocos fantasmas que utiliza para el procesamiento de su decisión y el resto de fantasmas, asegurándose de que siguiendo esa relación, será capaz de completar el 90 % del mapa. Por tanto, la red es capaz de inferir los movimientos de todos los fantasmas a través de los movimientos de unos pocos.

6. GESTIÓN DEL PROYECTO

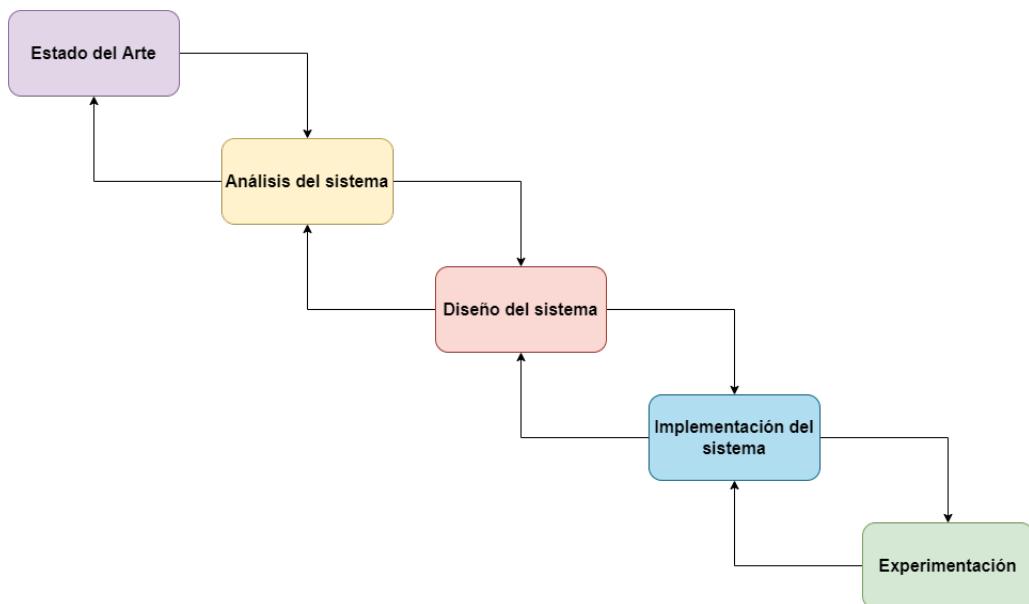
En este capítulo se describen los aspectos relativos a la gestión del proyecto. Específicamente, se explica la metodología usada, el desglose del presupuesto y se sitúa el trabajo en un entorno socioeconómico y un marco regulador.

6.1. Metodología

Para la realización de este trabajo se utilizó una metodología en cascada. La Figura 6.1 muestra un diagrama de la metodología en cascada aplicada a este estudio en particular.

Figura 6.1

Metodología del proyecto en cascada



La metodología en cascada establece diferentes etapas secuenciales. Sólo cuando finaliza una de ellas se empieza la siguiente, ayudando fijar metas específicas y mantener su seguimiento. Además, esta metodología permite revisiones, muy frecuentes en trabajos similares de experimentación con algoritmos, al terminar una etapa, lo que provoca que se vuelva a la etapa anterior y se hagan las modificaciones necesarias, dotando al proyecto con capacidad de corrección e innovación en su planteamiento.

6.2. Planificación

La planificación que se ha seguido durante este trabajo viene descrita en esta sección. En la tabla 6.1 se detallan la fecha de inicio, de finalización y las horas trabajadas en cada

Etapa	Fecha de inicio	Fecha de finalización	Duración (horas)
Planteamiento	02/12/2021	03/03/2022	60
Estado del arte	15/03/2022	30/04/2022	100
Análisis del sistema	02/05/2022	19/05/2022	50
Diseño del sistema	20/05/2022	10/06/2022	50
Implementación del sistema	11/06/2022	07/07/2022	120
Experimentación	08/07/2022	26/08/2022	200
Total del estudio	02/12/2021	03/09/2022	580
Documentación del estudio	15/03/2022	03/09/2022	400

Tabla 6.1

Planificación del proyecto

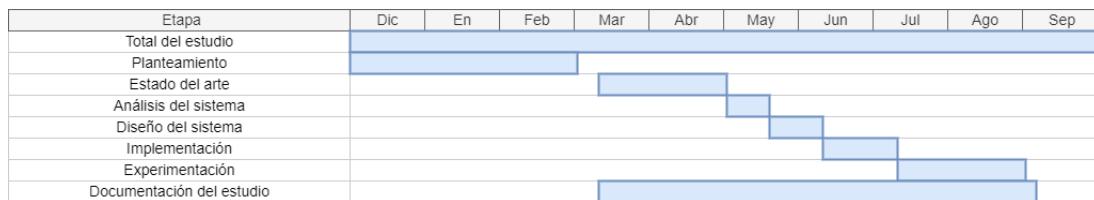
una de ellas.

En total, este trabajo ha durado 9 meses, con etapas con diferente carga de horas. Primero, se desarrolló un planteamiento del problema y del objeto a estudiar, para luego familiarizarse con él mediante la documentación de un Estado del arte. Tras esto se realizó un análisis y diseño del sistema, seguido de la implementación del mismo. Finalmente, se elaboró una experimentación exhaustiva del sistema. Desde el comienzo del Estado del arte, se empezó la documentación de todo el proyecto, hasta el final del mismo.

En la Figura 6.2 se muestra la planificación descrita anteriormente con un diagrama Gantt.

Figura 6.2

Diagrama de Gantt del estudio



6.3. Presupuesto

En esta sección se detalla el presupuesto necesario para llevar a cabo este proyecto. Para ello se desglosarán los costes en personal, material, indirectos y totales.

6.3.1. Costes de personal

El coste de personal dependerá de los roles asignados al proyecto. En este caso se han considerado 3 roles:

Rol	Coste/Hora	Horas trabajadas	Coste
Jefe de proyecto	25,18 €	20	503,6 €
Ingeniero analista	19,72 €	400	7.888 €
Ingeniero desarrollador	15,88 €	520	8.257,6 €
		Coste total	16.649,2 €

Tabla 6.2

Costes de personal

- Jefe de proyecto: Su función es supervisar, dirigir y coordinar el proyecto. En este caso, el rol se corresponde con el tutor.
- Ingeniero analista: Se encarga de la investigación necesaria para llevar a cabo el proyecto, además de realizar el análisis y diseño del sistema. Este rol ha sido adoptado por el alumno.
- Ingeniero desarrollador: Su función es implementar y probar el correcto funcionamiento del sistema, además de elaborar la experimentación. Este rol también ha sido adoptado por el alumno.

Una vez establecidos los diferentes roles, se calcula el coste mediante los salarios y las horas respectivos a cada rol. El coste por hora de cada rol se ha obtenido de la Dirección General de Trabajo, que registra y publica los Acuerdos de aprobación de las tablas salariales para el 18º Convenio de Empresas de Consultoría, Estudios de Mercado y de la Opinión Pública (Servicios, s.f.). En la tabla 6.2 se desglosan los costes por rol. Para el tiempo relativo a la documentación del proyecto se asume que el ingeniero analista y el ingeniero desarrollador se han repartido las horas a la mitad.

6.3.2. Costes de material

El material usado en el estudio ha consistido en herramientas de hardware y software. Para el cálculo de los costes del material utilizado se tendrá en cuenta el periodo de amortización del material y el periodo en el que se ha usado. Así, el coste se calculará de acuerdo a la siguiente ecuación:

$$Coste = \frac{Precio}{PeriodoAmortizacin(meses)} \times TiempodeUso(meses) \quad (6.1)$$

La tabla 6.3 muestra dicho cálculo de costes materiales.

Material	Precio	Periodo Amortización (meses)	Tiempo de Uso (meses)	Coste
Portátil HP Omen 15	899 €	36	6	149,83 €
Overleaf Standard	168 €	12	6	84 €
Monitor HP OMEN 25	239 €	36	6	39,83 €
			Coste total	
				273,66 €

Tabla 6.3

Costes de material

Concepto	Cantidad
Coste de Personal	16.649,2 €
Coste material	273,66 €
Total costes directos	16.922,86 €
Costes indirectos (15 %)	2.538,43 €
Margen de Beneficios (30 %)	5.076,86 €
Margen de Riesgos (10 %)	1.692,29 €
Total costes (sin IVA)	26.230,44 €
IVA (21 %)	5.508,39 €
Total Costes (con IVA)	31.738,83 €

Tabla 6.4

Costes Totales

En esta sección se desglosan los costes totales del proyecto. Para ello se tendrán en cuenta los costes anteriores, los costes indirectos, el margen de beneficios, el margen de riesgos y el IVA (Impuesto sobre el Valor Añadido), de acuerdo a los siguientes criterios:

- Se ha establecido un 15 % de **costes indirectos**. Estos incluyen gastos en luz o Internet.
- Se ha establecido un **margen de beneficios** de un 30 %.
- Se estima un **margin de riesgo** de un 15 %, en el que se cubren los posibles imprevistos como averías o adquisición de software nuevo.
- Se suma un 21 % de **IVA** al coste calculado.

En la tabla 6.4 se muestran los costes totales del proyecto. Finalmente, se obtiene que los costes totales del proyecto son 31.738,83 €.

6.4. Impacto socio-económico

Este trabajo ha consistido en la implementación y el análisis del algoritmo de Neuroevolución de Topologías Aumentadas en una abstracción de Pac-man. El fin del mismo era estudiar y profundizar en los conceptos de NEAT y de la neuroevolución en general, a través de la creación de un agente de IA. Este estudio podría utilizarse con fines sociales y económicos.

En el **ámbito social**, se podría usar el trabajo para motivar el uso de métodos neuroevolutivos como NEAT para crear agentes de IA capaces de solucionar problemas complejos o para optimizar sistemas que utilicen RNAs. Por ejemplo, muchos sistemas de clasificación de enfermedades usan las RNAs como algoritmo, (Li et al., 2014) o (Liang et al., 2018). Estas RNAs podrían reducir su coste computacional a la hora de hacer predicciones mediante la minimización de estructura que propone NEAT u otros métodos neuroevolutivos. También, sería de gran ayuda para tareas donde se deba trabajar en entornos no lineales y espacios de búsqueda grandes y complejos, donde el tiempo de ejecución es un factor importante, como por ejemplo los agentes de IA que controlan los robots de rescate (Davids, 2002).

Con el rápido crecimiento de sistemas que usan RNAs en todo el planeta, una tecnología que las minimice y las haga menos complejas computacionalmente significaría un descenso del tiempo de ejecución, y por tanto de recursos, en cada una de ellas, lo que a gran escala se podría traducir en un gran ahorro energético. Por tanto, se podría decir que el estudio analiza una tecnología con un **impacto medioambiental** positivo.

Sobre el **impacto económico** se podrían usar argumentos similares al impacto social y medioambiental detallados previamente. Sistemas de RNAs más eficientes computacional y temporalmente se traducen en menos gastos y mayores beneficios para las empresas que utilicen dichos sistemas. A su vez, con la proliferación de sistemas con RNAs profundas o RNAs convoluciones, mucho más costosas al ser estructuras complejas, algoritmos neuroevolutivos como el moderno DeepNEAT (Miikkulainen et al., 2017) podrían ser importantes para controlar los gastos indirectos de la empresa. Por otra parte, se ha visto en este estudio que estos algoritmos, en particular NEAT, son buenos como agentes de IA en videojuegos simples como Pac-man, Pong o Super Mario, por lo que empresas de videojuegos, sobre todo de videojuegos para móviles debido a su menor complejidad, podrían usar este algoritmo para desarrollar las IAs de sus productos, ya sea para que los jugadores humanos jueguen contra ellas, o para gestionar elementos de IA internos, como el comportamiento de NPCs (Non Playable Character).

6.5. Marco regulador

La Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y Garantía de los Derechos Digitales se aplica a cualquier tratamiento de datos personales

incluidos en un fichero digital (BOE, [s.f.-b](#)).

Esta ley establece diferentes normas para proteger los datos personales y los derechos digitales de los usuarios. Algunas de las normas de la ley son las siguientes:

- Los usuarios involucrados deben consentir la información que van a proporcionar.
- Deben tomarse estrictas medidas de seguridad de manera que se garantice la integridad de los datos.
- Los usuarios involucrados deben ser informados de la información que van a proporcionar y que, si así lo desean, pueden ordenar eliminar dichos datos.
- Es fundamental que los datos obtenidos de los usuarios sean sólo los necesarios para lograr los objetivos del trabajo.

El trabajo cumple todas las medidas ya que el agente de IA desarrollado no utiliza datos que vulneren los derechos de ninguna persona, al solo hacer uso de la información del entorno de Pac-man, sin acceder a datos internos del usuario.

La **Ley de la Ciencia, la Tecnología y la Innovación** (BOE, [s.f.-a](#)) pretende el establecimiento de un marco para el fomento de la investigación técnica y científica. Integra tanto en el ámbito público como privado planes y mecanismos para el desarrollo y la promoción de técnicas de I + D + I, tratando de promover inversiones más altas en proyectos de investigación en diferentes organismos.

Este trabajo, se incluye dentro del ámbito de la investigación, tecnología e innovación técnico. Por tanto, esta ley comprende este proyecto, pudiendo pretender la promoción de las técnicas utilizadas en el trabajo.

6.6. Gestión de riesgos

En todos los sistemas informáticos se encuentran vulnerabilidades que pueden provocar situaciones de riesgo para el proyecto o para las leyes acatados por el mismo. El plan de riesgos identifica estas vulnerabilidades en los sistemas, para que una vez se han detectado, se tomen las medidas necesarias para prevenir los problemas que pudiesen surgir.

La evaluación de riesgos incluye las siguientes actividades y acciones, tal y como se describe en la Norma ISO/IEC 27001:2005 de la familia de normas ISO 27000 para la seguridad informática:

- Identificación de activos
- Identificación de los requisitos legales y de negocio relevantes para la identificación de activos.

Riesgo	Nivel de riesgo	Acción	Plan de respuesta
Avería del portátil	Medio	Mitigar	Contratar servicio de mantenimiento
Baja del ingeniero analista	Medio	Evitar	Contratar personal de sustitución
Baja del ingeniero desarrollador	Medio	Evitar	Contratar personal de sustitución
Robo del equipo	Medio	Transferencia a terceros	Contratar seguro contra robos

Tabla 6.5
Gestión de riesgos

- Valoración de activos identificados, teniendo en cuenta los requisitos legales y de negocio identificados en el punto anterior, y el impacto de una pérdida de confidencialidad (C), disponibilidad (D) e integridad (I).
- Identificación de amenazas y vulnerabilidades para los activos identificados.
- Evaluación y cálculo del riesgo (alto, medio, bajo).

Por otro lado, tras la evaluación de riesgos, existen diversas acciones que se pueden tomar. En este proyecto se han vislumbrado las siguientes:

- Evitar que ocurra el riesgo modificando la planificación del proyecto de manera que se eliminen las causas del riesgo.
- Transferir a terceros en caso de que se tenga un seguro contra incendios, robos, etc.
- Mitigar el riesgo con la finalidad de disminuir su impacto lo máximo posible.
- Establecer un plan de contingencia en caso de que no se pueda evitar ni disminuir el riesgo.

En la tabla 6.5 se muestran los posibles riesgos contemplados y el plan de respuesta asociado.

7. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se presentan las conclusiones finales obtenidas en este estudio, además de ciertas reformas e ideas para la elaboración de trabajos relacionados con este.

7.1. Conclusiones

El objetivo principal de este trabajo ha sido la implementación y evaluación del algoritmo de Neuroevolución de Topologías Aumentadas para resolver una abstracción del juego de Pac-man, basada en la maximización de la puntuación en un mapa con fantasmas con movimientos no deterministas. A continuación se analiza cada uno de los subobjetivos marcados al comienzo de este estudio.

- **Obtener una configuración de RNA a través del algoritmo NEAT adecuada para jugar a la abstracción de Pac-man propuesta:** Este objetivo no se ha cumplido para un entorno no determinista, al no obtener agentes que tuvieran un comportamiento general que les permitiera esquivar a todos los fantasmas de manera dinámica, sin poder predecir sus movimientos. La cantidad de ruido que estos movimientos no deterministas añadían a la evaluación de los individuos de la población era inmenso.

Sin embargo, se propuso un entorno determinista donde los fantasmas seguían un comportamiento codicioso para acercarse a Pac-man con cada movimiento y acabar la partida. En este entorno sin ruido el algoritmo NEAT mostró su verdadero potencial, llegando a alcanzar más de un 90 % de puntuación en el mapa con 8 fantasmas. Además, el algoritmo ha sido capaz de demostrar sus características más especiales como la búsqueda, aparte de una red que maximice la aptitud, de una red que minimice su estructura, siendo capaz de resolver tareas deterministas complejas con redes muy simples.

- **Analizar y mostrar la evolución del algoritmo y de las RNAs obtenidas durante la ejecución:** Este objetivo se ha cumplido correctamente, mostrando un análisis exhaustivo de la evolución del desempeño de la población durante las generaciones de cada prueba, además de estudiar la estructura de las RNAs más efectivas para la solución de las diferentes tareas.
- **Sacar conclusiones sobre la relación entre las configuraciones de las RNAs y su desempeño posterior en el juego:** Este objetivo se ha cumplido satisfactoriamente analizando las estructuras de las RNAs creadas por el algoritmo. NEAT ha demostrado que a la hora de resolver tareas con RNAs, la estructura más óptima puede estar muy lejos de lo que parece intuitivo. Como se ha visto en la experimentación

determinista, la red óptima mínima usaba el 24 % de las entradas disponibles, reduciendo la complejidad de la red enormemente. Por ello, NEAT muestra lo cruciales que pueden ser los métodos de optimización de estructuras para reducir la complejidad computacional en el trabajo con redes. Como se hablaba al principio del estudio, y en parte lo que motivó el mismo, fue que uno de los métodos más usados para probar estructuras de RNAs sigue un proceso de prueba y error. Este estudio muestra que es un proceso ineficiente para muchas tareas. Por ejemplo, un humano nunca hubiera adivinado que para resolver la tarea determinista de 8 fantasmas sólo era necesaria la información de la posición de 3 fantasmas y la coordenada Y de sí mismo, formando una red con 6 conexiones, cuando lo intuitivo es que la red necesite un conocimiento total del entorno. Por tanto, nuestro propio poder de cálculo no es suficiente para predecir qué estructura necesitará una red, sobre todo para agentes de IA complejos. Utilizar métodos de optimización de estructuras proporciona redes más eficientes que pueden ser diferenciales en la complejidad computacional del sistema a largo plazo, como se ha visto en este estudio o en trabajos referenciados en el Estado del Arte. Sin embargo, tanto NEAT como otros métodos neuroevolutivos tienen una gran desventaja que puede ser importante para determinados proyectos, el largo tiempo de entrenamiento que necesitan debido al infinito espacio de búsqueda que forma la topología y los valores de los pesos de una red neuronal.

En conclusión, los objetivos del trabajo se han cumplido de manera general, habiendo analizado y experimentado con el algoritmo NEAT de manera exhaustiva hasta comprender sus entresijos y entender profundamente sus ventajas y sus desventajas.

7.2. Trabajos Futuros

En este estudio se ha tratado de utilizar el algoritmo NEAT para solucionar una abstracción del juego de Pac-man, en un entorno no determinista y otro determinista. En el entorno determinista el algoritmo ha tenido resultados excelentes debido al nulo ruido que afectaba a la función de evaluación. Sin embargo, en el caso no determinista este ruido era extremadamente alto, causando que el algoritmo bajara bastante su rendimiento. Por tanto, un adecuado trabajo futuro sería la mejora del sistema al lidiar con el no determinismo.

7.2.1. Mitigación de los efectos no deterministas a la función de evaluación

Esto se podría acatar de diversas maneras. Primero, se podría disponer de un servidor potente que permita al sistema ejecutar más generaciones del algoritmo, que junto a la capacidad intrínseca de las RNAs de lidiar con el ruido, podría resultar en buenos individuos a largo plazo. En este trabajo no se han utilizado servidores ya que se ha querido

mantener el estudio dentro de un marco técnico de potencia media, como un portátil, al tratarse de un sistema para un videojuego, que se debe jugar en ordenadores comerciales.

Sin embargo, la solución de alargar las generaciones del algoritmo no asegura dar resultados, ya que podría seguir estancado en generar individuos con estrategias estáticas que para movimientos específicos de los fantasmas tienen buenos resultados, pero que al variar dichos movimientos en la siguiente partida, su rendimiento caiga en picado.

Una solución más prometedora es tratar de mitigar el ruido del no determinismo. Para ello, se podría evaluar a los individuos en base a su puntuación en varias partidas, pero dentro de la misma generación. Entonces, se debería modificar el sistema para que un individuo juegue un número de partidas específico y tras ello sea evaluado con la puntuación media de las mismas. De esta manera, se penaliza a los individuos que por probabilidad han tenido una partida buena siguiendo una estrategia estática, pero luego caen en rendimiento al tener una estrategia inútil para partidas con otros movimientos de los fantasmas. Por el contrario, se recompensaría a aquellos modelos que tienen un rendimiento adecuado en varias partidas. La mitigación del ruido en la función de evaluación estabiliza el fitness landscape de la tarea, que antes era altamente volátil imposibilitando la obtención de mínimos locales o globales de confianza. De esta manera, la función de evaluación sería consistente, y por tanto podría guiar la búsqueda adecuadamente hacia los individuos que maximizan el objetivo.

La desventaja de esta reforma es la importantísima subida del coste computacional y temporal del algoritmo. Por ejemplo, se supone que para evaluar un individuo de una generación debe jugar 10 partidas consecutivas. Para una configuración con 150 individuos como las de este estudio, supondría 1.500 partidas por generación. Suponiendo 200 generaciones, sería un total de 300.000 partidas, siendo el tiempo de ejecución del algoritmo 10 veces mayor que el que se ha tenido en este estudio, ya bastante alto de por sí. Además, el tiempo usado escalaría proporcionalmente al número de partidas determinadas necesarias para evaluar. Sin embargo, esta reforma del sistema, sumado a un entrenamiento en servidor como se hablaba anteriormente, sería un adecuado trabajo futuro.

Al sistema comentado previamente se le podrían añadir además las reformas para lidiar con funciones de evaluación estocásticas que se recomiendan en (McDonnell et al., 1995). Se podría dividir la función de evaluación en diversas funciones que evaluen diferentes aspectos del individuo, para luego ordenar descendentemente cada individuo en relación con el resto de la población para cada función de aptitud, obteniendo una matriz de clasificaciones de estos individuos. La función de evaluación sería la mediana de los rangos de cada individuo, proporcionando el mejor individuo. Esta reforma es particularmente provechosa para el problema a tratar, ya que regula los outliers extremos que se puedan dar, mejorando aún más la atenuación del ruido.

7.2.2. Variación de entorno

Por otro lado, se podrían investigar variaciones en el entorno y sus efectos, añadiendo más fantasmas o añadiendo a éstos nuevos tipos de movimientos. También se podría variar la forma del mapa o devolverlo a su formato original, antes de ser reformado para este estudio. Aún más interesante sería la inclusión de las Power-pellets, unos puntos de comida que permiten a Pac-man acabar con los fantasmas durante un tiempo limitado.

Fuera de la abstracción, se podría aplicar el algoritmo NEAT a entornos reales complejos. Por ejemplo, se podría aplicar para crear agentes IA en robots de rescate para zonas destruidas o con escombros, donde el agente deba optimizar el tiempo de búsqueda de personas accidentadas.

7.2.3. Variación del algoritmo

También se podría estudiar las diferentes variaciones del algoritmo. Por ejemplo, seguir con el análisis y experimentación de los hiperparámetros que no han sido contemplados en este trabajo.

Otro trabajo futuro sería la implementación en el sistema de la Retro-propagación como algoritmo de aprendizaje de las RNAs generadas por NEAT, como propone (L. Chen y Alahakoon, 2006). Según este estudio el sistema L-NEAT propuesto mejora el rendimiento de NEAT para tareas de clasificación, por lo que se podría aplicar este sistema a predicciones de enfermedades, de materiales a través de la espectroscopia, o de fraudes bancarios entre otros. El algoritmo sería potencialmente útil si se pretende elaborar un modelo de clasificación de bajo coste computacional.

BIBLIOGRAFÍA

- Ahmed, M., Seraj, R. e Islam, S. M. S. (2020). The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8), 1295.
- Ahumada, T. y Bergel, A. (2020). Reproducing Bugs in Video Games using Genetic Algorithms. *2020 IEEE Games, Multimedia, Animation and Multiple Realities Conference (GMAX)*, 1-6. <https://doi.org/10.1109/GMAX49668.2020.9256837>
- Angeline, P. J., Saunders, G. M. y Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Netw.*, 5(1), 54-65.
- Arulkumaran, K., Cully, A. y Togelius, J. (2019). AlphaStar: An Evolutionary Computation Perspective. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 314-315. <https://doi.org/10.1145/3319619.3321894>
- Athey, S. (2018). The impact of machine learning on economics. *The economics of artificial intelligence: An agenda* (pp. 507-547). University of Chicago Press.
- Azuelo, G. (2022). Pacman in Python code. <https://sourcecodehero.com/pacman-in-python-code/>
- Baluja, S. (1994). *Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning* (inf. téc.). USA, Carnegie Mellon University.
- Basu, J. K., Bhattacharyya, D. y Kim, T.-h. (2010). Use of artificial neural network in pattern recognition. *International journal of software engineering and its applications*, 4(2).
- Blasco Millan, D. (2019). *Estudio del algoritmo NEAT aplicado al videojuego Pac-Man* (B.S. thesis). Universitat Politècnica de Catalunya.
- BOE. (s.f.-a). Agencia Estatal. Boletín Oficial del Estado. Ley 14/2011, de 1 de junio, de la Ciencia, la Tecnología y la Innovación. <https://www.boe.es/buscar/act.php?id=BOE-A-2011-9617>
- BOE. (s.f.-b). Agencia Estatal. Boletín Oficial del Estado. Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales. <https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673>
- Braun, H. y Weisbrod, J. (1993). Evolving Neural Feedforward Networks. En R. F. Albrecht, C. R. Reeves y N. C. Steele (Eds.), *Artificial Neural Nets and Genetic Algorithms* (pp. 25-32). Springer Vienna.
- Channon, A. y Damper, R. (1998). Evolving novel behaviors via natural selection. *Proceedings of Artificial Life VI, Los Angeles*, 384-388.
- Chaves, P. y Chang, F.-J. (2008). Intelligent reservoir operation system based on evolving artificial neural networks. *Advances in Water Resources*, 31, 926-936. <https://doi.org/10.1016/j.advwatres.2008.03.002>

- Chen, J. y Jenkins, W. K. (2017). Facial recognition with PCA and machine learning methods. *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 973-976.
- Chen, L. y Alahakoon, D. (2006). NeuroEvolution of Augmenting Topologies with Learning for Data Classification. *2006 International Conference on Information and Automation*, 367-371. <https://doi.org/10.1109/ICINFA.2006.374100>
- Coello Coello, C., Rudnick, M. y Christiansen, A. (1994). Using genetic algorithms for optimal design of trusses. *Proceedings Sixth International Conference on Tools with Artificial Intelligence. TAI 94*, 88-94. <https://doi.org/10.1109/TAI.1994.346509>
- Cunningham, P., Cord, M. y Delany, S. J. (2008). Supervised learning. *Machine learning techniques for multimedia* (pp. 21-49). Springer.
- Darwin, C. (1909). *The origin of species*. PF Collier & son New York.
- Dasgupta, D. y McGregor, D. (1992). Designing application-specific neural networks using the structured genetic algorithm. *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, 87-96. <https://doi.org/10.1109/COGANN.1992.273946>
- Davids, A. (2002). Urban search and rescue robots: from tragedy to technology. *IEEE Intelligent Systems*, 17(2), 81-83. <https://doi.org/10.1109/MIS.2002.999224>
- Donate, J. P., de Miguel, A. S. y Sánchez, G. G. (2012). Diseño automático de redes de neuronas artificiales para la predicción de series temporales.
- Elsken, T., Metzen, J.-H. y Hutter, F. (2017). Simple And Efficient Architecture Search for Convolutional Neural Networks. <https://doi.org/10.48550/ARXIV.1711.04528>
- Flinton, G. y Sejnowski, T. (1999). Unsupervised learning and map formation: Foundations of neural computation.
- Floreano, D., Dürr, P. y Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary intelligence*.
- Floreano, D. y Mattiussi, C. (2008). Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies.
- Fullmer, B. y Miikkulainen, R. (1991). Using Marker-Based Genetic Encoding Of Neural Networks To Evolve Finite-State Behaviour.
- Gallagher, M. y Ryan, A. (2003). Learning to play Pac-Man: an evolutionary, rule-based approach. *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, 4, 2462-2469 Vol.4. <https://doi.org/10.1109/CEC.2003.1299397>
- Gomez, F. J., Miikkulainen, R. et al. (1999). Solving non-Markovian control tasks with neuroevolution. *IJCAI*, 99, 1356-1361.
- Gomez, F. J. (2003). *Robust non-linear control through neuroevolution*. The University of Texas at Austin.
- Gour, R. (2019). Python genetic algorithms with Artificial Intelligence. <https://medium.com/@rinu.gour123/python-genetic-algorithms-with-artificial-intelligence-b8d0c7db60ac>

- Granter, S. R., Beck, A. H. y Papke Jr, D. J. (2017). AlphaGo, deep learning, and the future of the human microscopist. *Archives of pathology & laboratory medicine*, 141(5), 619-621.
- Gruau, F., Whitley, D. y Pyeatt, L. (1996). A Comparison between Cellular Encoding and Direct Encoding for Genetic Neural Networks. *Proceedings of the 1st Annual Conference on Genetic Programming*, 81-89.
- Hochreiter, S. y Schmidhuber, J. (1997). Long Short-term Memory. *Neural computation*. <https://doi.org/10.1162/neco.1997.9.8.1735>
- John-Trager. (2022). neat-pong: Machine learning pong AI using neat. <https://github.com/John-Trager/NEAT-Pong>
- Kitano, H. (1990a). Designing Neural Networks Using Genetic Algorithms with Graph Generation System. *Complex Systems*, 4, 461-476.
- Kitano, H. (1990b). Empirical Studies on the Speed of Convergence of Neural Network Training Using Genetic Algorithms. *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 2*, 789-795.
- Kononenko, I. (2001). Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in medicine*, 23(1), 89-109.
- Kubat, M. (1999). Neural networks: a comprehensive foundation by Simon Haykin. *The Knowledge Engineering Review*.
- Li, Q., Cai, W., Wang, X., Zhou, Y., Feng, D. D. y Chen, M. (2014). Medical image classification with convolutional neural network. *2014 13th international conference on control automation robotics & vision (ICARCV)*, 844-848.
- Liang, G., Hong, H., Xie, W. y Zheng, L. (2018). Combining convolutional neural network with recursive neural network for blood cell image classification. *IEEE access*, 6, 36188-36197.
- Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *Machine learning proceedings 1994* (pp. 157-163). Elsevier.
- Mahfoud, S. W. (1995). *Niching Methods for Genetic Algorithms* (inf. téc.).
- Mandischer, M. (1994). Representation and Evolution of Neural Networks. https://doi.org/10.1007/978-3-7091-7533-0_93
- Maqsood, I., Khan, M. R. y Abraham, A. (2004). An ensemble of neural networks for weather forecasting. *Neural Computing & Applications*.
- McDonnell, J. R., Reynolds, R. G. y Fogel, D. B. (1995). Optimizing Stochastic and Multiple Fitness Functions. *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming* (pp. 127-134).
- McIntyre, A., Kallada, M., Miguel, C. G., Feher de Silva, C. y Netto, M. L. (s.f.). *neat-python*.
- Merrill, J. W. y Port, R. F. (1991). Fractally configured neural networks. *Neural Networks*, 4(1), 53-60. [https://doi.org/https://doi.org/10.1016/0893-6080\(91\)90031-Y](https://doi.org/https://doi.org/10.1016/0893-6080(91)90031-Y)
- Miikkulainen, R., Liang, J. Z., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N. y Hodjat, B. (2017). Evolving Deep Neural Networks. *CoRR*.

- Miller, G. F., Todd, P. M. y Hegde, S. U. (1989). Designing Neural Networks Using Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, 379-384.
- Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.
- Monnier, Y., Beauvais, J.-P. y Deplanche, A.-M. (1998). A genetic algorithm for scheduling tasks in a real-time distributed system. *Proceedings. 24th EUROMICRO Conference (Cat. No.98EX204)*, 2, 708-714 vol.2. <https://doi.org/10.1109/EURMIC.1998.708092>
- Montana, D. J. y Davis, L. (1989). Training Feedforward Neural Networks Using Genetic Algorithms. *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, 762-767.
- Nygaard, T. (2015). *Evolutionary optimization of robot morphology and control - Using evolutionary algorithms in the design of a six legged robotic platform* (Tesis doctoral).
- Parvathy, P., Subramaniam, K., Prasanna Venkatesan, G., Karthikaikumar, P., Varghese, J. y Jayasankar, T. (2021). Development of hand gesture recognition system using machine learning. *Journal of Ambient Intelligence and Humanized Computing*, 12(6), 6793-6800.
- Pepels, T., Winands, M. y Lanctot, M. (2014). Real-Time Monte-Carlo Tree Search in Ms Pac-Man. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6, 245-257. <https://doi.org/10.1109/TCIAIG.2013.2291577>
- Pujol, J. C. F. y Poli, R. (2004). Evolving the Topology and the Weights of Neural Networks Using a Dual Representation. *Applied Intelligence*, 8, 73-84.
- Pygame documentation. (s.f.). <https://www.pygame.org/docs/ref/time.html#pygame.time.Clock>
- Ruff, K. M. y Pappu, R. V. (2021). AlphaFold and implications for intrinsically disordered proteins. *Journal of Molecular Biology*, 433(20), 167208.
- Russell, S. y Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd). Prentice Hall Press.
- Servicios. (s.f.). Plataforma de CCOO para el 18º Convenio de Empresas de Consultoría, Estudios de Mercado y de la Opinión Pública (TIC). <https://www.ccoo-servicios.es/tic/pagweb/3522.html>
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*.
- Skinner, G. y Walmsley, T. (2019). Artificial intelligence and deep learning in video games a brief review. *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, 404-408.
- Stacy. (2020). Deepmind's AI, AlphaStar showcases significant progress towards AGI. <https://medium.com/@towardai/deepminds-ai-alphastar-showcases-significant-progress-towardsagi-93810c94fbe9>

- Stanley, K., Kohl, N., Sherony, R. y Miikkulainen, R. (2005). Neuroevolution of an Automobile Crash Warning System, 1977-1984. <https://doi.org/10.1145/1068009.1068340>
- Stanley, K. O., D'Ambrosio, D. B. y Gauci, J. (2009). A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life*, 15(2), 185-212. <https://doi.org/10.1162/artl.2009.15.2.15202>
- Stanley, K. O. y Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*.
- Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O. y Clune, J. (2017). Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.
- Sultana, F., Sufian, A. y Dutta, P. (2018). Advancements in image classification using convolutional neural network. *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*.
- Techwithtim. (2019). An ai that plays flappy bird! using the NEAT python module. <https://github.com/techwithtim/NEAT-Flappy-Bird>
- Thawonmas, R. y Matsumoto, H. (2009). Automatic Controller of Ms. Pac-Man and Its Performance: Winner of the IEEE CEC 2009 Software Agent Ms. Pac-Man Competition.
- TIBCO. (s.f.). What is a neural network? <https://www.tibco.com/reference-center/what-is-a-neural-network>
- Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Dabihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., ... y Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782), 350-354. <https://doi.org/10.1038/s41586-019-1724-z>
- Viñuela, P. I. y León, I. M. G. (2004). *Redes de neuronas artificiales: un enfoque práctico*. Prentice Hall.
- Watson, I. D., Azhar, D., Chuyang, Y., Pan, W. y Chen, G. Y.-H. (2009). Optimization in Strategy Games : Using Genetic Algorithms to Optimize City Development in FreeCiv.
- White, C., Neiswanger, W. y Savani, Y. (2019). BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. *CoRR*. <http://arxiv.org/abs/1910.11858>
- Whitley, D. y Hanson, T. (1989). Optimizing Neural Networks Using Faster, More Accurate Genetic Search. *Proceedings of the Third International Conference on Genetic Algorithms*, 391-396.
- Wright, S. et al. (1932). The roles of mutation, inbreeding, crossbreeding, and selection in evolution.
- wts42. (2018). Neat-genetic-mario. <https://github.com/wts42/Neat-Genetic-Mario>

- Xin, M. y Wang, Y. (2019). Research on image classification model based on deep convolution neural network. *EURASIP Journal on Image and Video Processing*, 2019(1), 1-11.
- Yannakakis, G. N. y Hallam, J. (2004a). Evolving opponents for interesting interactive computer games.
- Yannakakis, G. N. y Hallam, J. (2004b). Interesting games through stage complexity and topology.
- Zhai, W., Kelly, P. y Gong, W.-B. (1996). Genetic algorithms with noisy fitness. *Mathematical and Computer Modelling*. [https://doi.org/https://doi.org/10.1016/0895-7177\(96\)00068-4](https://doi.org/https://doi.org/10.1016/0895-7177(96)00068-4)
- Zhang, B.-T., Muhlenbein, H. et al. (1993). Evolving optimal neural networks using genetic algorithms with Occam's razor. *Complex systems*, 7(3), 199-220.
- Zhang, J., Zhang, Y. y Gao, R. (2006). Genetic Algorithms for Optimal Design of Vehicle Suspensions. *2006 IEEE International Conference on Engineering of Intelligent Systems*, 1-6. <https://doi.org/10.1109/ICEIS.2006.1703182>

ANEXO

A. Summary

1. Motivation

In the field of computer science, different attempts have been made to imitate the human learning process in a number of ways. One of them is by getting inspiration from the natural structures that enable it, such as the brain or genetics. This is known as Biologically Inspired Artificial Intelligence. Within these methods, there are two main paradigms: genetics and artificial neural networks. Both genetic algorithms and ANNs are techniques that are currently very useful in a variety of fields. In the case of the latter, their ability to solve non-linear problems and their tolerance to noise make them a very powerful technique for learning certain types of tasks or patterns, for example, image classification or detection of diseases.

However, one of the problems with ANNs is that in order to be able to solve a problem effectively, their design must be appropriate. Thus, for a specific problem, the design of the network can drastically vary the outcome. Therefore, it is necessary to determine network parameters, such as network type or network architecture (topology and connection weights) in an optimal way in order to obtain the most complete solution.

Usually, the choice of parameters is made by an heuristic of trial and error. The network is tested with different configurations and a training and validation error are obtained for each configuration, which will indicate which network generalises best for the given problem. This process is inefficient and costly. In addition, some suitable configurations may be discarded due to the huge solution space that comprises all possible combinations of parameters of a network.

One of the most interesting methods to address this disadvantage is Neuroevolution. This form of artificial intelligence uses genetic algorithms (GAs) to generate different ANNs with different configurations, until the one that best suits the proposed task is found under the measure of a fitness function.

In this work, the algorithm Neuroevolution of Augmenting Topologies (NEAT) has been selected as the object of study. This decision was taken because it is a paradigmatic algorithm that can be used to understand most of the concepts of Neuroevolution. Moreover, it has interesting features and good results that make it one of the most famous methods in this field.

2. Problem's description

In order to test the capabilities of NEAT, it will be tested in an abstraction of the famous Pac-man game. It is referred to as an abstraction because several concepts from the original game will be used, but modified in order to be able to analyse other behaviours of NEAT. The main modifications made to the original game are:

- Enemies: Instead of 4, there will be 8.
- Enemy movements: The movements of the ghosts will not be deterministic, they will choose their movements randomly. This will be a critical change in the learning process of the algorithm, as will be seen later in the study.
- Enemy death: There will be no Power-Pellets, so Pac-man can't eat the ghosts at all.

As can be inferred from the changes in the rules, the game of Pac-man has been complicated into a task of dodging numerous enemies whose movement pattern is unpredictable. This makes the algorithm unable to define an unique optimal path to avoid the ghosts, but must learn more abstract behaviour such as dodging dynamically and waiting for enemy movements, while maximising its score. Therefore, Pac-man's goal is to eat the maximum amount of points on the map without being eaten by one of the eight ghosts.

3. Objectives

The goal is to implement a system that can learn to maximise Pac-man's score without colliding with the ghosts.

This system will be a model of an ANN that will have been optimised by the NEAT algorithm, so that it can perform adequately in the Pac-man task.

In this study, 3 main objectives have been proposed, which aim, apart from obtaining a good result in the modified Pac-man game, is to gain knowledge and understand the depths of NEAT and Neuroevolution in general. These objectives are listed below:

- Obtain an ANN configuration through the NEAT algorithm suitable for playing the proposed Pac-man abstraction.
- Analyse and show the evolution of the algorithm and the ANNs obtained during the execution.
- Draw conclusions on the relationship between ANNs configurations and their subsequent in-game performance.

4. Implementation

Firstly, in order to implement the algorithm to the Pac-man environment, the source code suffered a variety of changes:

- Elimination of the start menu: So that training could be automated easily, the main menu, where you could select whether to quit the game or start a new game, was eliminated. After the change, the game starts and when it reaches a state of completion, the process is closed directly, without going through the main menu. This allows the automation of chained games in a more efficient way.
- Termination states: In the original version, the game only ends when Pac-man collides with a ghost, returning to the main menu. In order to speed up the training process so that it doesn't get stuck, two new end conditions have been added. The game will then end if:
 - Pac-man collides with a ghost.
 - Pac-man obtains all points of the map.
 - If end condition 1 or 2 has not been met, the game ends if a time limit set by the user is exceeded.
- Game speed increase: The original version had an imposed limit of 30 frames per second (FPS), in order to be assimilated by a human being's vision and reflexes. At automating the task and changing the agent to an ANN, 30 FPS slows down the time per generation extremely. To address this problem, a limit of 600 FPS was placed on the game. In general, due to the power of the computer where the training is run, the FPS vary around 300 - 400, accelerating the pace of the training by more than 10 times.
- Change in the initial map: The complexity of capturing Pac-man is inversely proportional to the length of the map corridors. Therefore, in the case of this study, the complexity of the map will be given by:

$$C = M\{P\}$$

where MP is the average length of the map corridors. Under this measure of complexity, the original map was quite complex, which could be verified simply by playing around with the controls. To simplify the initial task, new corridors were added to the map, shortening the length of the previous ones.

After changing the source code, the information that is provided to the agent has to be decided. Different configurations of inputs and outputs for the RNAs generated were analysed, tested and implemented. The two input configurations used in this study are the following:

- **Coordinates:** In this format, the information that is provided to Pac-man is the x and y coordinates of the ghosts, himself and the nearest dot. x and y coordinates of the ghosts, itself and the nearest dot. Therefore, each of the individual inputs would be associated with an input neuron in the network. This format needs 20 input neurons for the ANN.
- **Distance - corridor:** In an attempt to provide the agent with a more informed and easier to interpret view of the environment than the abstraction of a set of coordinates, a system was designed to inform the agent of the distance (euclidean or manhattan) to Pac-man of the elements in the map and the corridor in which the map elements were located. In this way, the agent can learn to dodge the ghosts without having to deal with coordinates, which can be more time-consuming and complex. Instead, it must learn the relationships between corridors and the distances from the ghosts to him. In conclusion, the agent receives, for each ghost and for the dot closest to Pac-man, the distance from the player and the corridor in which it is. This format needs 19 input neurons for the ANNs.
- **Supporting inputs:** To the two sets of available inputs, extra support inputs are added to help the agent be sensitive to his own movement. Specifically, the inputs indicate, in a binary way, in which direction Pac-man is moving and whether is at an intersection, since this is the place on the map that allows a change from horizontal to vertical movement or vice versa. That results in ANNs using a format of coordinates needing 25 inputs neurons, and the ones using the distance - corridor format, 24 input neurons.

The output neurons will control the movement of the agent. Specifically, there will be 4 output neurons in all ANNs created. Each output neuron indicates a direction that Pac-man can take: Up, down, right, left. In the middle of the execution, the output neuron with the maximum value will be the one that changes Pac-man direction.

Other crucial element of the implementation is the fitness function of the algorithm, that is, the function in charge of measure which ANNs of the total population of ANNs in the algorithm are the best for the Pac-man task. After analysing and experimenting with different functions, these are the ones that were more important throughout the project.

- **Score and duration:** It takes into account the score (number of dots eaten by Pac-man) and the time Pac-man stayed alive. In case it gets the maximum score for the map, the time it took to complete the task is saved as the duration. This fitness function tries to reward high-scoring ANNs, while penalising those with a long duration. However, this approach did not work very well, as there were models with high score, and consequently more duration, that get low evaluated. Therefore, it was assigned a weight to the importance of the duration when calculating the fitness.

$$f = score - \alpha * duration_{individuo}$$

- **Score:** This fitness function is simply $f = score$, but it is very powerful at distinguish between bad and good ANNs. Moreover, the previous evaluation functions slowed down an already rather slow training process, due to the fact that there were individuals with bad but high consuming strategies, such as standing still or only moving in one direction. Therefore, it was decided not to use time in the evaluation calculation, at least explicitly. A new counter was implemented which timed how long the agent had been without scoring. If it exceeded a certain time limit set by the user, the individual's execution was stopped and its fitness was calculated.
- **Individual average:** Given the non-deterministic nature of ghosts' movements, an individual following a specific succession of steps could achieve more or less score depending on whether a ghost randomly crosses its path. This adds noise to the evaluation of the individuals and makes the evaluation function less consistent, as the same individual in the same generation can obtain different values in the fitness function. Thus, accuracy in selecting the best individual in each generation is lost. Therefore, the fitness function was calculated as the individual's average fitness history, that is, the average of all fitness value obtained for the same individual throughout the generations. The problem with this fitness function is that individual did not live enough time in the generation, so the results were very similar as using the score fitness function.

Finally, the parameters of the algorithm, mainly the parameters of the GA part, could be modified through the configuration file (see Appendix). For example, it allows to change and test different population sizes, objective functions, initial connections for the ANNs, probabilities of mutation in the structure of the ANNs, etc.

5. Experimentation

The experimentation was divided in three phases, due to the complexity of the non-deterministic Pac-man task.

- **Experimentation with 8 non-deterministic ghosts:** In this first section of experimentation, an initial contact with the problem has been established, demonstrating that the current method is not enough to solve it correctly. The algorithm, due to the random nature of the problem, suffers from a lot of noise in the evaluations. The evaluation function is not able to measure which individuals are good and which are not, because the fitness landscape is highly dynamic, and in each game it is different, causing that a strategy that is promising in the first game, not to be promising in a second game or third game. This causes the evaluation function to be inconsistent and the algorithm to execute a blind random search.

In order to try to help the agent to learn correctly, a progressive experimentation was implemented, where Pac-man will learn gradually the task, increasing the number of ghosts in the map sequentially, starting in 0.

- **Progressive experimentation with non-deterministic ghosts:** In this experimentation we have seen that the problem existing in the experimentation with 8 ghosts persists, seeing how performance is severely affected by the inclusion of only 2 ghosts in the map. These two ghosts are already capable of contributing enough noise to the evaluation function to slow down the search. In addition, the algorithm is still trying to find simple strategies that are able to achieve a decent score most of the time. However, the last generated network has been the closest thing to a ghost-aware agent, despite not using its full coordinates and therefore not being able to dodge them in all contexts. Still, the best individuals are those who make good use of the supporting inputs, as they allow the agent to create static routes with a low probability of death, although they are still far from being adequate agents, not even reaching 40 % of the map total score.

Therefore, a last phase of experimentation was implemented, in order to test NEAT in a static environment, that is, a deterministic one.

- **Experimentation with deterministic ghosts:** The ghost's movement pattern was changed from random, to a greedy algorithm that tried, in each step, to minimize the distance between Pac-man and the ghost. This makes the ghosts behaviour more aggressive, but more predictable.

In a deterministic environment the task is simplified and the enormous noise caused by randomness is eliminated, so it is consistent that the results have improved drastically. The results of the experimentation with 8 deterministic and non-deterministic ghosts differ by more than 150 points, 81.52 % of the map total. In this environment, the NEAT algorithm has shown its potential, especially its ability to optimise the search, while optimising its structure. NEAT is able to solve a complex task such as the Pac-man task with 90 % map resolution with 8 ghosts, using only 6 connections and 6 neurons. Thus, the winner ANN is simpler than a model created to solve the map with two ghosts, also surpassing previous models that had needed more than 70 connections and more than 20 neurons to obtain a lower percentage.

6. Socio-economic impact

In the social domain, the work could be used to motivate the use of neuroevolutionary methods such as NEAT to create AI agents capable of solving complex problems or to optimise systems using ANNs. For example, many disease classification systems use ANNs as an algorithm. These ANNs could reduce their computational cost when making predictions through the minimisation of structure proposed by NEAT or other neuroevolutionary methods. Also, it would be of great help for tasks where it is necessary to work on non-linear environments and large and complex search spaces, where the execution time is an important factor, such as AI agents controlling rescue robots.

With the fast growth of systems using ANNs across the globe, a technology that mi-

nimises them and makes them less computationally complex would mean a decrease in runtime, and therefore resources, for each of them, which on a large scale could translate into large energy savings. Therefore, it could be said that the study analyses a technology with a positive environmental impact.

Similar facts could be used for the economic impact as for the social and environmental impact detailed above. More computationally and temporally efficient ANN systems translate into lower costs and higher profits for the companies using such systems. In turn, with the proliferation of systems with deep ANNs or convolutional ANNs, which are much more costly as they are complex structures, neuroevolutionary algorithms such as the state-of-the-art DeepNEAT could be crucial to control the company's overhead costs. On the other hand, it has been seen in this study that these algorithms, in particular NEAT, are good as AI agents in simple video games such as Pac-man, Pong or Super Mario, so that video game companies, especially mobile game companies due to their lower complexity, could use this algorithm to develop AIs for their products, either for human players to play against them, or to manage internal AI elements, such as the behaviour of NPCs (Non Playable Character).

Regulatory framework

The work complies with all measures defined on *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y Garantía de los Derechos Digitales*, applicable to any processing of personal data contained in a digital file, as the developed AI agent does not use data that violates the rights of any person, as it only makes use of information from the Pac-man environment, without accessing internal user data.

The *Ley de la Ciencia, la Tecnología y la Innovación* aims to establish a framework for the promotion of technical and scientific research. It integrates both public and private plans and mechanisms for the development and promotion of I + D + I techniques, trying to promote higher investments in research projects in different bodies.

In addition, this study is included within the scope of research, technology and technical innovation. Therefore, this law includes this project, being able to get the promotion of the techniques used in the work.

8. Conclusions

Each of the sub-objectives outlined at the beginning of this study is discussed below:

- **Obtain an ANN configuration through the NEAT algorithm suitable for playing the proposed Pac-man abstraction:** This objective was not met for a non-deterministic environment, as the algorithm did not obtain agents who had a general behaviour that allowed them to avoid all the ghosts dynamically, without being able

to predict their movements. The amount of noise that these non-deterministic movements added to the evaluation of the individuals in the population was immense.

However, a deterministic environment was proposed where the ghosts followed a greedy behaviour to approach Pac-man with each move and finish the game. In this noiseless environment the NEAT algorithm showed its true potential, reaching more than 90 % score on the map with 8 ghosts. Furthermore, the algorithm has been able to demonstrate its special features such as finding, apart from a network that maximises fitness, a network that minimises its structure, being able to solve complex deterministic tasks with very simple networks.

- **Analyse and show the evolution of the algorithm and the ANNs obtained during the execution:** This objective has been successfully achieved, showing an exhaustive analysis of the evolution of the performance of the population during the generations of each test, as well as studying the structure of the most effective ANNs for the solution of the different tasks.
- **Draw conclusions on the relationship between ANNs configurations and their subsequent in-game performance:** This objective has been successfully achieved by analysing the structures of the ANNs created by the algorithm. NEAT has shown that when solving ANN tasks, the most optimal structure can be far from what seems intuitive. As seen in the deterministic experimentation, the minimum optimal network used 24 % of the available inputs, reducing the complexity of the network enormously.

Thus, NEAT shows how crucial structure optimisation methods, whether Neuroevolution or other, can be in reducing computational complexity when working with networks. As discussed at the beginning of the study, and in part what motivated the study, one of the most commonly used methods for testing RNA structures follows a trial-and-error process. This study shows that it is an inefficient process for many tasks. The human own computational power is not sufficient to predict what structure a network will need, especially for complex AI agents. Using structure optimisation methods provides more efficient networks that can be differential in the computational complexity of the system in the long run. However, both NEAT and other neuroevolutionary methods have a major disadvantage that can be significant for certain projects, the long training time they require due to the infinite search space that forms the topology and the values of the weights of a neural network.

9. Future work

In this study NEAT algorithm has been used to solve an abstraction of the Pac-man game, in a non-deterministic and a deterministic environment. In the deterministic environment the algorithm had excellent results due to the zero noise affecting the evaluation function. However, in the non-deterministic case this noise was extremely high, causing

the algorithm to lower its performance considerably. Therefore, a suitable future work would be to improve the system when dealing with non-determinism.

The strategies for mitigating the noise of non-determinism in the fitness function are plenty. The following are a sample of what could be implemented:

- A powerful server could be available to allow the system to run more generations of the algorithm, which together with the intrinsic ability of ANNs to deal with noise, could result in good individuals in the long run. No servers have been used in this work because it was chose to keep the study within a medium-powered technical framework, such as a laptop, as this is a system for a video game, which must be played on commercial computers.

However, the solution of lengthening the generations of the algorithm is not guaranteed to give results, as it could remain stuck in generating individuals with static strategies.

- Individuals could be evaluated based on their score in several games, but within the same generation. Then, the system should be modified so that an individual plays a specific number of games and is then evaluated on the average score of those games. This would reward those models that perform adequately over several games. Mitigating the noise in the evaluation function stabilises the fitness landscape of the task, which was previously highly volatile making it impossible to obtain local or global minima in confidence range. In this way, the evaluation function would be consistent, and could therefore guide the search appropriately towards the individuals that maximise the objective.

The disadvantage of this is the significant increase in the computational and time cost of the algorithm.

- To the system discussed above, it could also be added the modifications for dealing with stochastic fitness functions. One could divide the evaluation function into several functions that evaluate different aspects of the individual, and then rank each individual relative to the rest of the population for each fitness function, obtaining a matrix of rankings of these individuals. The evaluation function would be the median of the ranks of each individual, providing the best individual. This reform is particularly beneficial for the problem, as it regulates the extreme outliers that may occur, further improving noise attenuation.

On the other hand, apart of the mitigation of the effects of non-determinism, more studies and projects could emerge from this one. In this study, two branchs have been considered.

- **Change of environment:** Variations in the environment and its effects could be investigated by adding more ghosts or adding new types of movement to the ghosts. The shape of the map could also be changed or returned to its original format, before

being reworked for this study. Also Power-pellets could be included, food dots that allow Pac-man to kill ghosts for a limited time, present in the original Pac-man games.

Outside of Pac-man, the NEAT algorithm could be applied to complex real-world environments. For example, it could be applied to create AI agents in rescue robots for destroyed or ruined areas, where the agent must optimise the time spent searching for injured people.

- **Changes in the algorithm:** Different variations of the algorithm could also be studied. For example, the analysis and experimentation of parameters that have not been considered in this work could be continued.

Another future work would be the implementation in the system of Retro-propagation as a learning algorithm for the ANNs generated by NEAT, as proposed by (L. Chen y Alahakoon, 2006). According to this study, the proposed L-NEAT system improves the performance of NEAT for classification tasks, so this system could be applied to disease prediction, materials prediction through spectroscopy, or bank fraud, among others. The algorithm would be potentially useful if a classification model with strong restrictions in the computational complexity is to be developed.

In conclusion, the objectives of the work have been met in a general way, having analysed and experimented with the NEAT algorithm in an exhaustive way to understand its mechanisms and to deeply comprehend its advantages and disadvantages. Furthermore, different modifications have been proposed to promote the continuation of this study, either to follow the investigations on the Pac-man environment, or to bring NEAT to other useful applications.

B. Archivo de configuración completo

Formato de archivo de configuración por defecto. Los campos marcados por el símbolo \$ son los que han podido variar de modelo a modelo. Los que no, han tenido esos valores para todas las configuraciones propuestas en el estudio.

```
[NEAT]
fitness_criterion      = max $
fitness_threshold     = 185 $
pop_size               = 150
reset_on_extinction   = True $
```

```
[DefaultGenome]
# node activation options
activation_default      = sigmoid
activation_mutate_rate  = 0.0
```

```
activation_options      = sigmoid

# node aggregation options
aggregation_default    = sum
aggregation_mutate_rate = 0.0
aggregation_options    = sum

# node bias options
bias_init_mean          = 0.0
bias_init_stdev          = 1.0
bias_max_value          = 30.0
bias_min_value          = -30.0
bias_mutate_power        = 0.5
bias_mutate_rate         = 0.7
bias_replace_rate        = 0.1

# genome compatibility options
compatibility_disjoint_coefficient = 1.0 $
compatibility_weight_coefficient   = 0.6 $

# connection add/remove rates
conn_add_prob            = 0.5 $
conn_delete_prob          = 0.5 $

# connection enable options
enabled_default           = True
enabled_mutate_rate        = 0.01

feed_forward               = False $
initial_connection          = full_direct $

# node add/remove rates
node_add_prob              = 0.2 $
node_delete_prob            = 0.2 $

# network parameters
num_hidden                 = 0  $
num_inputs                  = 24 $
num_outputs                 = 4  $

# node response options
response_init_mean          = 1.0
```

```
response_init_stdev      = 0.0
response_max_value       = 30.0
response_min_value       = -30.0
response_mutate_power   = 0.0
response_mutate_rate    = 0.0
response_replace_rate   = 0.0

# connection weight options
weight_init_mean         = 0.0
weight_init_stdev         = 1.0
weight_max_value          = 30
weight_min_value          = -30
weight_mutate_power      = 0.5
weight_mutate_rate        = 0.8
weight_replace_rate       = 0.1

[DefaultSpeciesSet]
compatibility_threshold = 3.0 $

[DefaultStagnation]
species_fitness_func = max $
max_stagnation        = 20 $
species_elitism        = 2 $

[DefaultReproduction]
elitism                 = 2 $
survival_threshold      = 0.2 $
```