Universitat Politècnica de Catalunya

Facultat d'Informática de Barcelona

Mención en Computació

Estudio del algoritmo NEAT aplicado al videojuego *Pac-Man*

Memoria del proyecto

Autor:
Darío Blasco

Dirección:
René Alquezar
Enrique Romero

Junio 2019

Contenido

Listado de figuras

Listado de tablas

Parte I

Introducción

En esta parte se pretende dar una idea general sobre este proyecto. Para ello se detalla la contextualización del mismo, se trata el estado actual tanto de las tecnologías como los métodos usados y se resumen otros estudios que guardan cierta relación con éste.

1 Contexto

A la hora de plantear el entrenamiento de *ANNs*[1] para el aprendizaje de juegos, se usa el paradigma conocido como ***reinforcement learning***[2]. Este paradigma, a diferencia del ***entrenamiento supervisado*** y el **no supervisado**, no dispone de datos iniciales para el entrenamiento, estos se generan según el agente codificado realiza interacciones con su entorno, o se simulan como por ejemplo con los métodos *Monte Carlo*[3], siendo el objetivo a largo plazo la optimización (generalmente en forma de minimización) de una función de coste.

Similarmente los algoritmos evolutivos intentan hallar la solución a un problema usando mecanismos derivados de las teorías evolutivas darwinistas. En ellos se parte de un conjunto inicial de soluciones y mediante mutación y recombinación de las mismas se optimiza la función objetivo.

Partiendo de ambos conceptos se desarrolla la idea de **neuroevolución**, codificando cada individuo una *ANN*. Para este proyecto se quiere trabajar sobre este concepto pues suscita gran interés el hecho de lograr aprender un comportamiento, el requerido para saber completar un videojuego en este caso, mediante herramientas de ensayo y error, el algoritmo concreto que se quiere investigar es el *NEAT*.

1.1 Objetivo

El objetivo concreto de este proyecto consiste en la realización de un estudio sobre la aplicación del algoritmo *NEAT* a fin de entrenar una red neuronal para jugar al videojuego *Pac-Man*. Se trata

[1] Red Neuronal Artificial, por sus siglas en ingles Artificial Neural Network
[2] Que se podría traducir cómo entrenamiento por refuerzo
[3] Mas información en: <https://towardsdatascience.com/the-house-always-wins-monte-carlo-simulation-eb82787da2a3>

por tanto de un estudio experimental, en el que se pretende explorar el impacto de diversos factores del algoritmo en el rendimiento de los agentes resultantes.

1.2 Actores implicados

A continuación se detallan los diferentes perfiles implicados en el proyecto, para cada uno se tratan sus responsabilidades.

1.2.1 Desarrollador

El desarrollador del proyecto es el perfil sobre el que recaen las responsabilidades del desarrollo de todo el código necesario para la ejecución del proyecto. Estas incluyen principalmente la interoperabilidad del algoritmo *NEAT* con el juego *Pac-Man* así como cualquier ajuste requerido de la lógica del juego en si.

1.2.2 Analista

Se trata del perfil responsable de diseñar la red, tratar los datos que esta consuma y realizar el análisis posterior de los resultados obtenidos. Adicionalmente es el perfil que decide que experimentos hacer y cómo enfocarlos.

Para este proyecto los perfiles de **desarrollador** y **analista** han estado a cargo del alumno, Darío Blasco.

1.2.3 Director

El director es el perfil encargado de asesorar a los dos perfiles ya definidos a fin de lograr la correcta realización de sus respectivas responsabilidades y, guiar al **analista** a la hora de plantear experimentos y analizar los resultados de los mismos.

Para este proyecto el perfil de **director** ha recaído sobre René Alquezar y Enrique Romero.

2 Estado del arte

En esta sección se detalla el estado actual de las tecnologías y técnicas usadas para el desarrollo del proyecto, y se realiza un breve resumen de estudios similares, tanto en su foco como en los elementos usados, hallados.

2.1 Redes neuronales

Una *ANN* es un sistema de *ML*[4] inspirado en la biología, concretamente en la estructura neuronal del cerebro animal[1]. Estructuralmente, se puede definir una red como un grafo dirigido con pesos en su aristas, cuyos vértices se denominan neuronas y sus aristas conexiones. Dentro de la red las neuronas se ubican en capas diferenciadas

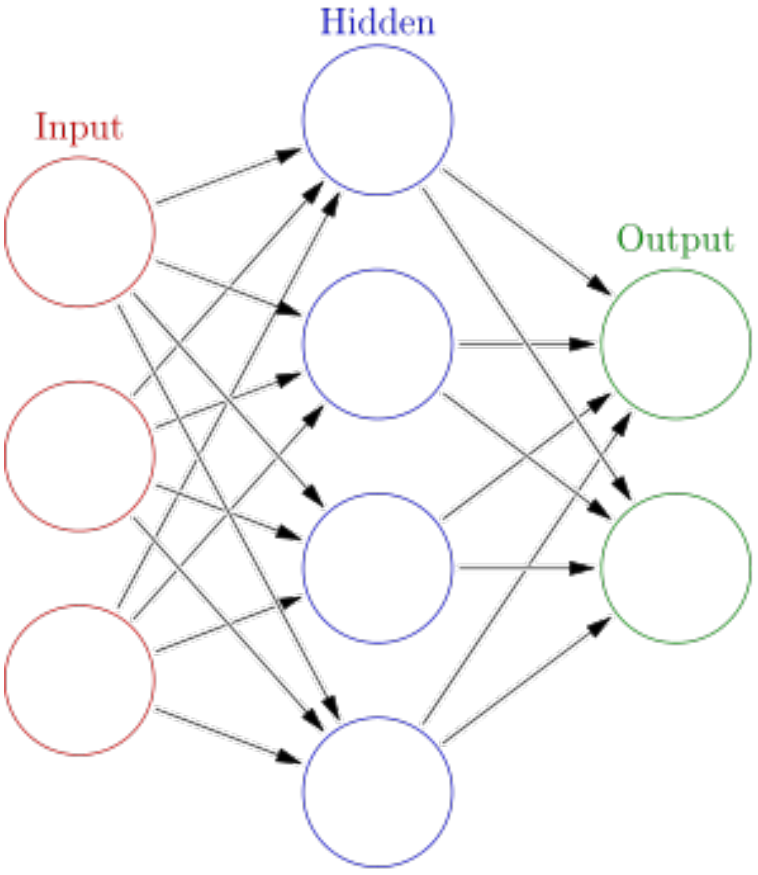


Figura 1: Esquema estructural de una *ANN*

Fuente: By Glosser.ca - Own work, Derivative of File:Artificial neural network.svg, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=24913461>

[4]Aprendizaje Automático, por sus siglas en ingles Machine Learning

La primera capa es la conocida como **capa de entrada**, representada en la imagen superior por las neuronas rojas. Esta capa es la que recibe información sobre el estado del mundo, por ello, todas las neuronas de esta capa presentan únicamente conexiones salientes[5].

La segunda capa es la **capa de salida**, representada en la imagen por las neuronas marcadas en verde. Estas neuronas son las que presentan los resultados de la red al mundo y, por ello, todas sus conexiones son entrantes.

Finalmente, la tercera sección de la red son las denominadas capas ocultas, azul en la imagen. Estas capas conectan la entrada con salida y no hay un número determinado pues depende de cada red, tampoco son imprescindibles, pues pueden existir redes que conecten directamente entrada con salida, por ello, se denomina también a una red con capas ocultas *DNN* [6].

2.1.1 Paradigmas de entrenamiento

A la hora de entrenar *ANN* se distinguen principalmente tres paradigmas distintos según el tipo de problema a tratar.

El primero de estos paradigmas es el **entrenamiento supervisado**, en este caso se dispone de un conjunto de pares entrada-salida, siendo el objetivo que la red derive una función capaz de mapear las entradas a las salidas. El algoritmo usado para este paradigma es el de *back-propagation*[7].Las aplicaciones de este paradigma son el **reconocimiento de patrones**[8] y la **aproximación de funciones**[9].

El segundo paradigma es el **entrenamiento no supervisado**, en el que se dispone de un conjunto de datos de entrada y una función a optimizar (habitualmente minimizar) que relaciona los datos con la salida producida por la red. Este tipo de entrenamiento se usa para toda tarea cuyo objetivo es el de obtener una estimación, por ejemplo el ***clustering***.

[5]Dadas 2 neuronas N_1, N_2 y una conexión $C = N_1 \rightarrow N_2$, se denomina C **saliente** para N_1 y **entrante** para N_2

[6]Red profunda, por sus siglas en inglés Deep Neural Network

[7]También conocido como the backward propagation of errors. Mas información en: <https://brilliant.org/wiki/backpropagation>

[8]También conocido como clasificación

[9]También conocida como regresión

El tercero y último de estos paradigmas es el ***reinforcement learning*** en el que no se disponen de datos de entrada, siendo estos generados según un agente interactúa con el entorno. El objetivo de este paradigma es optimizar (habitualmente minimizar) una función que representa el coste a largo plazo de resolver el problema. Bajo este paradigma se hallan todas las tareas decisionales, por ejemplo juegos o problemas de control.

2.2 Algoritmos evolutivos

Se denominan EA[10] aquellos algoritmos cuyo objetivo es el de simular el proceso de selección natural según las **teorías darwinistas**[2]. Para ello se codifica un conjunto de posibles soluciones al problema a resolver, denominando cada solución individual como un **individuo** y el conjunto de individuos como **población**. Para mejorar los resultados se simula en proceso de selección natural en la población mediante el uso de dos operadores diferentes, la acción de **mutar** toma un individuo y lo modifica (muta) generando un nuevo individuo, por otro lado, la acción de **cruce** toma dos individuos diferentes (**progenitores**) y produce dos individuos nuevos (**sucesores**). La codificación de cada operador y de los individuos es específica de cada problema a resolver.

Adicionalmente se dispone de una función de evaluación f denominada comúnmente ***fitness function*** que permite, dado un individuo, obtener un indicador de la calidad de la solución que este codifica.

El funcionamiento de estos algoritmos procede entonces a aplicar los diferentes operadores hasta que se alcanza alguna condición de finalización, que habitualmente son, pero no limitadas a

- Obtención de un individuo cuya codificación cumple los requisitos mínimos pedidos
- Producción de un numero concreto de generaciones
- Alcance de una cota definida para f

2.3 Neuroevolución

Se denomina **neuroevolución** a la técnica resultante de la mezcla de *EAs* con *ANNs*, para ello cada individuo se encarga de codificar una red completa. Esta técnica presenta beneficios frente al aprendizaje supervisado, ya que la propia red evoluciona a fin de adaptarse al problema a resolver,

[10] Algoritmo Evolutivo, por sus siglas en inglés Evolutionary Algorithm

por lo que resulta una alternativa interesante cuando los datos que se disponen no son suficientes para aplicar un entrenamiento clásico y, por ello, está considerado un método aplicable a problemas menos específicos que las redes tradicionales[3].

Este proyecto trata con uno de estos métodos, el algoritmo *NEAT* que se detalla mas adelante en la parte IV.

2.4 Videojuegos

Se pueden definir los videojuegos cómo un tipo de **experiencias interactivas** en las que el usuario o jugador realiza acciones dentro de un entorno virtual. Se trata de una industria relativamente reciente, sus primeros registros datan de la década de 1950 y se define como tal en la década de 1980, pero presenta un peso creciente en la sociedad actual como parte de la macro industria del entretenimiento, llegando a mover $\sim$ 36000 millones de dólares únicamente en el mercado estadounidense, según cifras de la *ESA*[11][4].

Para este trabajo se ha elegido como problema a aprender el jugar a uno de los videojuegos Mas conocidos, y uno de los primeros en aparecer, *Pac-Man*, detallado mas adelante, en la parte III.

2.5 Relación videojuegos-*AI*

Dada la propia naturaleza interactiva que presentan, los videojuegos han mantenido desde su origen una relación estrecha con la *AI*[12]. Aunque originalmente se trataba de comportamientos precodificados, recientemente se están aplicando técnicas de *AI* mas complejas, por ejemplo **búsqueda local**, a fin de lograr una experiencia mas personalizada de cara al jugador.

Dentro de esta relación, se pueden distinguir dos vertientes claramente diferenciadas

2.5.1 Aprendizaje

Su objetivo es el de entrenar *AIs* a fin de comportarse como agentes del propio juego, con el objetivo de personalizar la experiencia de éste. Como ejemplos se pueden nombrar ***Dragon Age Inquisition***[5] que usa **búsqueda local** a un nivel de profundidad a fin de determinar la mejor acción a

[11] Asociación de Software para el Entretenimiento, por sus siglas en inglés Entertainment Software Association
[12] Inteligencia Artificial, por sus siglas en inglés Artificial Intelligence

realzar por los *NPCs*[13] o ***NERO***[14][6] que usa una extensión de *NEAT* en tiempo real *rtNEAT*[15] para entrenar a las unidades del jugador en una primera fase, usando ese comportamiento aprendido en la segunda fase, consistente en combates *PVP*[16].

A nivel académico o experimental, se puede destacar ***MarI/O***[7], un proyecto que aplica *NEAT* para desarrollar una *AI* capaz de completar el primer nivel de ***Super Mario World*** o diversos proyectos que se detallarán en la sección 2.6.

2.5.2 Contenido

El otro campo de aplicación es en el de generación de contenido, pese a ser un campo relativamente nuevo y considerablemente mas experimental que el anterior, hay proyectos interesantes a destacar, como ***GAR***[17], un *shooter*[18] ubicado en el espacio que se sirve de la extensión *cgNEAT*[19] para generar y modificar todas las armas del propio juego basándose en las estadísticas de uso de las mismas[8]. A nivel *amateur*, a finales del año 2018 un grupo de usuarios de los foros *resetera* empiezan a usar *ML* con el objetivo de remasterizar[20] el apartado visual de juegos clásicos[9][10].

2.6 Estudios similares

A continuación se detallan estudios e informes que mantienen relación con él foco de estudio de éste trabajo. Algunos tratan del uso de *AI* para el juego *Pac-Man* o su variación *Ms. Pac-Man*, llegando tres de ellos a usar la misma implementación del juego usada para este proyecto, se trata también un estudio sobre la aplicabilidad del algoritmo *NEAT* para el juego *Tetris*.

2.6.1 *The Effect of Varying Partial Observability in Ms. Pac-Man*

Se trata de un estudio que investiga si reducir el campo de observación del avatar (*Ms. Pac-Man* en este caso concreto) influye en el comportamiento de los agentes *AI* que juegan[11]. El estudio acaba concluyendo que la visualización parcial o total del mapa supone una ventaja clara en función de las estrategias de los agentes.

[13] Personaje No Jugable, por sus siglas en inglés Non Playable Character
[14] Neuro-Evolving Robotic Operatives
[15] NEAT en tiempo real, por sus siglas en inglés real-time NEAT
[16] Jugador contra Jugador, por sus siglas en inglés Player Vs Player
[17] Galactic Arms Race
[18] Juego cuya mecánica principal consiste en disparar
[19] NEAT generador de contenido, por sus siglas en inglés Content-Generating NEAT
[20] Acto de cambiar la calidad de sonido y/o imagen de un producto audiovisual sin tener que volver a crearlo

"...it is worth re-iterating that the balance of the game changes as visibility does, with ghosts seeing on average over twice as much of the maze as Ms. PacMan in Radius mode. AI experiments have shown that an advantage in the amount of the map that is visible tends to be an important influence on the performance of agents depending on their particular strategies."

(Piers R. Williams, Simon M. Lucas Senior Member, IEEE, and Michael Fairbank
The Effect of Varying Partial Observability in Ms. Pac-Man: 9)

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

Consiste en un informe en el que se detalla el comportamiento del agente ganador del concurso de *AI* para *Ms. Pac-Man* del año 2009[12]. Se trata de un agente complejo con capacidad de realizar secuencias de acciones, lo que como consecuencia otorga al agente la capacidad de atraer fantasmas hacia ciertas posiciones a fin de emboscarlos.

2.6.3 *Reinforcement Learning in Pacman*

Este estudio los resultados de aprendizaje vía *Q-Learning (SARSA), approximate Q-Learning* y *Deep Q-Learning* para desarrollar un agente que juegue automáticamente.[13] Sus conclusiones establecen que *approximate Q-Learning* y *Deep Q-Learning* funcionan correctamente en mapas pequeños, *approximate Q-Learning* también en mapas de tamaño medio, no obstante *Q-Learning* no alcanza el comportamiento esperado.

"From this project we can conclude that the SARSA update for reinforcement learning performs poorly in Pacman game. Approximate Q-learning can improve on SARSA update and the use of intelligent features for Approximate Q-learning works well for small and medium grids. If we don't want to miss any of the important features in the game Deep Q-learning it is a good alternative that can work well, as tested here for small grids. With our efficient representation of the state space the computational cost for DQN was reasonable."

(Abeynaya Gnanasekaran, Jordi Feliu Faba, Jing An
Reinforcement Learning in Pacman: 5)

2.6.4 *Machine Learning Applied to Pac-Man*

En este **TFG** se aplica *AI* a los fantasmas con objetivo de ajustar dinámicamente la dificultad del juego acorde a la capacidad del jugador (*Pac-Man*), para ello se usa *Q-Learning, Double Q-*

Learning, SARSA y *expected SARSA*[14]. Pese a no tener una prueba exhaustiva con usuarios reales, los encuestados decían preferir la implementación resultante de este proyecto respecto del original.

2.6.5 *Automatically Configuring Deep Q-Learning agents for the Berkeley Pacman project*

Se trata de un **TFM** que usa como elemento de experimentación del sistema de configuración automática para *Deep Q-Learning* agentes para *Pac-Man*, logrando obtener una mejora del $\sim 20\%$ en el rendimiento de los agentes auto-configurados.[15]

2.6.6 *Estudi de l'algoritme NEAT aplicat als videojocs*

Otro **TFG** que estudia la aplicación del algoritmo *NEAT* para resolver el videojuego del *Tetris*[16]. Respecto del rendimiento de los modelos resultantes el autor dice obtener resultados por debajo de lo esperado

> *"L'algoritme NEAT ha sigut demostrat en diverses ocasions com una bona solució pel problema de programar un agent que aprengui a jugar a un joc sense coneixement d'aquest. En el cas del nostre projecte això ha sigut veritat a mitges, ja que tot i així que l'algoritme aprenia certes maneres de jugar (e.g distribuir peces per tot el tauler) no ha aconseguit jugar d'una manera estable sense perdre en un interval relativament gran de temps."*
>
> *(Fernandez Villalba, Mario*
> *Estudi de l'algoritme NEAT aplicat als videojocs: 55)*

Parte II

Gestión

En esta parte se detallan los datos referentes a la gestión del proyecto.

Es necesario destacar que el objetivo y planificación originales para el proyecto han resultado ser excesivamente ambiciosos por lo que a mitad del desarrollo ha sido necesario reducir la escala del mismo. Por ello en las secciones siguientes se comentan los detalles del planteamiento original y las modificaciones necesarias a causa de dicha reducción.

3 Alcance

Originalmente se plantea el proyecto como un estudio sobre el efecto de diferentes métodos de entrenamiento, *back-propagation* y neuroevolución, para el aprendizaje de un juego. Por ello, el alcance original definía tres fases en las que se elegían tanto el juego cómo el método neuroevolutivo específico a estudiar, se realizaban los entrenamientos hasta la obtención de los mejores modelos posibles y finalmente se analizaban los modelos resultantes de cada método. Se planteaba también reiterar el proceso modificando el juego a aprender o el método neuroevolutivo de haber tiempo.

Sin embargo, una vez determinados el juego y algoritmo neuroevolutivo, *Pac-Man* y *NEAT* respectivamente, y se empezado el entrenamiento, se detecta que obtener un modelo aceptable vía *NEAT* requiere más tiempo del estimado, por lo que se decide, previo acuerdo con la dirección del proyecto, modificar y concretar el foco del estudio al algoritmo *NEAT* aplicado sobre el juego *Pac-Man* específicamente.

Una vez redefinido el foco del estudio, se procede a, vía experimentación, explorar los efectos de diferentes configuraciones de red y del algoritmo *NEAT*, así como de distintos tratamientos en la entrada de la red o modificaciones en la propia función de evaluación de un individuo afectan al rendimiento y comportamiento de los modelos resultantes.

3.1 Obstáculos

Bajo el planteamiento original se definen cuatro obstáculos con posibilidad de afectar el correcto desarrollo del proyecto.

El primero, y más grave, de estos obstáculos hacía referencia al calendario, pues ya se preveía un calendario ajustado tratando además con métodos de entrenamiento para los que resulta complicado hacer una estimación precisa de su coste temporal. Como resulta evidente dado el cambio de foco del estudio, este obstáculo finalmente ha evitado poder realizar una parte considerable del proyecto original.

El segundo obstáculo consiste en la falta de datos para poder aplicar *back-propagation*, en ese caso se considera implementar un agente que genere los datos, a fin de disponer, como mínimo de una aproximación. Todo y no usar finalmente *back-propagation*, ha sido necesario igualmente implementar un agente de evaluación heurística, para disponer de una referencia a la hora de analizar el resultado de la neuroevolución.

El tercer obstáculo trata la posibilidad de requerir una implementación propia tanto del juego como del algoritmo en caso de no encontrar ninguna que satisfaga. En este caso, se ha encontrado una implementación de *NEAT* altamente ajustable, por lo que no se ha requerido implementación propia. Respecto al *Pac-Man*, pese a hallar una implementación, ha sido necesario abstraerla para su correcto funcionamiento tanto con *ANNs* como dentro del proceso de *NEAT*.

El último obstáculo planteado originalmente, y que al cambiar el foco de estudio ya no afecta, trataba la potencial baja fiabilidad de los modelos resultantes de cada método de entrenamiento.

4 Metodología

En esta sección se detalla la metodología a seguir durante el desarrollo del proyecto, se define la organización y seguimiento del mismo, el sistema de validación de resultados y las herramientas a usar.

4.1 Organización

Se plantea una organización del proyecto basada en métodos *Agile*. Se trata de una metodología de desarrollo incremental, organizado en bloques cortos a fin de entregar una funcionalidad completa al término de cada ciclo.

Originalmente se plantean ciclos bisemanales más definitivamente se establecen ciclos semanales, esto ha permitido mantener una reunión fija semanal con la dirección del proyecto en la que se revisan y comentan los resultados obtenidos de los experimentos realizados durante esa semana y se plantean los nuevos experimentos a realizar.

Usar este tipo de método ha permitido una temprana detección de la excesiva ambición en el alcance del proyecto, y ha facilitado reducir el mismo.

4.2 Validación

Respecto a la validación de los resultados de los experimentos, se hace caso por caso, una vez obtenidos modelos aceptables, se validan éstos haciendo jugar al modelo un numero fijado de partidas para obtener datos de su puntuación media en el mapa y el porcentaje de juegos que logra ganar.

5 Planificación

A continuación se detallan las tareas recursos y plan de acción del proyecto, en su planteamiento original y aclarando las modificaciones requeridas.

5.1 Tareas

Como se ha comentado anteriormente, la planificación original estructura el proyecto en tres fases. El objetivo de la primera fase sería evaluar las soluciones tecnológicas a usar, el de la segunda consistía en realizar los entrenamientos de los modelos y finalmente, durante la tercera se analizarían los resultados. Por ello la lista de tareas original consistía en

- Evaluación de soluciones software para el desarrollo
- Evaluación de soluciones hardware para el entrenamiento
- Obtención, análisis y tratamiento de datos para el entrenamiento y validación
- Entrenamiento de la red vía *back-propagation*
- Entrenamiento de la red vía neuroevolución
- Validación de ambas redes
- Análisis de complejidad
- Análisis de costes
- Análisis de comportamiento

Debido al cambio de foco del proyecto el listado de tareas realizadas queda modificado, pasando a ser el siguiente

- Evaluación de soluciones software para el desarrollo
- Familiarización con los entornos elegidos
- Abstracción sobre la implementación elegida para *Pac-Man*
- Experimentación

5.2 Recursos

La planificación original plantea usar una solución *cloud computing* para el entrenamiento, finalmente el entrenamiento se ha hecho en local, no optando por esa opción. El resto de recursos se han mantenido según el detalle que sigue.

5.2.1 Hardware

Únicamente se ha usado el portátil del desarrollador, siendo este un **MacBook Pro Retina** de 13 pulgadas, modelo *early 2015*.

5.2.2 Software

Respecto del lenguaje de programación usado para el desarrollo, se ha decidido según las implementaciones del algoritmo y juego halladas, se ha optado definitivamente por **Python**[21].

El código desarrollado para el proyecto se encuentra almacenado en un repositorio privado ubicado en **Bitbucket**[22]. Se ha usado para su desarrollo la herramienta **Visual Studio Code**[23].

La documentación del proyecto se ha hecho usando la herramienta **Overleaf**[24], se trata de una solución *online* para el procesado de texto vía LaTeX[25]. Finalmente, el sistema operativo sobre el que se ha trabajado es el **MacOS Mojave** en su versión 10.14.4.

5.3 Estimación de tiempos

A continuación se detalla la estimación de tiempos tanto para la planificación original como la modificada.

[21] <https://www.python.org/>
[22] <https://bitbucket.org/>
[23] <https://code.visualstudio.com/>
[24] <https://www.overleaf.com>
[25] <https://www.latex-project.org/>

5.3.1 Original

Tarea	Tiempo (horas)	Inicio	Fin
Realización hito inicial	90	2019.02.18	2019.03.29
Evaluación de soluciones software	20	2019.04.01	2019.04.03
Evaluación de soluciones hardware	20	2019.04.03	2019.04.05
Obtención, análisis y tratamiento de datos	30	2019.04.08	2019.04.12
Entrenamiento de la red vía *back-propagation*	110	2019.04.12	2019.05.01
Entrenamiento de la red vía neuroevolución	110	2019.05.02	2019.05.21
Validación de ambas redes	10	2019.05.21	2019.05.22
Análisis de complejidad	20	2019.05.23	2019.05.27
Análisis de costes	20	2019.05.27	2019.05.29
Análisis de comportamiento	20	2019.05.30	2019.06.03
Documentación	60	2019.06.10	2019.06.18
Total	**510**	**2019.02.18**	**2019.06.18**

Tabla 1: Estimación de tiempos por tarea acorde a la planificación original

Fuente: Elaboración propia

Con la estimación planteada en la tabla 1, los diagramas de ***GANTT*** y ***PERT*** correspondientes serían

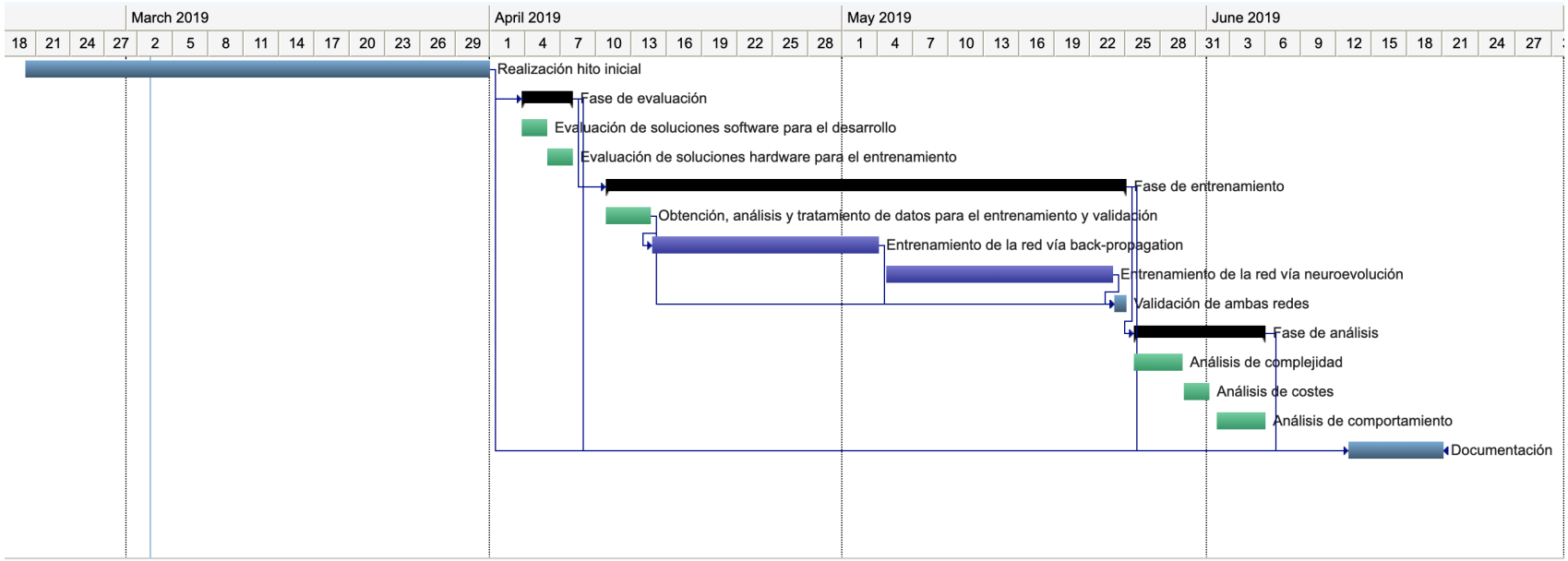
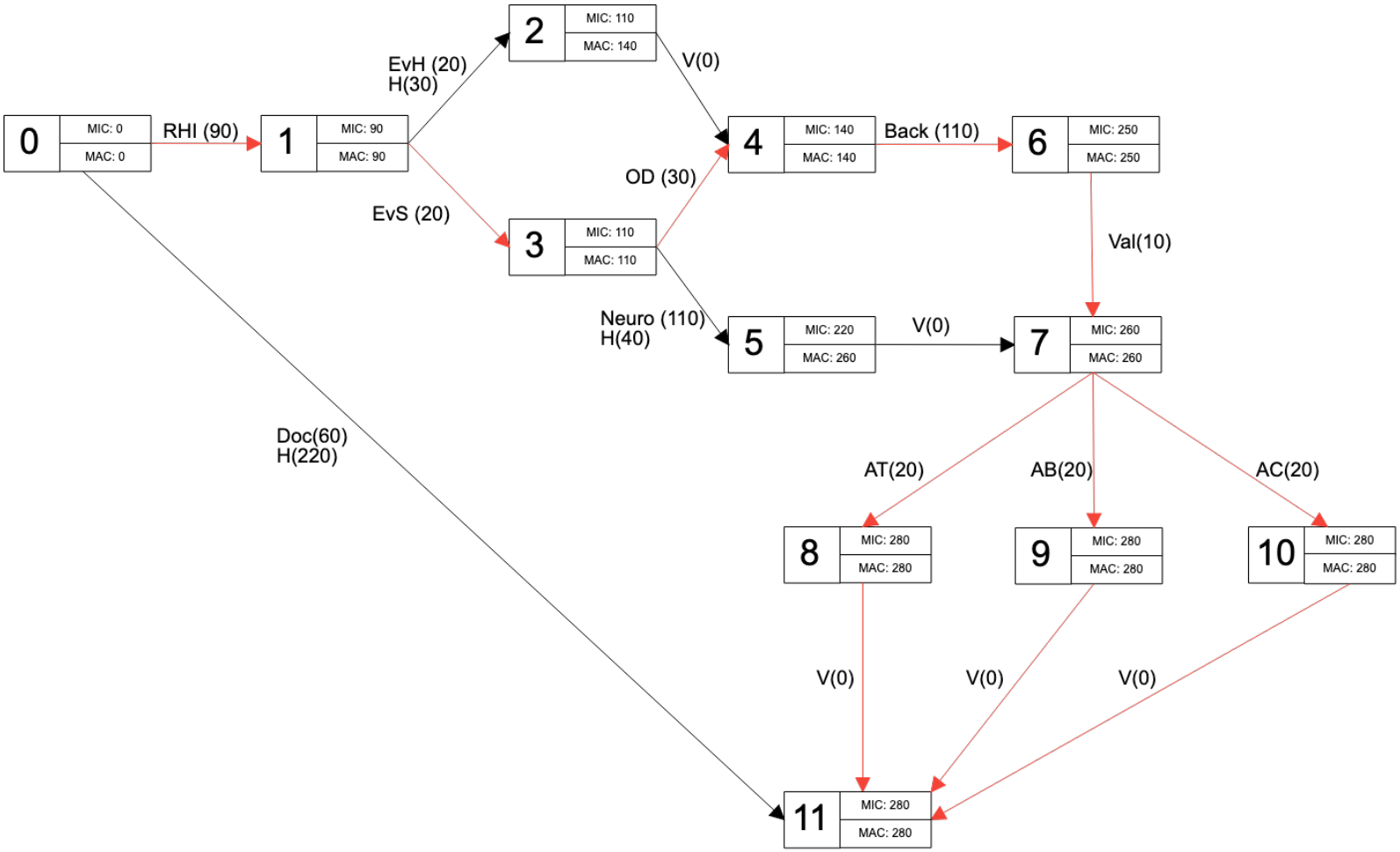


Figura 2: Diagrama de ***GANTT*** correspondiente a la planificación de presentada en la tabla 1

Fuente: Elaboración propia



RHI	EvH	EvS	OD	Neuro	Back	Val	AT	AB	AC	Doc	H
Realización Hito Inicial	Evaluación Hardware	Evaluación Software	Obtención Datos	Entrenamiento via Neuroevolución	Entrenamiento via Back-propagation	Validación de modelos	Análisis complejidad	Análisis Comportamiento	Análisis Coste	Documentación	Holgura del camino

Figura 3: Diagrama de ***PERT*** correspondiente a la planificación de presentada en la tabla 1

Fuente: Elaboración propia

5.3.2 Final

Tarea	Tiempo (horas)	Inicio	Fin
Realización hito inicial	90	2019.02.18	2019.03.29
Evaluación de soluciones software para el desarrollo	20	2019.04.01	2019.04.03
Familiarización con los entornos elegidos	40	2019.04.03	2019.04.05
Abstracción sobre *Pac-Man*	80	2019.04.08	2019.04.19
Experimentación	300	2019.04.22	2019.06.12
Documentación	60	2019.06.11	2019.06.20
Total	**590**	**2019.02.18**	**2019.06.20**

Tabla 2: Estimación de tiempos por tarea acorde al final

Fuente: Elaboración propia

Siendo el diagrama de ***GANTT*** correspondiente a la planificación de la tabla 2 el siguiente, en este caso no se plantea ***PERT*** pues todas las tareas a realizar son completamente secuenciales.

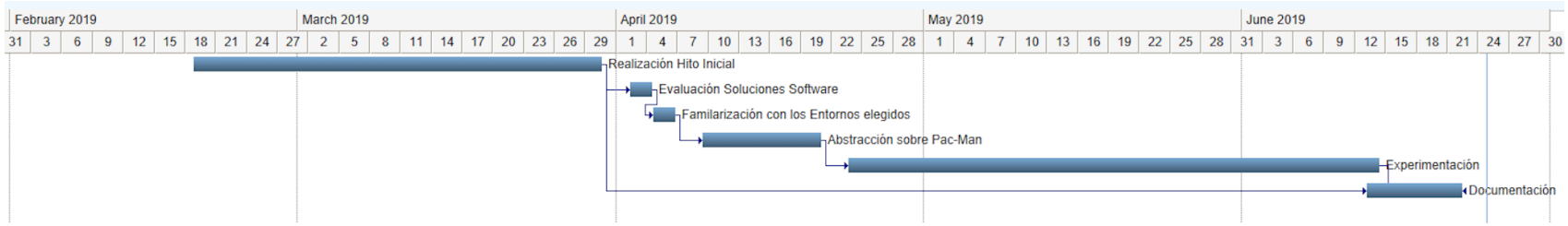


Figura 4: Diagrama de ***GANTT*** correspondiente a la planificación de presentada en la tabla 2

Fuente: Elaboración propia

6 Presupuesto

El desglose de costes para el proyecto es el siguiente

Rol	**Tiempo (horas)**	**Precio (€/hora)**	**Total (€)**
Manager	150	50	7500.00
Analista	250	45	11250.00
Desarrollador	190	35	6650.00
Total	**590**		**25400.00**

Tabla 3: Desglose de los costes de recursos humanos

Fuente: Elaboración propia

Concepto	**Precio (€)**	**Amortización (€/hora)**	**Uso (horas)**	**Total (€)**
MacBook Pro (incluye MacOS)	1505.59	0.19	590	112.10
Overleaf (licencia anual)	168.00	0.08	150	12.00
Total	**1673.59**			**124.10**

Tabla 4: Desglose de los costes de software y hardware

Fuente: Elaboración propia El cálculo para las amortizaciones se hace asumiendo 250 días hábiles por año.

Concepto	**Precio (€)**	**Unidades**	**Total (€)**
Electricidad	0.11	59000	6490.00
Libreta *Moleskine*	15.20	1	15.20
Pizarra	50.76	1	50.76
Herramientas de escritura	12.13	1	12.13
Total			**6568.09**

Tabla 5: Desglose de los costes para gastos adicionales

Fuente: Elaboración propia

Con los datos de las tablas 3, 4 y 5 el presupuesto total para el proyecto es

Concepto	**Coste**
Recursos Humanos	25400.00
Hardware, Software y licencias	124.10
Gastos adicionales	6568.09
Contingencias (20%)	6418.44
Total	**38510.63**

Tabla 6: Presupuesto definitivo para el desarrollo del proyecto

Fuente: Elaboración propia

7 Sostenibilidad

7.1 Económica

El coste planteado en el presupuesto no difiere en exceso de otros realizados, que presentaban una duración y requisitos técnicos similares.

El coste referente a recursos humanos se ha calculado tomando como referencia de precio por hora el que se podría hallar en cualquier empresa de consultoría, a fecha de realización de este documento. Similarmente para el gasto en electricidad, se ha usado para ese cálculo una aproximación de un kW consumido por cada hora de desarrollo.

Respecto a los presupuestos similares hallados, no parece que el planteado en este documento mejore el coste planteado en ellos.

7.2 Ambiental

Con respecto a la dimensión ambiental, el proyecto no causa un impacto directo, afectando únicamente a esta vertiente el consumo de energía requerida para el entrenamiento.

Dado que no se ha hallado información respecto al impacto de estudios similares no es posible medir cómo mejora ambientalmente lo planteado en este documento respecto de lo ya existente.

7.3 Social

El estudio pretende incrementar la literatura ya existente sobre las aplicaciones del algoritmo *NEAT*. Se ha optado por el campo de la neuroevolución pues es un campo que suscitaba interés para el autor. Similarmente el campo de los videojuegos se ha elegido pues la continuación de estudios que se quiere realizar incluye un máster relacionado con la industria y se espera que el proyecto pueda servir a su vez para ampliar el porfolio profesional, adicionalmente, dentro de al industria del entretenimiento, se trata de uno de los sectores más económicamente activos y con mayor crecimiento.

Cómo se ha introducido en la sección 2, la relación entre videojuegos e inteligencia artificial existe desde la concepción de los mismos, con objetivo de ofrecer experiencias cada vez más adaptadas al jugador, empezando algunas empresas ya a usar el entrenamiento de modelos predictivos para

preparar sus *AIs*.

No se ha encontrado mención a la necesidad de proyectos similares a este, no obstante ya empiezan a existir algunos videojuegos, si bien es cierto que principalmente en el campo experimental, que aplican métodos mencionados en éste documento, por lo que no es difícil asumir que en un futuro no muy lejano se empiecen a aplicar estos métodos en productos con un foco más comercial.

7.4 Matriz de sostenibilidad

	Dimensión económica	**Dimensión ambiental**	**Dimensión Social**
Hito Inicial	7	6	9
PPP	7	7	8

Tabla 7: Puntuación de sostenibilidad para el hito inicial y el proyecto puesto en producción

Fuente: Elaboración propia

Parte III

Pac-Man

Ésta parte trata todos los elementos del proyecto relacionados con el juego *Pac-Man*.

Para ello en las siguientes secciones se describe el juego original y su funcionamiento, que elementos lo componen y su objetivo. A continuación, se procede a detallar las diferencias que presenta la implementación de *Stanford*, elegida como base para abstraer una implementación propia, con lo descrito. Finalmente se describe la implementación propia realizada y cómo ésta interactúa con *NEAT*.

8 Juego original

Pac-Man se trata de un juego *arcade*[26] publicado por *Namco*[27] para máquinas recreativas en el año 1980. Se trata de un juego con mecánicas simples, fáciles de comprender para el jugador, por ello se ha elegido como el problema sobre el que realizar el aprendizaje.

En el año 1982 aparece una nueva versión del juego, *Ms.Pac-Man* que añade cambios estéticos e introduce mayor aleatoriedad para los movimientos de los fantasmas a fin de evitar patrones, adicionalmente la fruta se mueve por el mapa en lugar de mantenerse estática.

Pese a que hay más literatura tratando *Ms.Pac-Man* cómo problema a resolver vía *AI*, se ha preferido optar por el *Pac-Man* original, pues se considera un problema más simple a resolver.

[26] Género de juegos cuyo objetivo es obtener la mayor puntuación posible
[27] Fusionada con *Bandai* el año 2006, en 2015 adopta el nombre *Bandai Namco Entertainment*

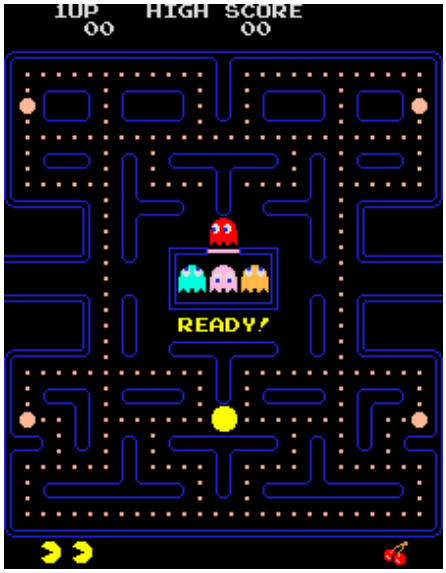


Figura 5: Primer nivel en *Pac-Man*

Fuente: Wikipedia,
<https://en.wikipedia.org/w/index.php?curid=533608>

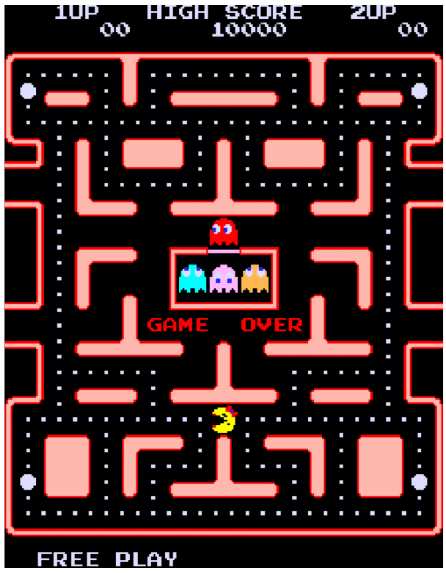


Figura 6: Primer nivel en *Ms.Pac-Man*

Fuente: Wikipedia,
<https://en.wikipedia.org/w/index.php?curid=13748802>

8.1 Elementos del juego

A continuación se detallan los diferentes elementos que intervienen en el juego, cómo estos afectan e interactúan con el resto y cómo modifican la puntuación del jugador.

8.1.1 *Pac-Man*

Se trata del avatar del jugador, el jugador puede por tanto interactuar con el juego realizando cualquiera de las 4 acciones que se le permiten:

- Desplazarse arriba
- Desplazarse abajo
- Desplazarse a la izquierda
- Desplazarse a la derecha

Una vez *Pac-Man* inicia el desplazamiento en cualquier dirección, continuará avanzando automáticamente en esa dirección hasta que encuentre una pared, por lo que no hay acción independiente para terminar el desplazamiento.

En caso de colisionar con otro elemento del juego, se evalúa el resultado de dicha colisión automáticamente, por lo que no se dispone de acción independiente para consumir la comida o cápsulas, se consumen al colisionar.

8.1.2 Fantasmas

El juego dispone de cuatro fantasmas que presentan cada uno un comportamiento específico y claramente distinguible por los colores del mismo. Los diferentes comportamientos son los que siguen[28]:

- El rojo, también conocido como *Blinky*, intenta perseguir a *Pac-Man*.
- El rosa, también conocido como *Pinky*, intenta emboscar a *Pac-Man*, ubicándose delante de él.
- El azul cian, también conocido como *Inkt*, alterna aleatoriamente entre perseguir a *Pac-Man* y alejarse de él.
- El naranja, también conocido como *Clyde*, elige su movimiento al azar.

Cada fantasma, a su vez, se puede hallar en uno de dos estados diferentes, **asustado** o **normal**, distinguibles su color, pasan a tener color azul, y expresión. En caso de estar en estado **asustado** el fantasma cambia su comportamiento y pasan todos ellos a alejarse de *Pac-Man*.
En caso de colisionar con un fantasma **asustado**, el jugador recibe una suma importante en su puntuación (*e.g. 200*). No obstante, si el estado del fantasma es **normal** el jugador perderá una de sus vidas.

A diferencia de *Pac-Man* los fantasmas no pueden realizar giros de 180 grados.

8.1.3 Comida

Se trata de los puntos pequeños, ocupan gran parte del mapa y añaden una suma pequeña (*e.g. 10*) a la puntuación del jugador si este los consume.

[28]Según comenta **Toru Iwatani** , diseñador del juego, en su conferencia del 2011 en la *GDC*

8.1.4 Cápsulas

Se trata de puntos de tamaño mayor que los de la comida, añaden una suma ligeramente superior que la de los puntos (*e.g. 50*) al ser consumidos y, adicionalmente, causan un cambio en el estado de los fantasmas a asustado..

8.1.5 Fruta

Adicionalmente hay un numero de frutas esparcidas por el mapa, que suman más puntos al jugador, la cuantía depende de la fruta, y el tipo de frutas que aparecen viene determinado por el mapa.

Figura 7: Elementos del juego descritos, con puntuaciones

Fuente: Wikipedia, <https://en.wikipedia.org/w/index.php?curid=13773907>

8.2 Fin de la partida

La partida se considera finalizada cuando el estado resultante de una colisión de *Pac-Man* con otro elemento del juego resulta en un estado final, estos estados pueden ser estados ganadores o perdedores, definido cada uno como sigue

8.2.1 Estado ganador

Se considera estado ganador aquel en el que se ha consumido toda la comida del mapa, independientemente del estado de los fantasmas, fruta y cápsulas presentes. Al alcanzar este estado, se finaliza el juego.

8.2.2 Estado de pérdida

Se considera estado de pérdida todo aquel que resulte de la colisión del jugador con un fantasma en estado **normal** cuando no dispone de vidas adicionales. Al alcanzar este estado se finaliza el juego.

Si el jugador dispone de una o más vidas, se considera una pérdida parcial y se le sustrae al jugador una de sus vidas, se reinicia el estado y posición de los fantasmas y *Pac-Man*, pero no el de la comida, cápsulas ni fruta.

9 Implementación de *Stanford*

Para el desarrollo de este trabajo se ha optado por usar una implementación del juego, desarrollada sobre *Python 2*, usada en la universidad de *Stanford* para el desarrollo de su curso de *AI*.

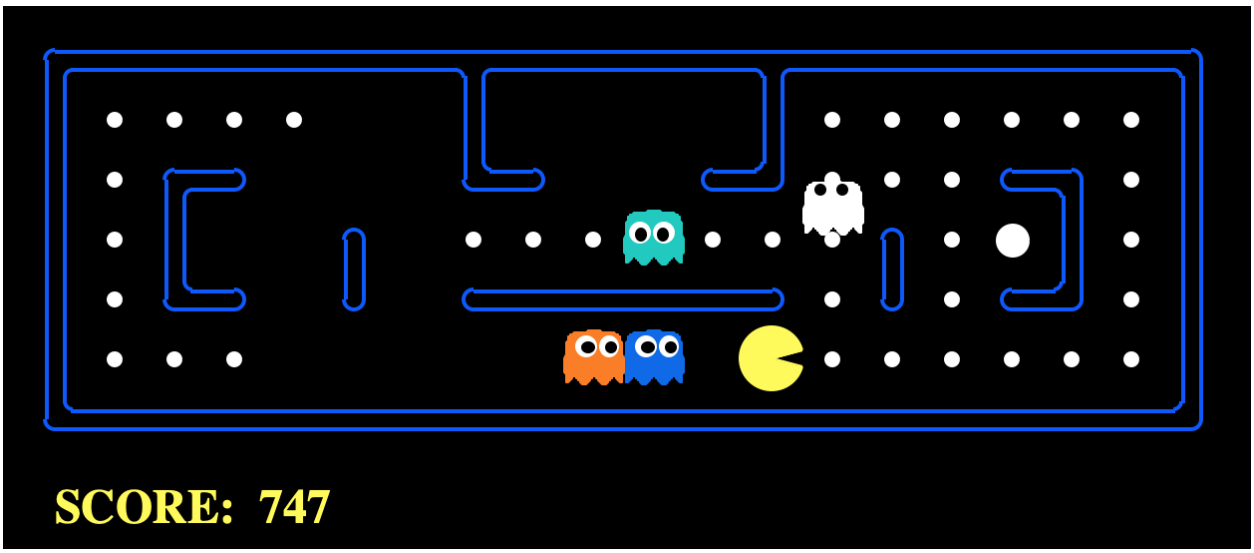


Figura 8: Ejemplo de la implementación de *Stanford*

Fuente: Elaboración propia

9.1 Diferencias notables

Existe un numero de diferencias importante en esta implementación respecto al juego original descrito con anterioridad, más allá de las puramente cosméticas. A continuación se detallan las más importantes de cara al desarrollo del proyecto.

En relación con *Pac-Man*, se pierde el movimiento automático, por lo que en cada ciclo de juego se debe proveer a *Pac-Man* con una dirección en la que moverse. Cómo consecuencia de este cambio, se añade a *Pac-Man* una nueva acción, no moverse.*Pac-Man* dispone únicamente de una vida.

En relación con la puntuación, las cápsulas únicamente modifican el estado de los fantasmas, y ya no otorgan puntos. Se puede definir una perdida de puntos por cada ciclo de juego (*e.g. 1*), a fin de penalizar comportamientos, con el objetivo de ayudar al aprendizaje de la *AI*. Se añade una suma de puntos (*e.g. 500*) al ganar y se resta una cantidad fija al perder (*e.g. 500*) a fin de facilitar el entrenamiento. Esta implementación no dispone del elemento fruta.

En relación con los fantasmas, se pierde el código de colores para su comportamiento, viniendo el color definido por el orden interno de los fantasmas. Adicionalmente, al consumir un fantasma, éste reaparece al instante en su posición de origen en lugar de trasladarse hasta ella cómo en el original.

9.2 Estructura

Esta sección pretende dar al lector una idea aproximada sobre cómo está estructurada esta implementación, por lo que se detallarán las clases más importantes, cuál es su objetivo y cómo se relacionan entre ellas.

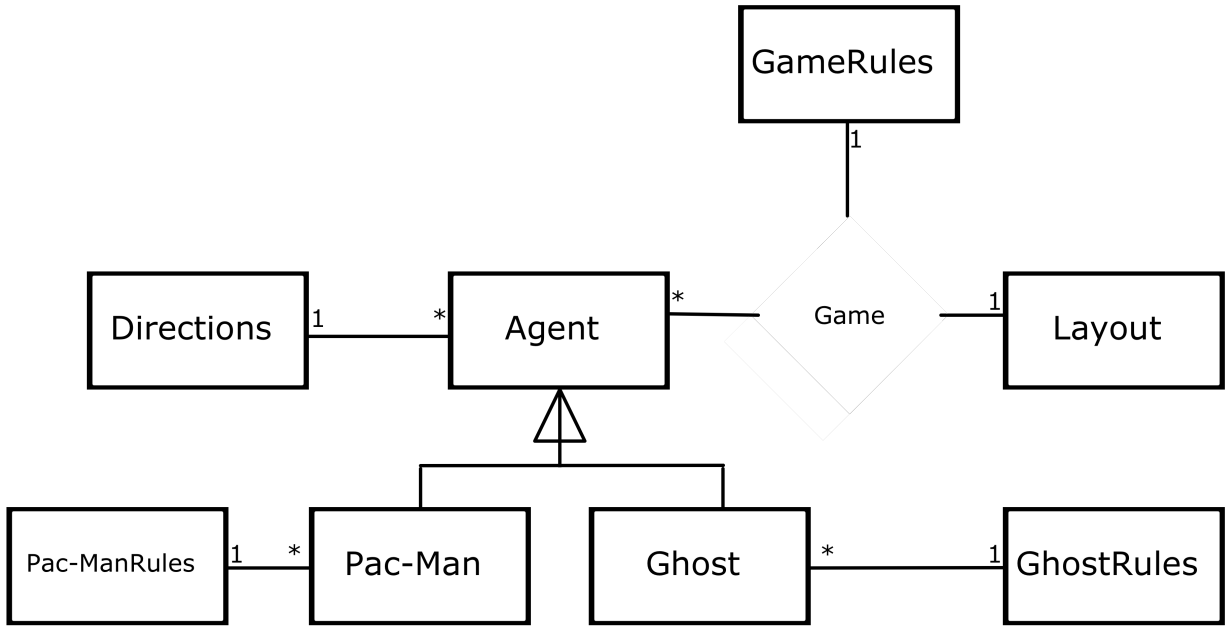


Figura 9: Esquema relacional de las clases relevantes en la implementación de *Stanford*

Fuente: Elaboración propia

9.2.1 Clase *Layout*

Se trata de la clase encargada de gestionar la información de los diferentes mapas del juego, estos mapas están almacenados en disco como un fichero de texto que esta clase se encarga de leer y generar el mapa que interactúa con el resto de clases. Cabe destacar que todo y almacenar en memoria el mapa en forma de matriz, se trata como si fuese una pantalla gráfica, con el $(0, 0)$ ubicado en la esquina inferior derecha y acceso en formato (x, y).

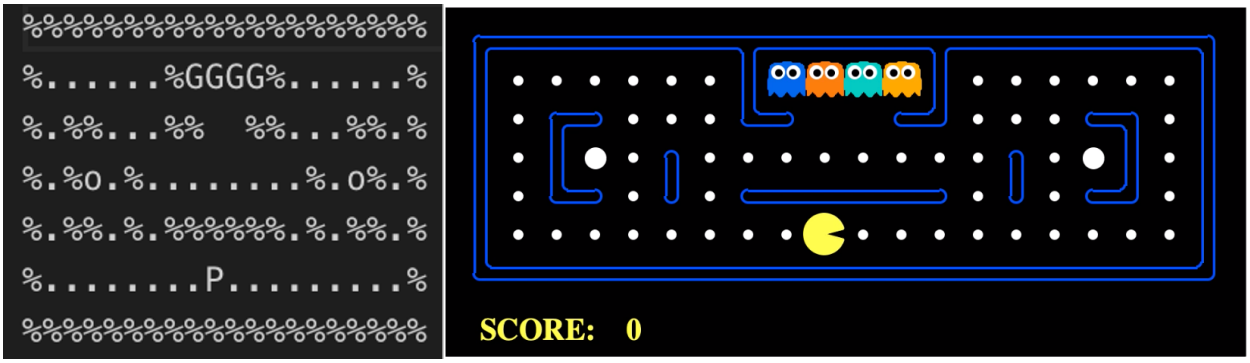


Figura 10: Comparación de un mapa en lectura y resultado final

Fuente: Elaboración propia

Como se observa en la imagen anterior, las paredes se representan con el símbolo %, la comida con ., las cápsulas con *o*, los fantasmas con *G* y *Pac-Man* con *P*.

9.2.2 Clase *Directions*

Se trata de un diccionario que almacena la nomenclatura de direcciones para el juego.

Las direcciones son norte, sur, este, oeste y *stop*, y se puede acceder a cada una de ellas vía notación de punto (*e.g. Directions.North*). Adicionalmente, contiene 3 diccionarios adicionales para codificar los cambios de dirección a derecha, izquierda y media vuelta.

9.2.3 Clase *GameState*

Es la clase encargada de mantener el estado del juego, dispone de un conjunto de accesores a fin de permitir a los agentes consultar el estado del juego a la hora de evaluar su movimiento.

9.2.4 Clase *Agent*

Se trata de la interfaz de clase para todos los agentes. Requiere de la implementación de un método que dado un estado, instancia de la clase *GameState*, retorne una acción válida de las codificadas en la clase *Directions*.

9.2.5 Clases *PacmanRules, GhostRules* y *ClassicGameRules*

Son las clases encargadas de codificar las normas de diferentes agentes y del juego. En ellas se codifican cosas cómo por ejemplo el comportamiento de cada agente al colisionar con otro, o el

proceso de inicialización o fin de un juego.

9.2.6 Clase *Game*

La clase padre que contiene la relación del resto de clases anteriormente detalladas

10 Implementación propia sobre Stanford

En esta sección se detalla el trabajo realizado sobre la implementación de Stanford, ya detallada anteriormente, a fin de facilitar la interoperabilidad con el algoritmo *NEAT* y usando *ANNs* como núcleos decisionales para los agentes.

10.1 Mapas

Principalmente concerniente a la clase *Layout*, a fin de facilitar el cálculo de distancias, necesario pues la mayoría de entradas de la red tratan con ellas, se añade crea una nueva clase *Distances* y se añade como atributo. Esta clase consta de una matriz de tamaño $h \times w$ siendo h la altura y w la anchura del mapa en cuestión.

Se define entonces el contenido de $distances[i][j] \forall i \in [0...h] \forall j \in [0...w]$ como sigue:

Distancia desde la posición (i, j) hasta el obstáculo (*i.e.* pared) más cerco en cada una de las 4 direcciones, junto a una matriz de tamaño $h \times w$ con cada posición (i', j') contiene la distancia (d) entre las posiciones $(i, j), (i', j')$, definiendo $d((i, j), (i', j'))$ cómo el camino mínimo entre $(i, j), (i', j')$.

Para calcular d se usa una modificación propia del algoritmo *Floyd-Warshall*[29] (se puede consultar el código en el anexo A) para hallar caminos mínimos en un grafo con pesos, considerando el grafo

$$G = (V, E)$$
$$V = (i, j) \forall i \in [0...h] \forall j \in [0...w]$$
$$E = ((i, j), (i', j')) \forall i, i' \in [0...h] \forall j, j' \in [0...w] \leftrightarrow (|i - i'| = 1 \wedge j = j') \vee (|j - j'| = 1 \wedge i = i')$$
$$\forall e \in E : weight(e) = 1$$

Floyd-Warshall permite obtener todos los caminos mínimos de G con complejidad $\Theta(|V|^3)$ en tiempo y $\Theta(|V|^2)$ en espacio. Por ello, al definir los vértices de G como pares de coordenadas (i, j) se pueden

[29] Más información en: <https://en.wikipedia.org/wiki/Floyd-Warshall_algorithm>

desarrollar los costes como sigue

$$|V| = h \times w$$
$$|V|^2 = (h \times w)^2 = h^2 \times w^2$$
$$|V|^3 = (h \times w)^3 = h^3 \times w^3$$

Se añaden adicionalmente atributos con valores precalculados cómo la comida total del mapa y la puntuación máxima obtenible en el mismo.

Con todas las modificaciones aquí detalladas, los tiempos medios de carga de los mapas usados son

Nombre del mapa	**Tiempo (s)**
smallClassic	1.13
mediumClassic	4.20
originalClassic	184.51

Tabla 8: Tiempos medios de carga por mapa

Fuente: Elaboración propia

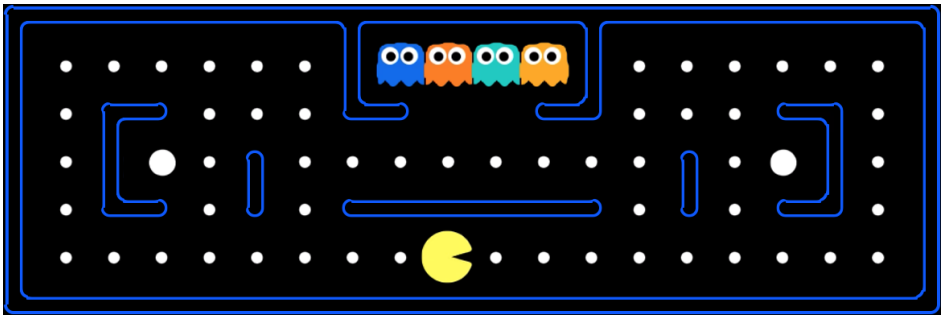


Figura 11: Mapa identificado como *smallClassic*

Fuente: Elaboración propia

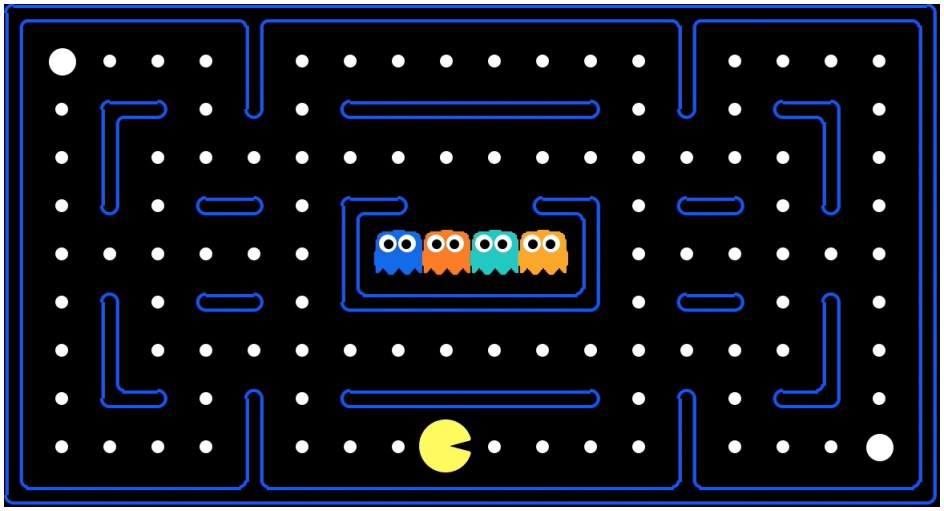


Figura 12: Mapa identificado como *mediumClassic*

Fuente: Elaboración propia

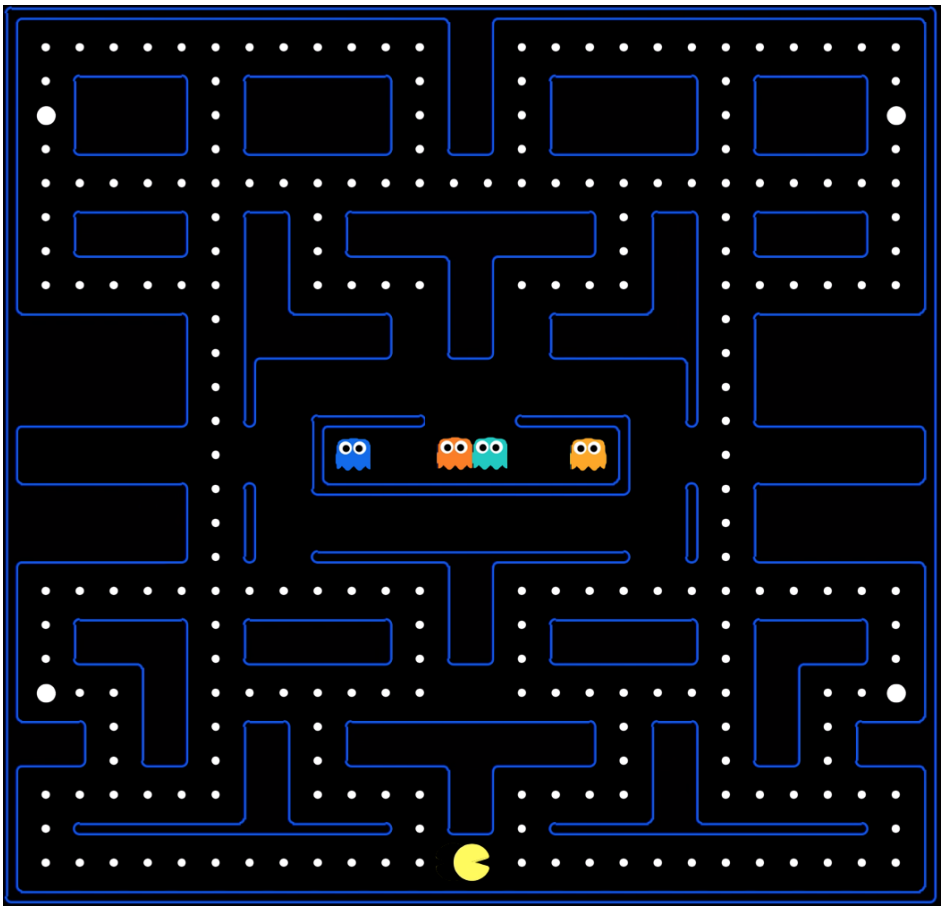


Figura 13: Mapa identificado como *originalClassic*

Fuente: Elaboración propia

10.2 Agentes

Respecto a los diferentes agentes del juego, se han desarrollado seis agentes de comportamiento, uno por fantasma, aproximando los comportamientos ya definidos en la sección8.1.2, un agente heurístico para *Pac-Man*, a fin de tener un marco de referencia para su comportamiento y, finalmente el agente de *Pac-Man* que usa una *ANN* como núcleo decisional. Dado el poco acoplamiento del código, generar un nuevo agente consiste simplemente en crear una subclase de *Agent* e implementar el método *getActions* del mismo.

10.2.1 Fantasmas

El agente correspondiente a *Blinky*, el que persigue a *Pac-Man*, determina su siguiente posición eligiendo de las posibles direcciones de movimiento aquella que minimice la distancia entre la nueva posición y la posición adyacente a la espalda de *Pac-Man*.

El agente correspondiente a *Pinky*, el que intenta interceptar a *Pac-Man*, determina su siguiente posición eligiendo de las posibles direcciones de movimiento aquella que minimice la distancia entre la nueva posición y la posición adyacente a la boca de *Pac-Man*.

El agente correspondiente a *Inky*, determina su siguiente posición eligiendo de las posibles direcciones de movimiento aquella que minimice la distancia entre la nueva posición y la posición de *Pac-Man* o aquella que maximice la misma, con un factor de aleatoriedad del 50%, modificable en caso de así desearlo.

Finalmente, el agente correspondiente a *Clyde*, determina su siguiente posición eligiendo una de las posibles direcciones de movimiento de manera aleatoria.

Todos los comportamientos definidos son para el estado en que los fantasmas no se encuentren asustados, en caso de estarlo, todos se comportan de manera idéntica, eligiendo la dirección que maximice la distancia entre su nueva posición y la de *Pac-Man*.

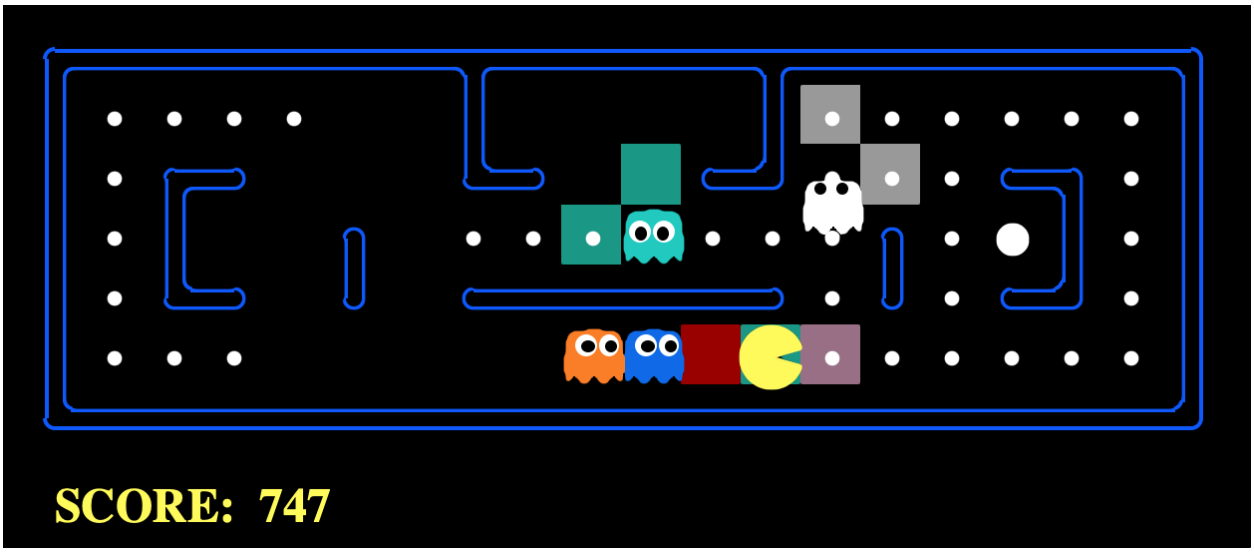


Figura 14: Representación gráfica de la posición a la que tratará de moverse cada fantasma, usando el color original como indicador

Fuente: Elaboración propia

10.2.2 *Pac-Man* heurístico

El agente heurístico se basa en una función h que da una aproximación sobre la puntuación alcanzable en caso de desplazarse en cada posible dirección, eligiendo como dirección de desplazamiento la que evalúe con un valor mayor de h. Tras realizar una búsqueda para hallar una implementación y no hallar resultados que se ajustasen a las necesidades del proyecto, se opta por desarrollar funciones h propias.

Se define por tanto h_1 de la siguiente forma:

Sea G el conjunto de fantasmas del juego, $G_s \in G$ el conjunto de fantasmas asustados.
Sean C, F el conjunto de cápsulas y el conjunto de comida que quedan en el mapa respectivamente.
Sea $d(p_1, p_2)$ la distancia medida en camino mínimo entre las posiciones p_1 y p_2 y $p(\,)$ la posición del elemento del juego.
Sean *dead, ghost, food, win* las bonificaciones obtenidas por perder, comer un fantasma, comer un punto y ganar respectivamente, todas en valores positivos.

$$h_1(pos) = -\sum_{\forall g \in G - G_s} \frac{dead}{d(pos, p(g))} + \sum_{\forall g \in G_s} \frac{ghost}{d(pos, p(g))} + \sum_{\forall c \in C} \frac{ghost \times |G - G_s|}{d(pos, p(c))} + \sum_{\forall f \in F} \frac{food + (\frac{win}{|F|})}{d(pos, p(f))}$$

Se implementa también otro agente que intenta normalizar las distancias usando una forma $1 - d$ en lugar de la $1/d$ presentada, por lo que la función h_2 se define como

$$h_2(pos) = -\sum_{\forall g \in G - G_s} dead \times (1 - d(pos, p(g))) + \sum_{\forall g \in G_s} ghost \times (1 - d(pos, p(g))) +$$
$$+ \sum_{\forall c \in C} (ghost \times |G - G_s|) \times (1 - d(pos, p(c))) + \sum_{\forall f \in F} (food + (\frac{win}{|F|})) \times (1 - d(pos, p(f)))$$

10.3 Estado

Se han añadido métodos accesores para todas las distancias y posiciones mentadas, modificando los métodos usados por los agentes para tratar las posiciones en formato $(fila, columna)$.

Parte IV

NEAT

Esta parte trata el algoritmo elegido para el estudio, *NEAT*. Para ello se detallan por separado el algoritmo en sí, y a continuación como se ha implementado para interactuar con la implementación propia de *Pac-Man*.

11 Definición

NEAT (Neuro Evolution of Augmented Topologies) es un algoritmo neuroevolutivo desarrollado por **Ken O. Stanley**[17] en el año 2002 para la ***University of Texas*** en *Austin, TX*. El objetivo de ésta sección es explicar el funcionamiento del algoritmo así como describir algunas de sus particularidades.

Originalmente se plantea cómo un método pensado para solventar algunos de los problemas que aparecen al tratar con ***TWEANNs***[30][18], concretamente los problemas que detectan son el denominado **problema de *competing conventions***, la **protección de la innovación** a fin de evitar pérdidas en la función de evaluación y, finalmente la **codificación de la población inicial**.

11.1 Algoritmo

NEAT usa una codificación directa para sus genomas, lo que significa que cada conexión y neurona de la red queda directamente representada a fin de facilitar el cruce entre genomas.

> *"NEAT's genetic encoding scheme is designed to allow corresponding genes to be easily lined up when two genomes cross over during mating. Genomes are linear representations of network connectivity. Each genome includes a list of connection genes, each of which refers to two node genes being connected. Node genes provide a list of inputs, hidden nodes, and outputs that can be connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an innovation number, which allows finding corresponding genes."*
>
> *(Stanley, Kenneth O. and Miikkulainen, Risto*
> *The MIT Press Journal - Evolutionary Computation 10(2): 107)*

[30] Red Neuronal de Topología y Peso Evolutivos, por sus siglas en inglés Topology and Weight Evolving Artificial Neural Network

Respecto a las mutaciones, se trabaja con dos tipos, mutaciones en el **peso de una conexión** o mutaciones **estructurales**.

Las mutaciones del **peso de una conexión** alteran el peso de la misma de forma aleatoria dentro de unos márgenes establecidos.

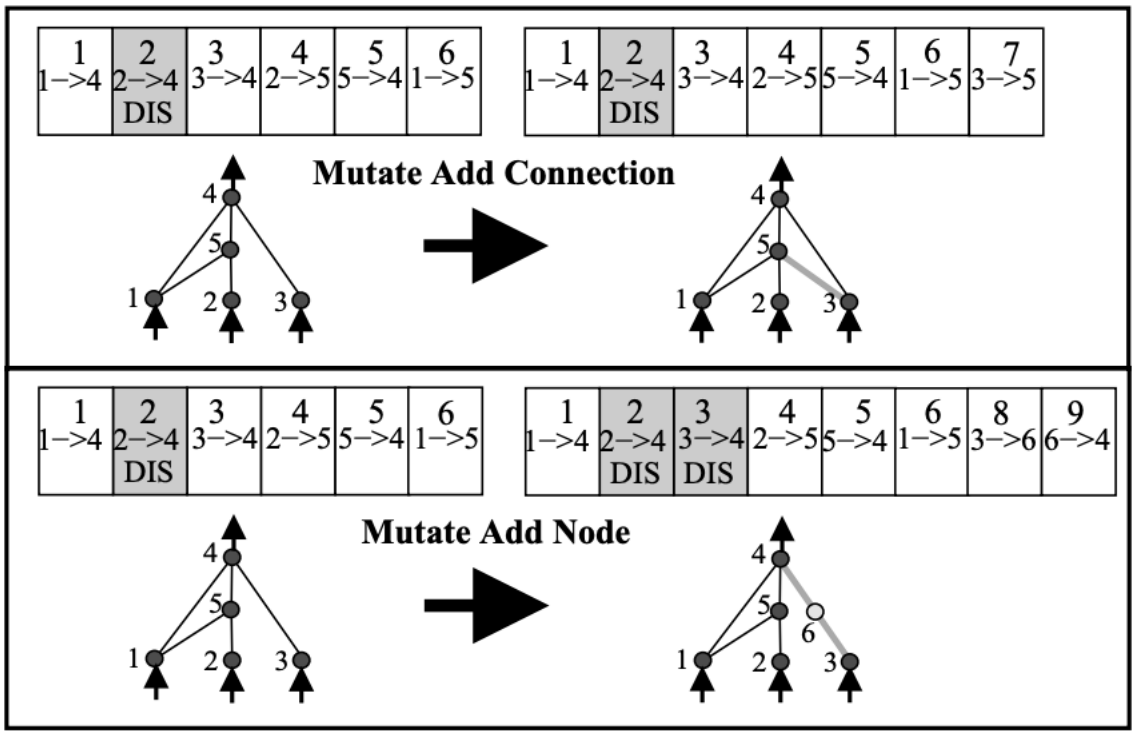


Figura 15: Representación gráfica de las mutaciones estructurales de *NEAT*

Fuente: The MIT Press Journal - Evolutionary Computation 10(2): 108

Las mutaciones **estructurales** sin embargo, pueden realizarse en dos formas distintas. Por un lado se puede añadir una conexión entre dos neuronas previamente no conectadas, asignando a esta un peso al azar dentro de los márgenes preestablecidos. La otra posibilidad es la de añadir una neurona en una conexión ya existente, para ello se deshabilita la conexión original y se crean dos conexiones nuevas, la conexión entrante a la nueva neurona tiene un peso de 1 y la saliente mantiene el peso de la conexión original, se elige este método para minimizar el impacto.

> *"This method of adding nodes was chosen in order to minimize the initial effect of the mutation. The new nonlinearity in the connection changes the function slightly, but new nodes can be*

immediately integrated into the network, as opposed to adding extraneous structure that would have to be evolved into the network later."

(Stanley, Kenneth O. and Miikkulainen, Risto
The MIT Press Journal - Evolutionary Computation 10(2): 108)

11.2 Alineamiento de genomas

El problema de ***competing conventions*** es uno de los principales problemas a afrontar al tratar con neuroevolución. Este surge al tener dos genomas que codifican la misma función de manera completamente diferente, esto se exacerba al tratar con ***TWEANNs*** puesto que dos genomas cuyas respectivas topologías son totalmente diferentes pueden, internamente, codificar soluciones muy similares e incluso idénticas, adicionalmente esta situación tiende a provocar la pérdida de información al cruzar los individuos.

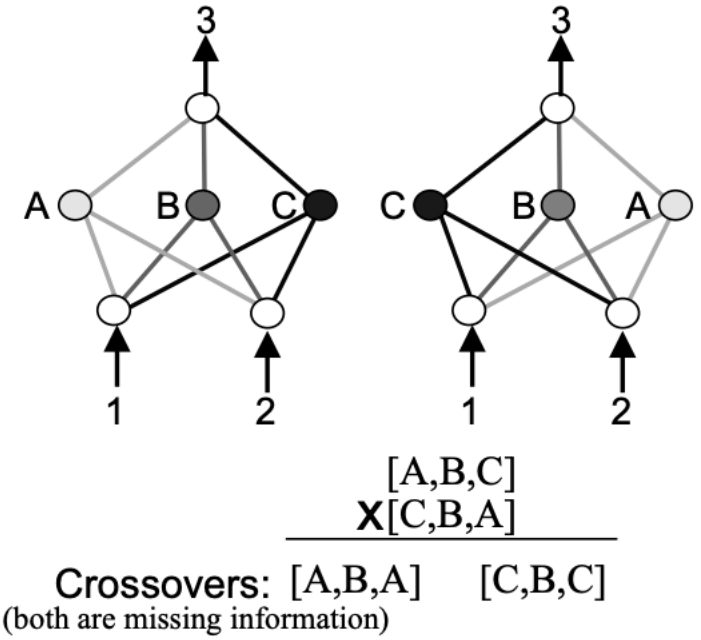


Figura 16: Representación del problema de ***competing conventions***
Fuente: The MIT Press Journal - Evolutionary Computation 10(2): 103

Para evitar este problema, *NEAT* usa un identificador global de innovación que se incrementa y asigna a cada nueva conexión creada, de este modo a la hora de cruzar dos individuos, se alinean sus conexiones siguiendo el identificador. Esto permite pasar todas las conexiones no alineadas automáticamente y elegir mediante un cruce tradicional que conexiones de cada padre pasan a la descendencia en las alineadas.

"When crossing over, the genes in both genomes with the same innovation numbers are lined up. These genes are called matching genes. Genes that do not match are either disjoint or excess... They represent structure that is not present in the other genome. In composing the offspring, genes are randomly chosen from either parent at matching genes, whereas all excess or disjoint genes are always included ..."

(Stanley, Kenneth O. and Miikkulainen, Risto
The MIT Press Journal - Evolutionary Computation 10(2): 108)

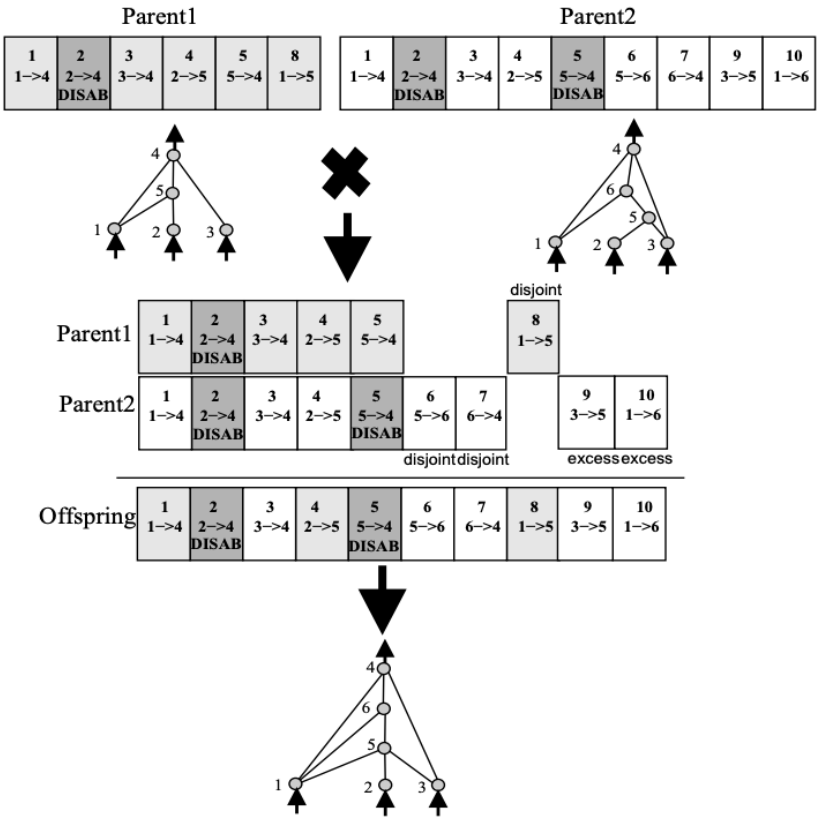


Figura 17: Representación del alineamiento para el cruce
Fuente: The MIT Press Journal - Evolutionary Computation 10(2): 107

11.3 Especiación

NEAT presenta ademas la idea de la especiación, esta consiste en agrupar a los individuos similares en grupos, o especies, manteniendo los coeficientes de cruce intra-especie considerablemente superiores que los de cruce inter-especie. Se pretende con ello segmentar la población en nichos. Cada

nicho se puede considerar un vector de deriva evolutiva diferente, pretendiendo con este mecanismo facilitar la exploración de ciertas regiones del espacio de soluciones.

> *"...This way, topological innovations are protected in a new niche where they have time to optimize their structure through competition within the niche. The idea is to divide the population into species such that similar topologies are in the same species..."*
>
> *(Stanley, Kenneth O. and Miikkulainen, Risto*
> *The MIT Press Journal - Evolutionary Computation 10(2): 109-110)*

11.4 Codificación inicial

A diferencia de otros sistemas ***TWEANN*** cuya población inicial está generada totalmente al azar, *NEAT* opta por inicializar todos sus individuos en su estructura mínima, *i.e.* únicamente la capa de entrada y salida, completamente conectadas entre ellas y con el peso elegido de manera aleatoria para cada conexión.

> *"...New structure is introduced incrementally as structural mutations occur, and only those structures survive that are found to be useful through fitness evaluations. In other words, the structural elaborations that occur in NEAT are always justified. Since the population starts minimally, the dimensionality of the search space is minimized, and NEAT is always searching through fewer dimensions than other TWEANNs and fixed-topology NE systems...."*
>
> *(Stanley, Kenneth O. and Miikkulainen, Risto*
> *The MIT Press Journal - Evolutionary Computation 10(2): 111)*

12 Implementación

A lo largo de esta sección se detalla el trabajo realizado en relación con el algoritmo *NEAT* tratado anteriormente.

Para ello se usa como marco de trabajo la librería *NEAT-Python*[31]. Se trata de una librería que provee la funcionalidad requerida para el desarrollo, permitiendo una alta personalizaron de los genomas (en el anexo B se muestra un ejemplo de fichero de configuración para la librería).

Se detallan los diversos genomas probados, así como los tratamientos hechos sobre los datos de entrada de la red y las diversas funciones de evaluación usadas.

12.1 Genomas

Esta sección pretende detallar de forma pseudo-cronológica la evolución de los diversos genomas con los que se ha experimentado. Para ello, de cada uno se detalla su estructura y una explicación básica del motivo del cambio respecto al anterior. Los experimentos y resultados realizados para evaluar dichos cambios se hallan en la parte V.

12.1.1 Original

Originalmente se plantea un genoma basado en la interpretación mas purista de $NEAT$, partiendo de una red que dispone únicamente de capa de entrada y salida, completamente conexa. Este dispone de 34 entradas y 5 salidas organizadas como sigue:

Siete neuronas entrantes por dirección (norte, sur, este, oeste) que reciben respectivamente:

- Distancia, en linea recta, a la que se halla el obstáculo, *i.e.* muro, mas cercano en esa dirección.
- Distancia, calculada como el camino mínimo, a la que se halla la cápsula mas cercana en esa dirección.
- Distancia, calculada como el camino mínimo, a la que se halla la pieza de comida mas cercana en esa dirección.
- Distancia, calculada como el camino mínimo, a la que se halla *Blinky* en esa dirección.

[31] Mas información en: <https://neat-python.readthedocs.io/en/latest/neat$_o verview.html$>

- Distancia, calculada como el camino mínimo, a la que se halla $Pinky$ en esa dirección.
- Distancia, calculada como el camino mínimo, a la que se halla $Inky$ en esa dirección.
- Distancia, calculada como el camino mínimo, a la que se halla $Clyde$ en esa dirección.

Seis neuronas entrantes que proveen información sobre el estado del juego:

- Estado de $Blinky$.
- Estado de $Pinky$.
- Estado de $Inky$.
- Estado de $Clyde$.
- Cantidad de comida restante en el mapa.
- Cantidad de cápsulas restantes en el mapa.

Cinco neuronas salientes, una por acción a realizar por el agente (una por dirección y la opción de no moverse).

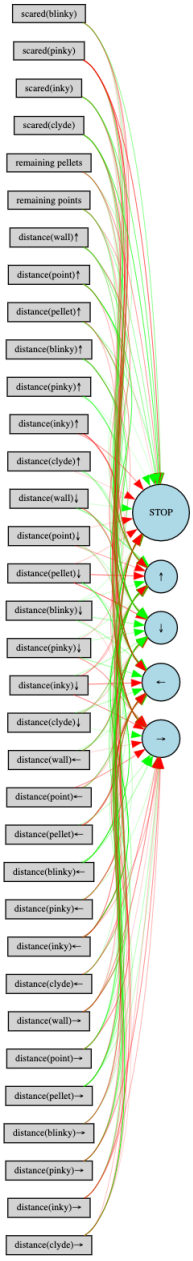


Figura 18: Ejemplo del genoma original

Fuente: Elaboración propia

12.1.2 Genoma propio

Se decide entonces probar si dar estructura a la red facilita su rendimiento. Para ello se deciden organizar la capa de entrada en cinco bloques claramente diferenciados, cuatro direccionales, compuestos por las siete neuronas que proveen información respecto a la misma dirección, y uno de estado, compuesto por las seis neuronas que proveen información sobre el estado del juego.

Se modifica la conexión inicial de la red alejándose del grafo $K_{i,o}$ original de $NEAT$ y procediendo a conectar cada bloque direccional exclusivamente con la neurona de salida que representa su dirección. El bloque de estado y la neurona que representa la acción de quedarse quieto se mantienen completamente conexas con sus respectivas.

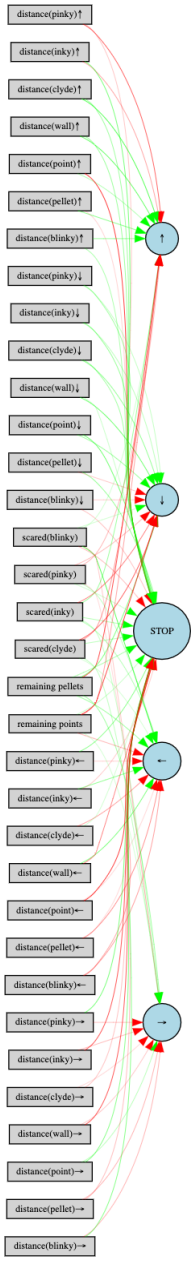


Figura 19: Esquema del genoma propio

Fuente: Elaboración propia

12.1.3 Simplificación de la salida

La siguiente evolución realizada sobre el genoma consiste en eliminar la opción de no realizar movimiento, acercando a su vez el comportamiento del mismo al del *Pac-Man* original.

Figura 20: Esquema del genoma sin opción a detenerse

Fuente: Elaboración propia

12.1.4 Unificación distancia-estado para fantasmas

El genoma final surge al considerar que mantener por separado la distancia a la que se halla un fantasma y su estado carece de sentido, pues se trata de variables claramente relacionadas, si el fantasma está **asustado** interesa, a priori, una distancia pequeña, mientras que en caso de hallarse en estado **normal** interesa una distancia grande.

Para ello se unifican las neuronas de estado, pasando únicamente a cambiar el signo de las entradas de distancia de los fantasmas, cuyo dominio pasa del $[0, 1]$ al $[-1, 1]$. Al mismo tiempo se decide eliminar las dos neuronas que indican la cantidad de cápsulas y comida restante en mapa, a fin de tener un modelo que trabaje con datos lo mas similares al heurístico posible.

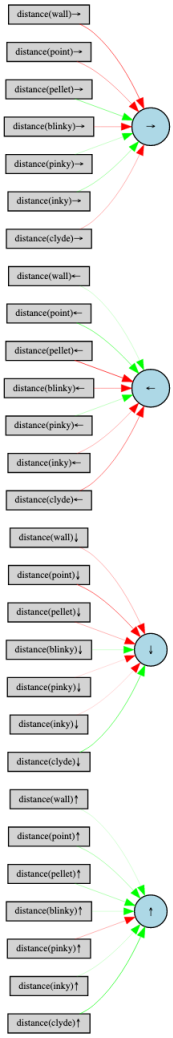


Figura 21: Esquema del genoma con unificación distancia-estado en las neuronas de los fantasmas

Fuente: Elaboración propia

12.2 Tratamiento de la entrada

Respecto a los datos de entrada, se hace un tratamiento a fin de proyectar todas las distancias en intervalos fijos, $[0, 1]$ y $[-1, 1]$ a fin de mantener la entrada consistente independientemente de las dimensiones del mapa con el que se juegue. Para ello se transforman los datos como sigue:

Se otorga al agente con un umbral de visibilidad (vis) comprendido en el intervalo $[0, 1]$, proyectando linearmente sobre el mismo los valores comprendidos en $[0, max(w, h) \times vis]$.

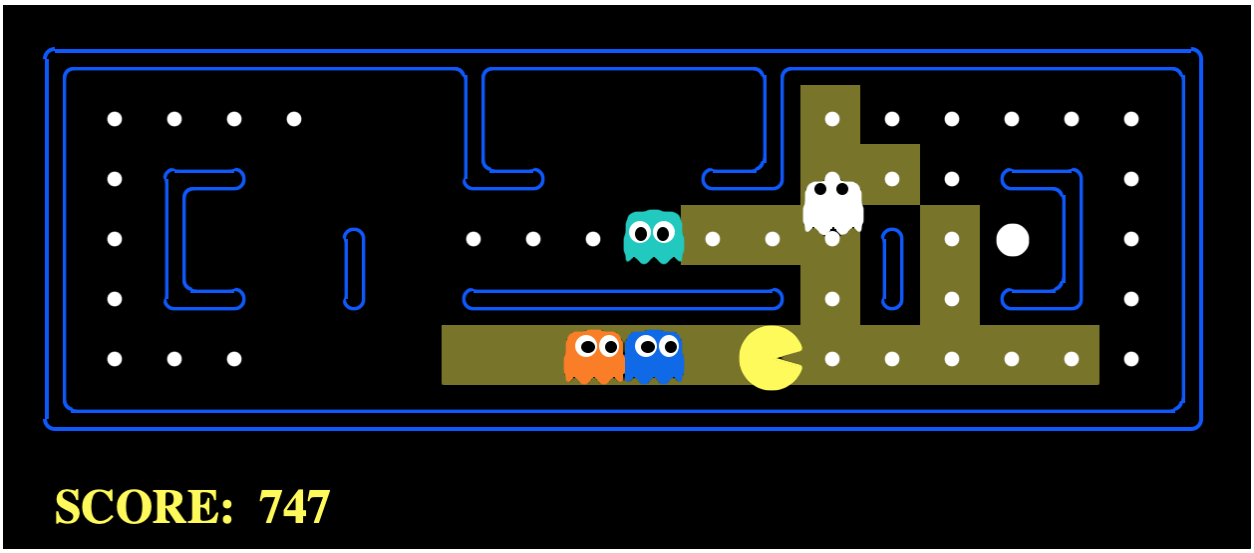


Figura 22: Ejemplo del rango de visión de *Pac-Man* con un valor de *vis* al *0.25*

Fuente: Elaboración propia

El estado de los fantasmas, antes de la unificación de estos con las neuronas de distancia, se trataban como un booleano, representando 1 el estado **asustado** y 0 el estado **normal**. Después de su unificación se consideran los valores en el intervalo $[0, 1]$ para el estado **asustado** y aquellos en el $[-1, 0]$ para el estado **normal**.

Finalmente, el número de cápsulas y comida que permanece en el mapa, se divide sobre la cantidad original de los mismos a fin de disponer de una representación de la parte proporcional que representa.

12.3 Evaluación

Análogamente a la evolución del genoma a lo largo del proyecto, la función de evaluación ha variado también. En esta sección se detallan las diversas formas que ésta ha tomado, el motivo de cada cambio se trata a la vez que se comentan los resultados de los experimentos, parte V.

Sea i un individuo cualquiera.
Sea g una generación cualquiera.
Sea $f(i, g)$ la evaluación del individuo i en la generación g.

12.3.1 Puntuación pura

La evaluación original consideraba únicamente el último juego jugado, matemáticamente se podría expresar como

$$f(i, g) = score(i, g)$$

Donde $score(i, g)$ representa la puntuación obtenida por el individuo i en el juego jugado en la generación g.

12.3.2 Puntuación normalizada

Se considera normalizar el valor de la puntuación obtenida sobre la total máxima del mapa. Por ello f procede a ser

$$f(i, g, m) = \frac{score(i, m, g)}{maxScore(m)}$$

Siendo m el mapa sobre el que se está entrenando, $score(i, m, g)$ la puntuación obtenida por el individuo i en el mapa m en el juego jugado en la generación g y, $maxScore(m)$ la puntuación máxima alcanzable en el mapa m, calculada como sigue

$$maxScore(m) = |food_m| \times foodBonus + |ghosts_m| \times ghostBonus + winBonus$$

Donde $food_m$ y $ghosts_m$ representan el conjunto de comida y fantasmas del mapa m respectivamente y, $foodBonus, ghostBonus, winBonus$ representan las sumas de puntuación obtenidas por comer un punto, comer un fantasma y ganar la partida respectivamente.

12.3.3 Puntuación respecto del máximo obtenible

La siguiente modificación a realizar se inspira en la formula del *MSE*[32] para predictores

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Siendo n el número de predicciones, Y el valor esperado y $\hat{Y}$ el valor observado. Pasando por ello f a

$$f(i, g, m) = (maxScore(m) - score(i, m, g))^2$$

Tomando las definiciones de m y $maxScore(m)$ dadas en la sección anterior.

12.3.4 Puntuación media

El siguiente paso parte de considerar que un único juego no es representativo del rendimiento de un individuo, por ello se decide hacer varios juegos por generación y considerar el total de juegos que ha jugado cada individuo a lo largo de su existencia, quedando f como

$$f(i, g) = \frac{\sum_{0...gen(i)} \frac{\sum_{j=0}^{k} score(i,j,g)}{k}}{gen(i)}$$

Siendo $score(i, j, g)$ la puntuación obtenida por el individuo i en la ejecución j-esima en la generación g y, k el numero de juegos que cada individuo juega por generación.

12.3.5 Puntuación media normalizada

Finalmente se decide mezclar las formulas vistas en las secciones 12.3.2 y 12.3.4 a fin de tener una función que permita evaluar un individuo en diversos mapas a la vez

$$f(i, g) = \frac{\sum_{0...gen(i)} \frac{\sum_{\forall m \in maps} \frac{\sum_{j=0}^{k} score(i,j,g)}{k}}{maxScore(m)}}{gen(i)}$$

Siendo *maps* el conjunto de mapas sobre el que se entrena y estando $maxScore(m)$ definido cómo en la sección 12.3.4

[32] Error Cuadrado Medio, por sus siglas en ingles Mean Squared Error

Parte V

Experimentación y resultados

En esta parte se detallan los experimentos realizados a lo largo del proyecto y se comentan sus resultados. Se tratan los experimentos a fin de ajustar parámetros del **agente heurístico** y una evolución sobre todas las funciones de evaluación de cada agente evolutivo.

13 Agente heurístico

Se define la función heurística que evalúa el beneficio potencial de desplazarse a una posición cómo

$$h_1(pos) = -\sum_{\forall g \in G - G_s} \frac{dead}{d(pos, p(g))} + \sum_{\forall g \in G_s} \frac{ghost}{d(pos, p(g))} + \sum_{\forall c \in C} \frac{ghost \times |G - G_s|}{d(pos, p(c))} + \sum_{\forall f \in F} \frac{food + (\frac{win}{|F|})}{d(pos, p(f))}$$

13.1 Ajuste de los parámetros

Originalmente se plantea usar como valores para las constantes de bonificación *dead, ghost, food, win* los mismos valores que los definidos en el código del juego, véase $500, 200, 10, 500$ respectivamente. No obstante se observa que el valor de $dead = 500$ provoca un comportamiento del agente inferior al esperado por ello se decide aproximar el valor óptimo. Para ello, usando el mapa ***smallClassic*** y la puntuación obtenida cómo indicador se procede a realizar una búsqueda lineal sobre las centenas a fin de hallar el punto de inflexión, realizando una búsqueda dicotómica sobre valores enteros una vez hallado éste. Los resultados así obtenidos son los siguientes

dead	100	200	225	250	300
Puntuación media	1235.49	1262.83	1218.49	1188.96	1217.64

Tabla 9: Efectos del valor de *dead* en el rendimiento. Valores mediados sobre 1000 juegos

Fuente: Elaboración propia

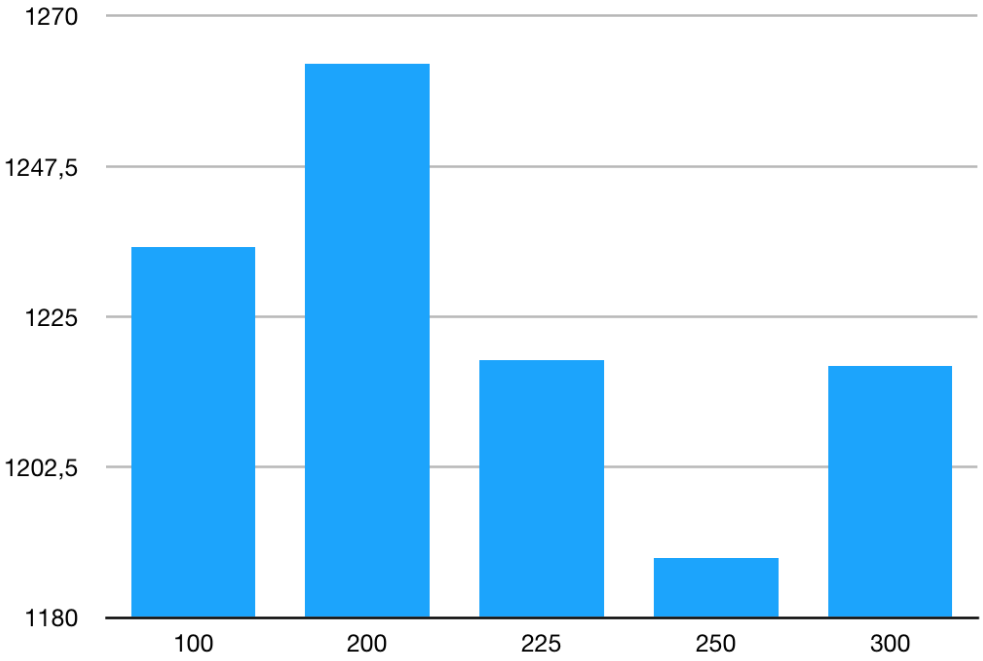


Figura 23: Visualización de la tabla 9

Fuente: Elaboración propia

Pese a no ser una búsqueda excesivamente exhaustiva y limitar los valores a rangos enteros, se decide dejar el valor de *dead* en 200. Con ello se prueba el agente en 3 mapas distintos, obteniendo los siguientes resultados

mapa	**% juegos ganados**	**puntuación media**
smallClassic	13.00	1222.00
mediumClassic	11.70	941.00
originalClassic	6.10	1007.00

Tabla 10: Resultados obtenidos con $dead = 200$. Valores sobre 1000 juegos

Fuente: Elaboración propia

Se observa que los valores para la puntuación media obtenida se mantienen relativamente similares independientemente del mapa, no así con el porcentaje de juegos ganados, que nunca llega a superar el 15%.

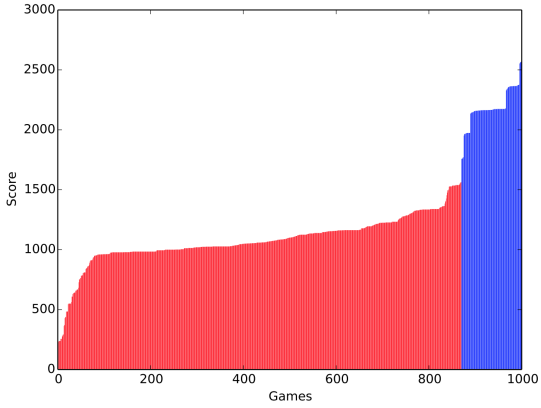


Figura 24: Resultados de la tabla 10 respecto al mapa *smallClassic*

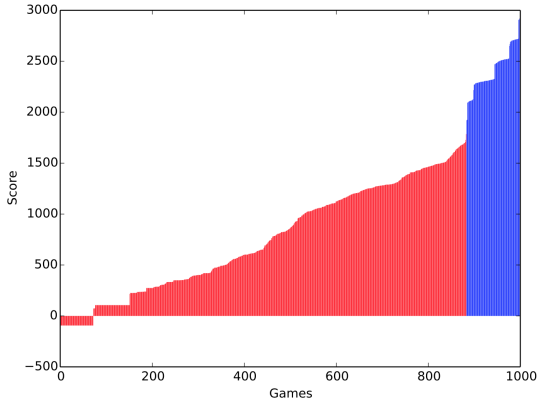


Figura 25: Resultados de la tabla 10 respecto al mapa *mediumClassic*

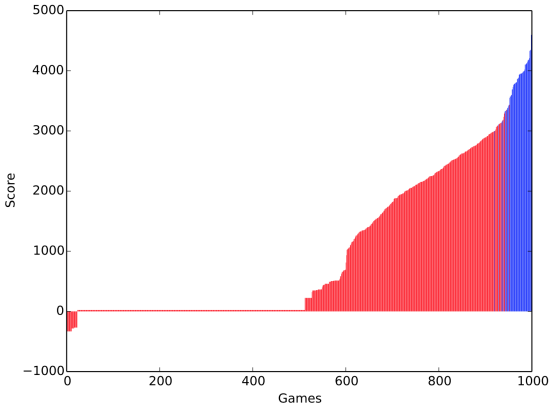


Figura 26: Resultados de la tabla 10 respecto al mapa *originalClassic*
Fuente: Elaboración propia

En las figuras 24, 25 y 26 el color indica el resultado del juego, rojo si se pierde, azul si se gana

13.2 $\frac{1}{d}$ frente $1 - d$

El siguiente ajuste a hacer sobre el agente heurístico consiste en determinar si se elige la formula original o la modificación

$$h_2(pos) = -\sum_{\forall g \in G - G_s} dead \times (1 - d(pos, p(g))) + \sum_{\forall g \in G_s} ghost \times (1 - d(pos, p(g))) +$$
$$+ \sum_{\forall c \in C} (ghost \times |G - G_s|) \times (1 - d(pos, p(c))) + \sum_{\forall f \in F} (food + (\frac{win}{|F|})) \times (1 - d(pos, p(f)))$$

El rendimiento obtenido con dicha modificación tras 1000 juegos en cada mapa es el siguiente

mapa	% juegos ganados	puntuación media
smallClassic	0.10	-180.00
mediumClassic	0.00	-65.00
originalClassic	0.00	-468.00

Tabla 11: Resultados obtenidos con la modificación. Valores sobre 1000 juegos

Fuente: Elaboración propia

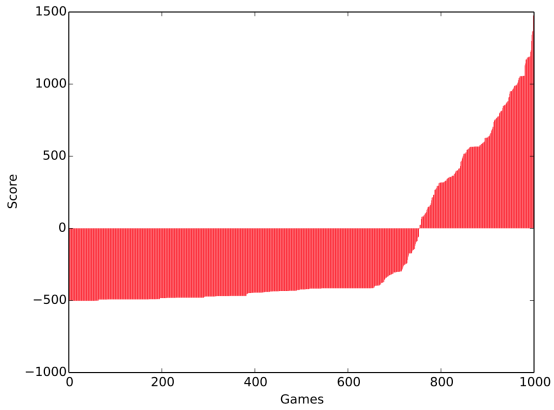


Figura 27: Resultados de la tabla 11 respecto al mapa *smallClassic*

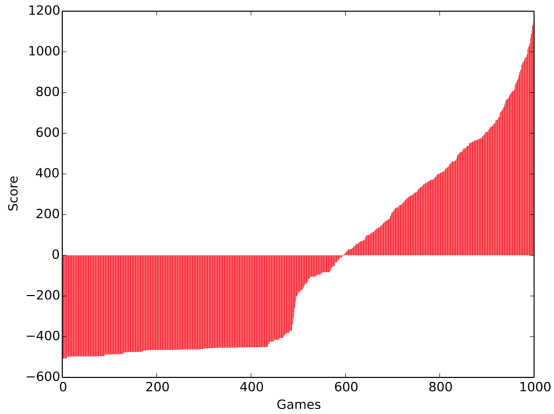


Figura 28: Resultados de la tabla 11 respecto al mapa *mediumClassic*

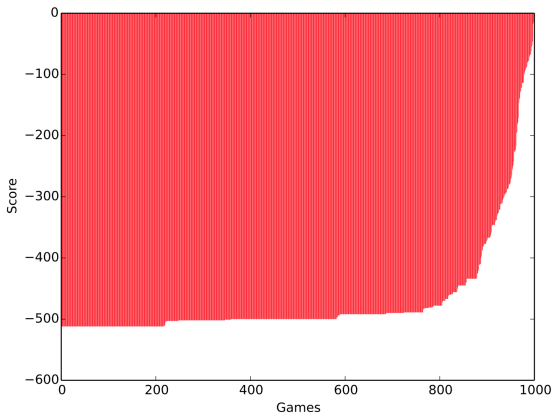


Figura 29: Resultados de la tabla 11 respecto al mapa *originalClassic*

Fuente: Elaboración propia

Queda claramente plasmado que la modificación a evaluar en este experimento presenta un rendimiento totalmente inaceptable.

Dados los resultados de ambos experimentos finalmente se decide optar por la función heurística original con el valor de $dead = 200$ como marco de referencia para el comportamiento de los agentes obtenidos vía neuroevolución.

14 Agente evolutivo

En esta sección se tratan los experimentos que han permitido mejorar la función de evaluación, organizados de forma cronológica según el desarrollo del proyecto.

14.1 Evaluación según último juego

El experimento se ha realizado considerando la evaluación de un individuo como la puntuación obtenida en el último juego que este juega. Se ha forzado la ejecución del algoritmo a 1000 generaciones independientemente del valor obtenido.

Adicionalmente se fija la población inicial a 1000 individuos y no se aplica tratamiento alguno a los datos que se proveen a la red. El resultado es el que sigue

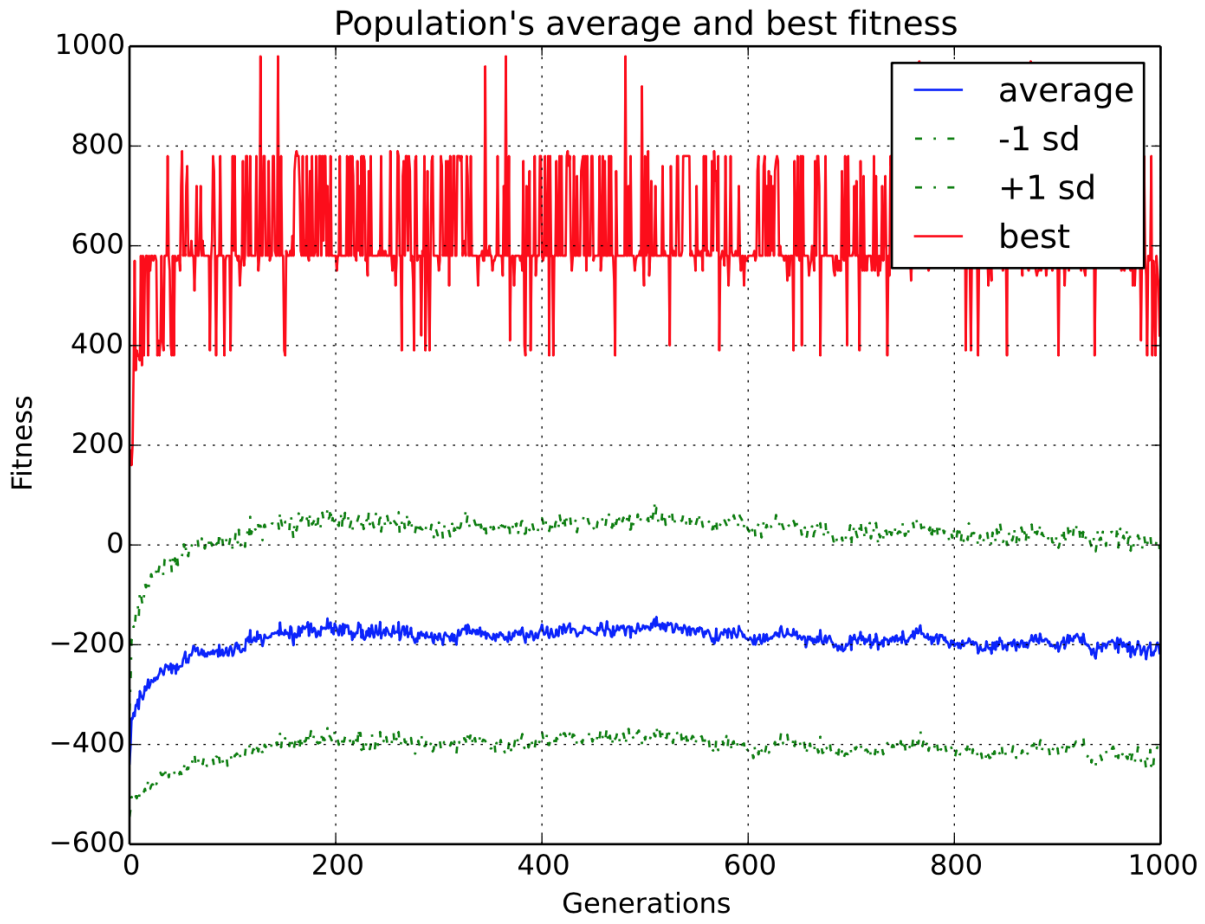


Figura 30: Visualización de la evolución de los valores de evaluación para el primer experimento.

Fuente: Elaboración propia

Contiene al mejor individuo de la generación (en rojo), la media poblacional (en azul) y la desviación de dicha media (en verde).

Se puede observar que la media poblacional se estanca rápidamente sobre un valor de -200 y se le empieza a intuir una caída sobre la generación 600, así como una enorme varianza en la evaluación del mejor individuo de cada generación, se cree que esto es dado a una mala elección para la evaluación ya que habiendo dos fantasmas con comportamiento aleatorio, un único juego para evaluar resulta insuficiente.

14.2 Evaluación normalizada

A continuación se prueba cambiando la evaluación para usar una normalización de la puntuación obtenida en relación a la puntuación máxima obtenida en el mapa. El resto de parámetros se mantienen como en el experimento anterior, obteniendo los siguientes resultados

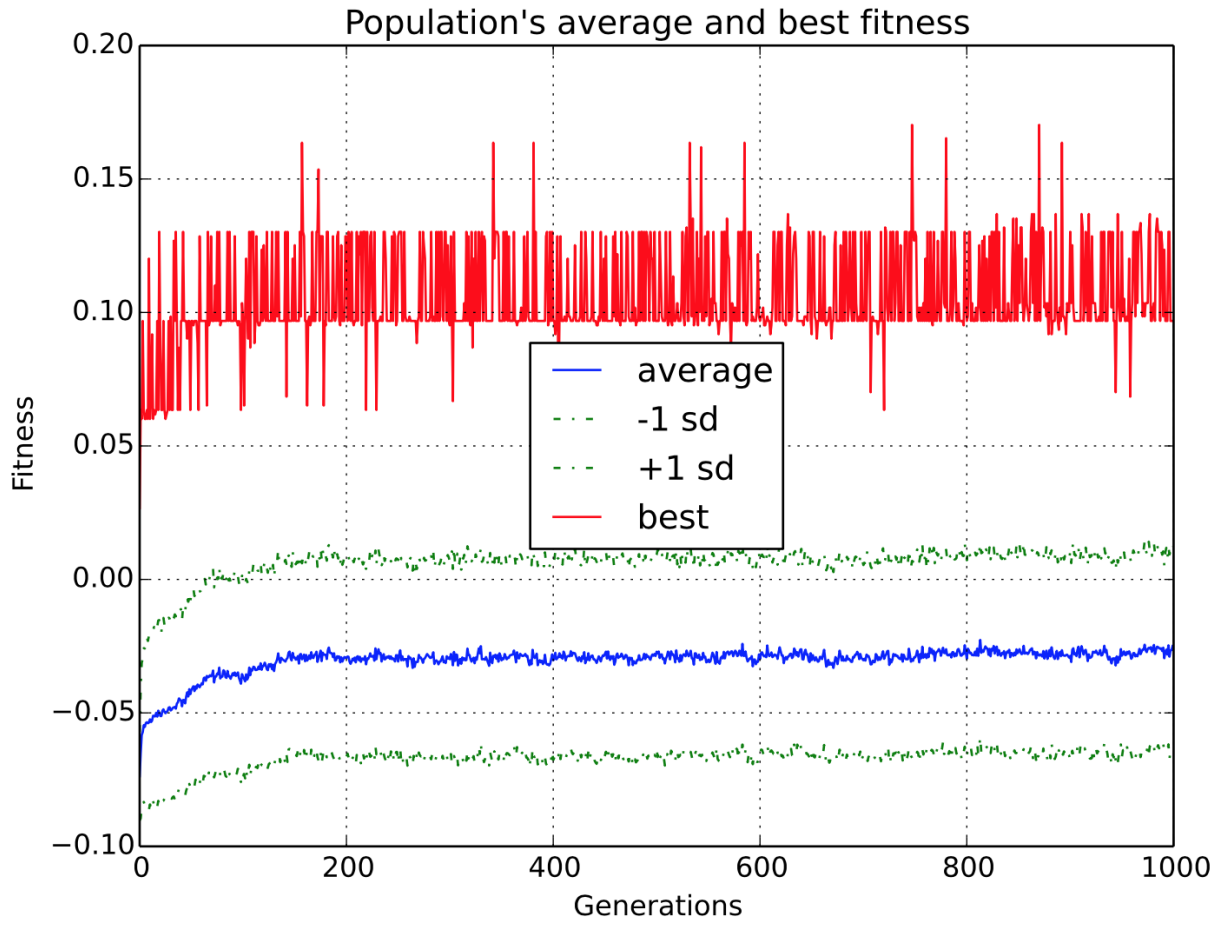


Figura 31: Visualización de la evolución de los valores de evaluación para el segundo experimento experimento

Fuente: Elaboración propia

Se observa cómo frente al experimento anterior, en este caso la media poblacional no decae, no obstante se sigue estancando rápidamente y el mejor individuo de cada generación sigue presentando mucha variación en sus resultados.

14.3 Evaluación respecto del máximo obtenible

El siguiente experimento pretende ver si optando por una evaluación usando como función la diferencia cuadrada respecto de la máxima puntuación obtenible en el mapa se logran solventar los problemas detectados en los dos experimentos anteriores. Los resultados obtenidos son

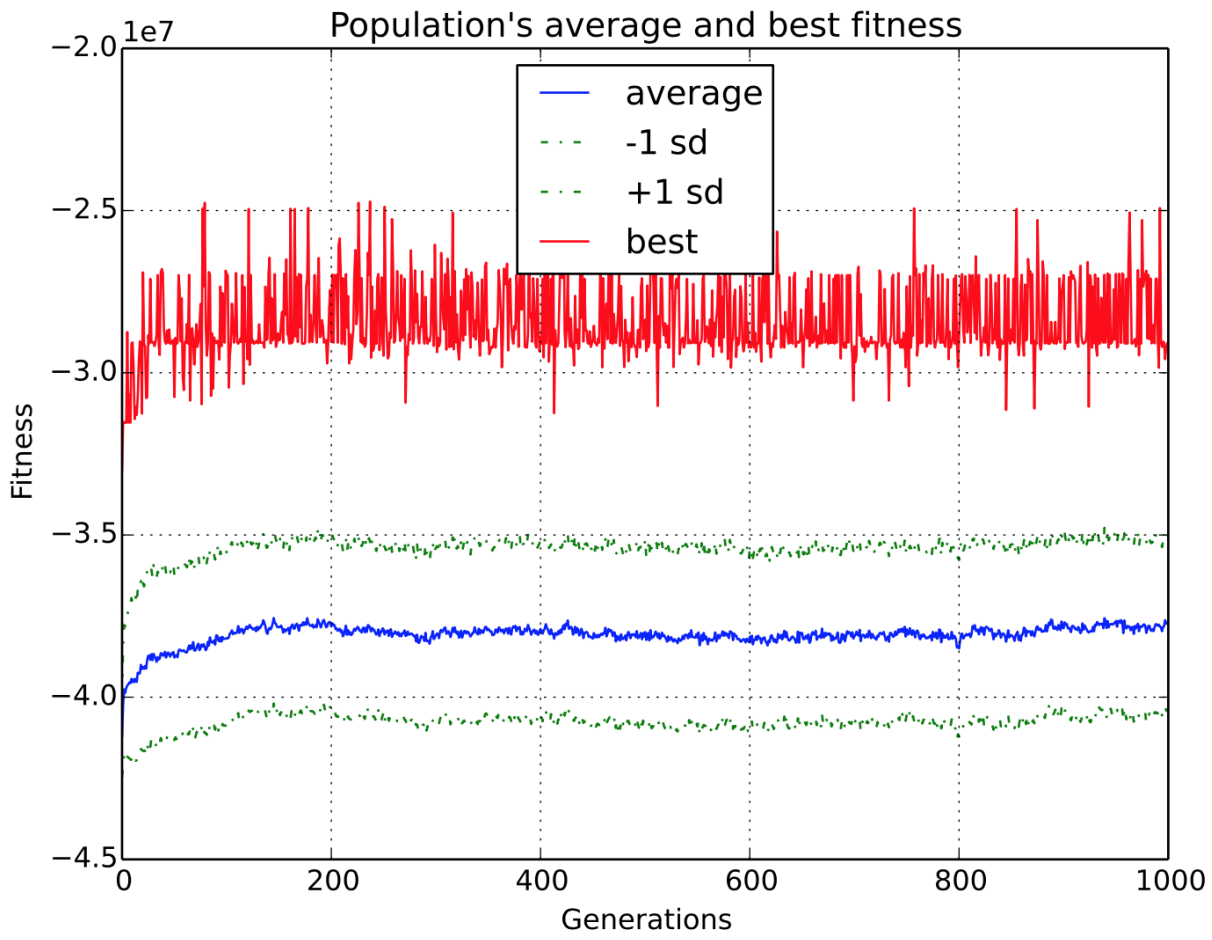


Figura 32: Visualización de la evolución de los valores de evaluación usando la diferencia respecto del máximo obtenible como indicador

Fuente: Elaboración propia

Desafortunadamente con éste cambio los resultados siguen mostrando los problemas encontrados anteriormente.

14.4 Evaluación según media histórica

Como consecuencia de la falta de avance tras estos experimentos se decide replantear el sistema de evaluación. Se opta por mantener un registro histórico de cada individuo a fin de mediar sus resultados y se añade un registro para mantener los datos del mejor individuo visto en todo el proceso evolutivo. Se procede también a tratar los datos que se proveen a la red, según lo ya especificado anteriormente.

Adicionalmente se detecta que el mismo experimento presenta gran variabilidad de resultados en diferentes ejecuciones del mismo, por lo que se decide realizar cinco ejecuciones distintas de cada experimento a fin de tener valores medios y horquillas de resultados.

14.4.1 Mutación estructural y *Stop* cómo salida válida

El primer experimento que se decide hacer bajo este nuevo marco de trabajo tiene como objetivo determinar el impacto real que tiene mantener la alternativa de **no moverse** como salida de la red, junto con la mutación estructural de la misma. Para este caso se considera mutación estructural toda aquella que altera la topología de la red.

Los resultados obtenidos son los siguientes

Parámetros	**Máximo**	**Media**	**Mínimo**	**Varianza**	**SD**
Mutación estructural y *Stop* como salida	647.00	512.30	368.00	279.00	16.70
Mutación estructural sin *Stop* como salida	912.00	311.80	122.00	790.00	28.11
Sin mutación estructural y *Stop* como salida	653.40	344.00	127.00	526.40	22.94
Sin mutación estructural y sin *Stop* como salida	772.00	425.20	183.00	589.00	24.27

Tabla 12: Resultados referentes al mejor individuo en la última generación

Fuente: Elaboración propia

Se observa cómo no contar con la acción *Stop* como posible salida de la red, causa los máximos mayores, pero a su vez maximiza las varianzas respecto de los mínimos. Así mismo la mejor media se consigue manteniendo tanto la mutación estructural como la salida referente a la acción de *Stop*.

Parámetros	**Máximo**	**Media**	**Mínimo**	**Varianza**	**SD**
Mutación estructural y *Stop* como salida	702.00	611.60	426.00	276.00	16.61
Mutación estructural sin *Stop* como salida	1025.00	546.10	239.00	786.00	28.04
Sin mutación estructural y *Stop* como salida	829.00	550.90	329.50	499.50	22.35
Sin mutación estructural y sin *Stop* como salida	1008.00	742.60	469.00	539.00	23.22

Tabla 13: Resultados referentes al mejor individuo respecto de toda la población

Fuente: Elaboración propia

Respecto al mejor individuo visto, parece que no permitir ni la mutación estructural ni la acción de *Stop* es la configuración que produce mejores resultados.

Parámetros	**Máximo**	**Media**	**Mínimo**	**Varianza**	**SD**
Mutación estructural y *Stop* como salida	87.58	-52.30	-141.79	229.37	15.15
Mutación estructural sin *Stop* como salida	37.90	-62.20	-109.20	147.10	12.13
Sin mutación estructural y *Stop* como salida	-62.67	-100.41	-123.22	60.55	7.78
Sin mutación estructural y sin *Stop* como salida	163.47	-25.00	-84.56	248.03	15.75

Tabla 14: Resultados referentes a la media poblacional

Fuente: Elaboración propia

Similarmente al anterior, parece ser que no permitir la mutación estructural ni la acción de *Stop* es la mejor guía para la evolución.

Adicionalmente, para validar los resultados se toman los genomas resultantes de cada ejecución y se les hace jugar un total de 1000 partidas, obteniendo los siguientes resultados

Parámetros	**Mejor Genoma**	**Peor Genoma**	**Varianza**	**SD**
Mutación estructural y *Stop* como salida	618.47	350.88	267.59	16.36
Mutación estructural sin *Stop* como salida	876.46	45.04	831.42	28.83
Sin mutación estructural y *Stop* como salida	590.70	88.88	501.82	22.40
Sin mutación estructural y sin *Stop* como salida	793.54	189.11	604.43	24.59

Tabla 15: Resultados referentes a los genomas resultantes

Fuente: Elaboración propia

Con estos resultados se opta por eliminar la acción de *Stop*.

Figura 33: Visualización de uno de los modelos con mejor rendimiento obtenido durante este experimento.

Fuente: Elaboración propia

En morado se observa la marca del mejor genoma obtenido hasta el momento y en negro el resultado medio del agente heurístico.

14.4.2 Mutación estructural y diversas funciones de activación

A continuación se realizan experimentos para comprobar el efecto tanto de la mutación estructural como diversas funciones de activación. Las funciones a comparar son la ***clamped*** frente a la ***softplus***.

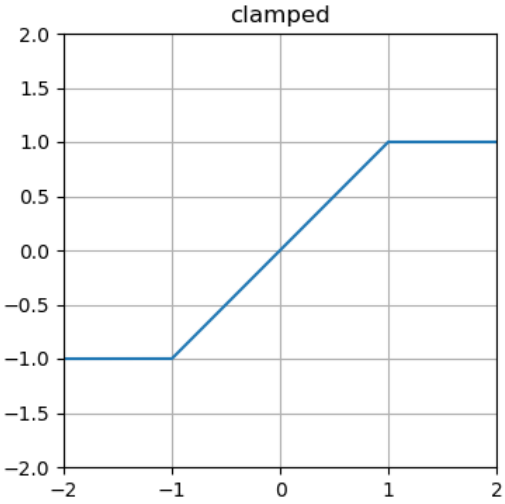


Figura 34: Función de activación ***clamped***

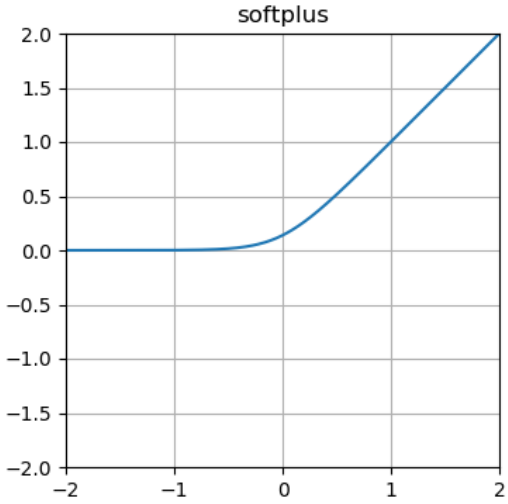


Figura 35: Función de activación ***softplus***

Fuente: NEAT-Python, <https://neat-python.readthedocs.io/en/latest/activation.html>

Se obtienen así los resultados siguientes

Parámetros	**Máximo**	**Media**	**Mínimo**	**Varianza**	**SD**
Mutación estructural y activación vía *clamped*	912.00	311.80	122.00	790.00	28.10
Mutación estructural y activación vía *softplus*	264.00	218.10	173.50	90.50	9.51
Sin mutación estructural y activación vía *clamped*	772.00	425.20	183.00	589.00	24.27
Sin mutación estructural y activación vía *softplus*	250.00	221.20	198.00	52.00	7.21

Tabla 16: Resultados referentes al mejor individuo en la última generación

Fuente: Elaboración propia

Parámetros	**Máximo**	**Media**	**Mínimo**	**Varianza**	**SD**
Mutación estructural y activación vía *clamped*	1025.00	546.10	239.00	786.00	28.04
Mutación estructural y activación vía *softplus*	354.00	301.00	253.00	101.00	10.05
Sin mutación estructural y activación vía *clamped*	1008.00	742.60	469.00	539.00	23.21
Sin mutación estructural y activación vía *softplus*	313.00	300.80	280.00	33.00	5.75

Tabla 17: Resultados referentes al mejor individuo respecto de toda la población

Fuente: Elaboración propia

Parámetros	Máximo	Media	Mínimo	Varianza	SD
Mutación estructural y activación vía *clamped*	37.90	-62.20	-109.20	147.10	12.13
Mutación estructural y activación vía *softplus*	-282.77	-297.73	-316.98	34.21	5.85
Sin mutación estructural y activación vía *clamped*	163.47	-25.00	-84.56	248.03	15.75
Sin mutación estructural y activación vía *softplus*	-274.65	-289.97	-301.85	27.20	5.22

Tabla 18: Resultados referentes a la media poblacional

Fuente: Elaboración propia

Parámetros	Mejor Genoma	Peor Genoma	Varianza	SD
Mutación estructural y activación vía *clamped*	876.46	45.04	831.42	28.83
Mutación estructural y activación vía *softplus*	147.50	55.60	91.90	9.59
Sin mutación estructural y activación vía *clamped*	590.70	88.88	501.82	22.40
Sin mutación estructural y activación vía *softplus*	168.00	-100.83	268.83	16.40

Tabla 19: Resultados referentes a los genomas resultantes

Fuente: Elaboración propia

En todos los casos se puede observar que la función ***softplus*** tiene un rendimiento notablemente inferior al de la ***clamped*** aunque, similarmente con resultados anteriores, presenta menor desviación en sus ejecuciones.

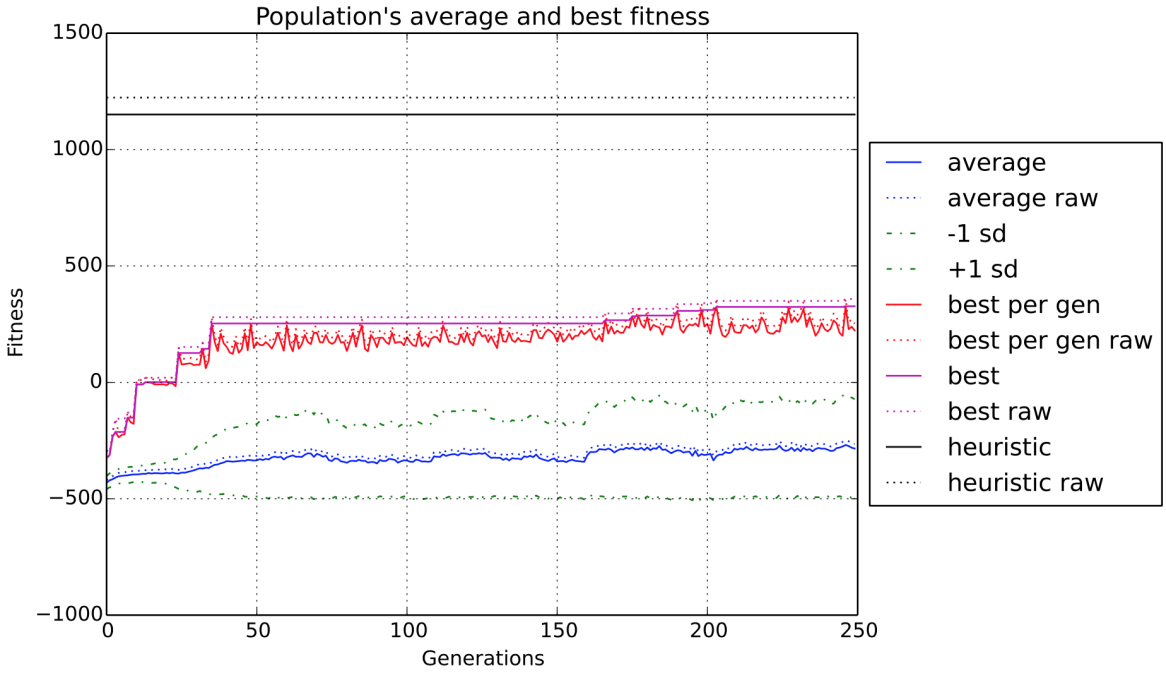


Figura 36: Visualización de uno de los modelos con mejor rendimiento, usando la función ***softplus*** como función de activación

Fuente: Elaboración propia

14.4.3 Efectos del tratamiento de la entrada

El siguiente experimento investiga los efectos que produce el tratamiento de la entrada para ello se decide usar cómo función de activación la ***sigmoide***, similar a la ***clamped*** pero derivable en todos los intervalos.

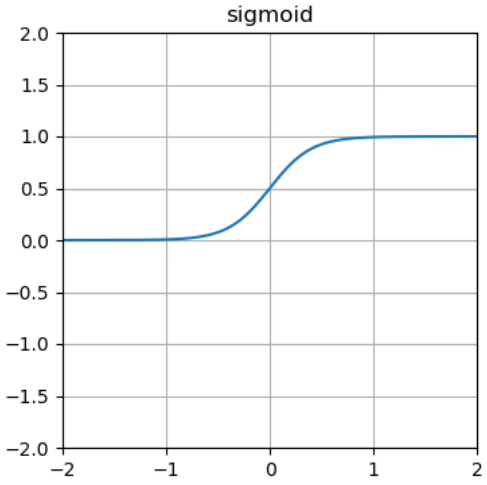


Figura 37: Función de activación ***sigmoide***

Fuente: NEAT-Python, <https://neat-python.readthedocs.io/en/latest/activation.html>

Los tratamientos de la entrada que se hacen para estos experimentos están inspirados en el funcionamiento de los agentes heurísticos ya definidos anteriormente en la sección 10.2.2. Por ello se unifica el estado de los fantasmas con sus distancias y se descartan las neuronas referentes a la cantidad de comida y cápsulas restantes en el mapa.

Los diferentes tratamientos que se realizan tienen como objetivo normalizar la entrada, bajo la premisa que interesa tener distancias cortas positivas a elementos que favorecen a *Pac-Man* (fantasmas asustados, comida, cápsulas) y distancias grandes y negativas a elementos que le perjudican (fantasmas en estado normal). Adicionalmente se aprovecha para discernir si la información sobre la distancia a los obstáculos en cada dirección favorece el comportamiento deseado.

Se obtienen los siguientes resultados

Tratamiento	Máximo	Media	Mínimo	Varianza	SD
$\frac{1}{d}$ considerando obstáculos	1197.00	912.00	728.00	469.00	21.66
$\frac{1}{d}$ sin considerar obstáculos	1153.00	314.55	-209.67	1362.67	36.91
$1-d$ considerando obstáculos	404.00	300.00	147.00	257.00	16.03
$1-d$ sin considerar obstáculos	-323.33	-360.20	-376.00	52.67	7.26

Tabla 20: Resultados referentes al mejor individuo en la última generación

Fuente: Elaboración propia

Tratamiento	Máximo	Media	Mínimo	Varianza	SD
$\frac{1}{d}$ considerando obstáculos	1675.00	1328.40	977.00	698.00	26.42
$\frac{1}{d}$ sin considerar obstáculos	1153.00	438.20	70.00	1083.00	32.91
$1-d$ considerando obstáculos	523.00	410.80	277.00	246.00	15.68
$1-d$ sin considerar obstáculos	-267.00	-306.20	-350.00	83.00	9.11

Tabla 21: Resultados referentes al mejor individuo respecto de toda la población

Fuente: Elaboración propia

Tratamiento	Máximo	Media	Mínimo	Varianza	SD
$\frac{1}{d}$ considerando obstáculos	167.02	115.53	51.37	115.65	10.75
$\frac{1}{d}$ sin considerar obstáculos	-18.09	-218.29	-349.00	330.91	18.19
$1-d$ considerando obstáculos	-295.28	-327.07	-349.23	53.95	7.34
$1-d$ sin considerar obstáculos	-383.02	-385.35	-387.31	4.29	2.07

Tabla 22: Resultados referentes a la media poblacional

Fuente: Elaboración propia

Analizando estos resultados respecto la afectación de la información de la distancia a los obstáculos queda claramente plasmado que dicha información ayuda a la obtención del comportamiento deseado, quedando éste efecto mas marcado realizando el tratamiento cómo $1-d$. A su vez se detecta que el tratamiento $\frac{1}{d}$ obtiene mejores resultados en general. Cabe destacar también que se repite el efecto ya comentado en otros experimentos por los que los modelos con peor rendimiento a su vez son los que presentan menor varianza.

Para finalizar este experimento se prueba el efecto de la mutación estructural de nuevo. Para ello se elige el modelo $\frac{1}{d}$ que considera la información de los obstáculos y se repite el proceso. Al hacer las pruebas de validación (jugar 1000 juegos) se obtienen los valores siguientes

Parámetros	Permitiendo mutación	Sin permitir mutación
Puntuación del Mejor Genoma	718.20	678.24
Puntuación del Peor Genoma	386.85	418.22
Varianza	331.36	260.03
SD	18.20	16.13
% juegos ganados del Mejor Genoma	65.00	65.00
% juegos ganados del Peor Genoma	0.00	9.00

Tabla 23: Resultados referentes a los genomas resultantes con el tratamiento $\frac{1}{d}$

Fuente: Elaboración propia

Cabe destacar que en el caso del mejor genoma se obtienen rendimientos similares, con una diferencia de puntuaciones medias inferiores a los 50 puntos y un % de juegos ganados idénticos en ambos casos, no obstante el limitar la deriva evolutiva a únicamente ajustar los pesos de la red parece tener efectos positivos, pues los peores genomas así obtenidos, pese a no diferir en exceso en sus puntuaciones, obtienen un ratio de juegos ganados del 9%.

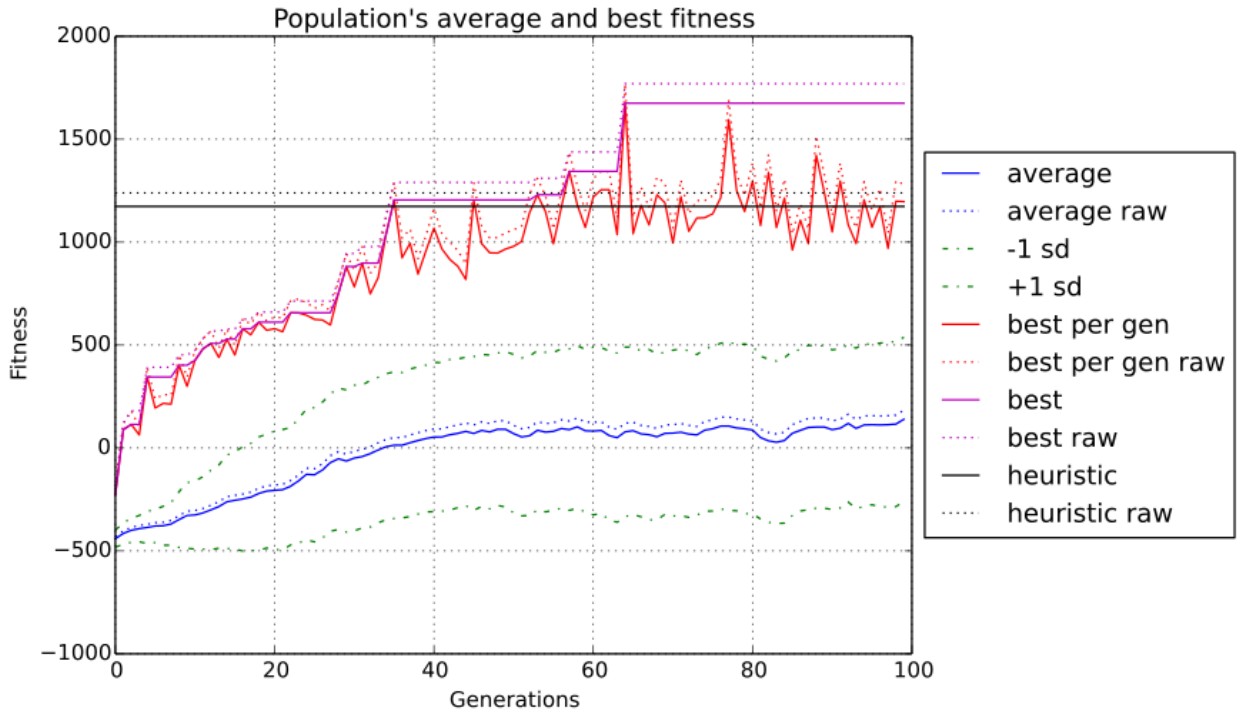


Figura 38: Visualización de la evolución para un modelo con mutación estructural

Fuente: Elaboración propia

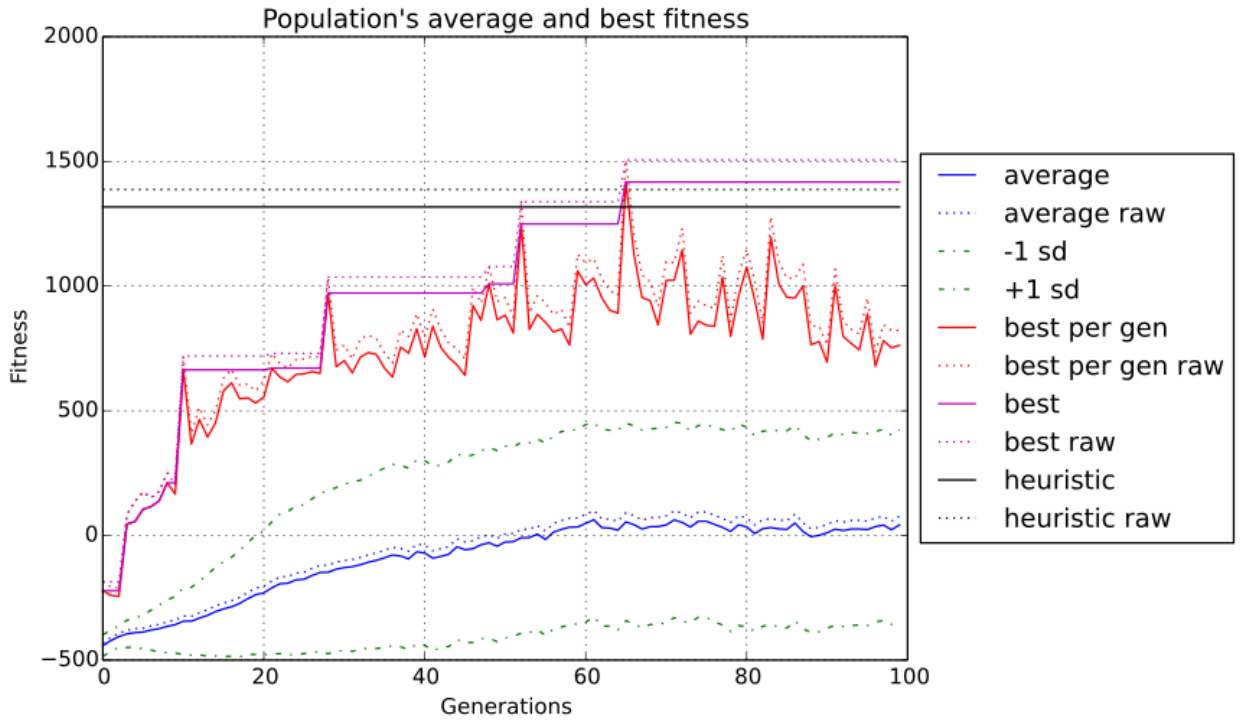


Figura 39: Visualización de la evolución para modelo sin mutación estructural

Fuente: Elaboración propia

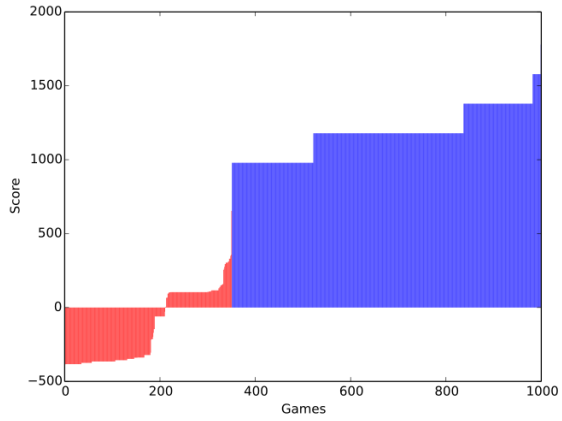


Figura 40: Representación de los 1000 juegos jugados cómo validación de un modelo con mutación estructural

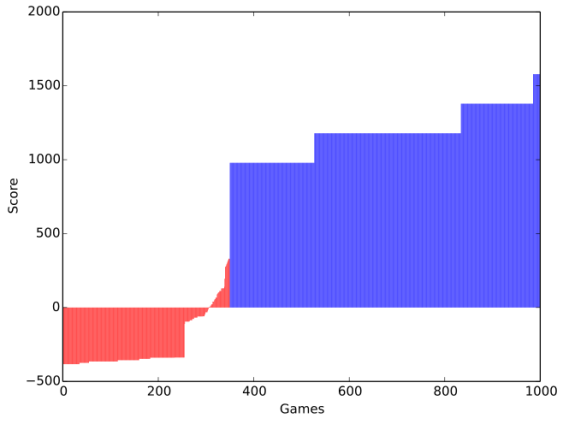


Figura 41: Representación de los 1000 juegos jugados cómo validación de un modelo sin mutación estructural

Fuente: Elaboración propia

14.4.4 Efectos del entrenamiento multimapa

Finalmente, el último experimento realizado refina la función de evaluación de los genomas a fin de permitir el entrenamiento en dos mapas distintos (*smallClassic* y *mediumClassic*). Analizando el rendimiento obtenido de este entrenamiento en relación con el comportamiento del agente heurístico desarrollado y el entrenamiento en un único mapa (*smallClassic*).

Haciendo jugar un total de 1000 juegos a cada modelo en cada mapa y usando tres mapas para ello a fin de tener también métricas del comportamiento en mapas en los que no se ha entrenado el modelo, se obtienen los siguientes resultados

Modelo	***smallClassic***	***mediumClassic***	***originalClassic***
Agente heurístico	1258.00	964.00	959.00
Modelo mono-mapa	726.00	-233.00	-356.00
Modelo bi-mapa	385.00	117.00	-317.00

Tabla 24: Puntuaciones medias para cada modelo

Fuente: Elaboración propia

Modelo	***smallClassic***	***mediumClassic***	***originalClassic***
Agente heurístico	16.80	11.90	5.90
Modelo mono-mapa	63.90	0.00	0.00
Modelo bi-mapa	0.00	0.00	0.00

Tabla 25: % juegos ganados para cada modelo

Fuente: Elaboración propia

Resulta destacable el hecho que para ambos modelos entrenados vía *NEAT* el rendimiento se desplome totalmente al ubicarlos en mapas en los que no se han entrenado pues, a priori, las *features* elegidas para las redes son transversales a cualquier mapa.

Así mismo se puede notar el incremento de dificultad en cada uno de los mapas usando al agente heurístico como referencia, consiguiendo éste únicamente ganar en sólo un 6% de los juegos en el último mapa.

Finalmente, destacar que el modelo mono-mapa, pese a tener un rendimiento considerablemente pobre en mapas nuevos, es capaz de ganar por lo menos en dos de cada tres partidas jugadas en el mapa sobre el que entrena. También hacer notar que, observando al modelo bi-mapa, se consiguen resultados relativamente similares en todos los mapas en los que entrena.

Destacar también el efecto que tiene el comportamiento desarrollado en el rendimiento de los agentes, como muestra la tabla siguiente

Comportamiento	**Puntuación**	**% juegos ganados**
Evitar fantasmas	63.90	726.00
Comer fantasmas	31.90	1020.00

Tabla 26: Efectos del comportamiento derivado

Fuente: Elaboración propia

Parte VI

Conclusiones

Desgraciadamente la exploración realizada durante el proyecto no ha permitido valorar todos los factores que influyen en el rendimiento del algoritmo *NEAT*, y el poco espacio explorado, tampoco se ha podido estudiar de forma exhaustiva, tomando muestras de sólo cinco modelos a cinco juegos por generación, a fin de extraer datos estadísticos que permitiesen iluminar que vertientes investigar a fin de intentar mejorar los resultados.

A la par, el necesitar alrededor de 300 horas de entrenamiento para aproximar el comportamiento de una función heurística, no excesivamente compleja en su evaluación, parece indicar que este método no es el más indicado para resolver el problema que plantea el juego de *Pac-Man*. Adicionalmente comentar el efecto de los entrenamientos usando diversos mapas, pues las *features* elegidas para la red deberían ser transparentes al mapa en el que se juega, cosa que no se ve reflejada en los resultados, ya que tanto la puntuación media como el porcentaje de juegos ganados caen completamente al intentar jugar sobre un mapa que no se conoce. No obstante el resultado del agente mono-mapa logra superar con creces al agente heurístico en el % de juegos que gana, no así en la puntuación obtenida.

Así mismo, dados los resultados de la tabla 26, se puede concluir que la decisión con mayor peso estratégico en el juego es en que momento comer los fantasmas asustados. Obteniendo peores puntuaciones, pero ganando más del doble de juegos, aquel agente que opta por usar las cápsulas como medida de seguridad para evitar morir, no comiendo fantasmas a no ser que sea imprescindible.

Cabe también destacar que resulta interesante el hecho de que los mejores rendimientos vía neuroevolución se hayan obtenido al no permitir la mutación estructural, cuando uno de los factores por los que se quiso investigar el algoritmo *NEAT* en concreto es la habilidad de mutar la red tanto en peso cómo en estructura. Este último factor podría estar marcado por la implementación del algoritmo elegida pues, en su concepción original, *NEAT* únicamente permite añadir nodos y conexiones, nunca quitarlos, cosa que sí que permite *NEAT-Python*.

Por lo comentado en el párrafo anterior, una vertiente exploratoria del proyecto a futuro sería

replicar el proceso con otro algoritmo neuroevolutivo. Se plantea $EANTT$[33] ya que este trabaja mediante la iteración de dos fases, claramente diferenciadas, refinando los pesos de la red ya existente hasta que no se puede mejorar más su rendimiento en la primera, procediendo a evolucionar la red ya optimizada en la segunda.

En relación a los resultados de la exploración, comentar que el mejor genoma obtenido presenta un coste lineal en su cálculo, pues al final se trata de un perceptrón, que en el fondo se podría considerar cómo cuatro distintos (uno por dirección), sobre los que se han ajustado los pesos a fin de obtener el mejor rendimiento posible.

En el campo personal, la realización de este estudio se considera fructífera, pues el objetivo autoimpuesto era adquirir conocimiento sobre las ideas de la neuroevolución, concretamente sobre el *NEAT*, y lograr aplicarlo sobre un videojuego. Ambas cosas se han conseguido, al mismo tiempo. La realización del proyecto ha permitido descubrir toda una vertiente de la AI de la que se desconocía la existencia, y a fecha de realización de este documento suscita un gran interés.

Referente al futuro, como ya se ha comentado, una clara linea de investigación es replicar el estudio aquí hecho con otros juegos u otros algoritmos neuroevolutivos, expandiendo también la fase exploratoria.

[33]Más información en: <http://www.siebel-research.de/evolutionary$_l earning$/>

Referencias

[1] Haykin, Simon. *Neural Networks and Learning Machines*,
Disponible en: <http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf>

[2] Apuntes de la asignatura IA. *Búsqueda Local [en línea]*
Disponibles en: <http://www.lsi.upc.edu/ bejar/ia/transpas/teoria/2-BH3-Busqueda_local.pdf>

[3] Wikipedia. *Neuroevolution [en línea]*
Disponible en: <https://en.wikipedia.org/wiki/Neuroevolution>

[4] Entertainment Software Association. *ESA Annual Report 2017 [en línea].*
Disponible en: <https://www.esaannualreport.com/>

[5] Game Developers Conference. *Getting Inquisitive About the AI of 'Dragon Age Inquisition' [en línea].*
Conferencia disponible en: <https://www.gdcvault.com/play/1023482/Getting-Inquisitive-About-the-AI>

Diapositivas disponibles en: <https://www.gdcvault.com/play/1023118/Getting-Inquisitive-About-the-AI>

[6] Neuro-Evolving Robotic Operatives [en línea].
Disponible en: <http://nn.cs.utexas.edu/nero/>

[7] MarI/O [en línea].
Disponible en: <https://github.com/wts42/Neat-Genetic-Mario/wiki/MarI-O>

[8] Galactic Arms Race [en línea].
Disponible en: <https://store.steampowered.com/app/249610/Galactic_Arms_Race/>

[9] ResetEra. *AI Neural Networks being used to generate HQ textures for older games (You can do it yourself!) [en línea].*

Disponible en: <https://www.resetera.com/threads/ai-neural-networks-being-used-to-generate-hq-textures-for-older-games-you-can-do-it-yourself.88272/>

[10] Youtube. *Video Game FMV's vs AI Machine Learning (Remastered Cutscenes | Artificial Intelligence Upscaling) [en línea].*
Disponible en: <https://www.youtube.com/playlist?list=PLTGEdY9AkWdVuhc54XFmH1-tgFliX9N67>

[11] Williams,Piers R.; Lucas, Simon M., IEEE, and Fairbank, Michael *The Effect of Varying Partial Observability in Ms. Pac-Man*
Disponible en: <http://www.pacmanvghosts.co.uk/downloads/journal.pdf>

[12] Thawonmas, Ruck and Matsumoto, Hiroshi *Automatic Controller of Ms. Pac-Man and Its Performance: Winner of the IEEE CEC 2009 Software Agent Ms. Pac-Man Competition*
Disponible en: <https://github.com/AI-Repository/PACMAN-AI/blob/master/paper.pdf>

[13] Gnanasekaran, Abeynaya; Feliu Faba, Jordi and An, Jing *Reinforcement Learning in Pacman*
Disponible en: <http://cs229.stanford.edu/proj2017/final-reports/5241109.pdf>

[14] Bigas Ortega, David *Machine Learning Applied to Pac-Man - Final Report*
Disponible en: <https://upcommons.upc.edu/bitstream/handle/2099.1/26448/108745.pdf>

[15] Merino Pulido, Albert Eduard *Automatically Configuring Deep Q-Learning agents for the Berkeley Pacman project*
Disponible en: <https://repositori.udl.cat/bitstream/handle/10459.1/64813/aemerinop.pdf>

[16] Fernández Villalba, Mario. *Estudi de l'algoritme NEAT aplicat als videojocs*
Disponible en: <https://upcommons.upc.edu/bitstream/handle/2117/127151/134627.pdf>

[17] *Kenneth O. Stanley personal page [en línea].*
Disponible en: <http://www.cs.ucf.edu/ kstanley/>

[18] Stantley, Kenneth O.; Miikkulainen, Risto (2002). *Evolving Neural Networks through Augmenting Topologies*, The MIT Press Journal, Evolutionary Computation 10(2): 99-127.
Disponible en: <http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf>

Anexo

A Implementación de la modificación de *Floyd-Warshall*

```
def calcGridDistances(self):
    # implements Modification of Floyd-Warshall
    board = [(r,c) for r in range(0, self.height) for c in range(0,self.width)]
    # for each vertex v: dist[v][v] <- 0
    for (row, col) in board:
        self.distances[row][col].grid[row][col] = 0
        # for each edge(u, v): dist[u][v] <- w(u, v)
        if(not self.isWall((col, row))):
            for (r, c) in self.__getAdjacentPositions(row, col):
                self.distances[row][col].grid[r][c] = 1
    # for k from 1 to |V|
    for (krow, kcol) in board:
        # for i from 1 to |V|
        for (irow, icol) in board:
            # for j from 1 to |V|
            for (jrow, jcol) in board:
                # if dist[i][j] > dist[i][k] + dist[k][j]
                self.updateDistances(irow, icol, jrow, jcol, krow, kcol)


def updateDistances(self, irow, icol, jrow, jcol, krow, kcol):
    if(self.distances[irow][icol].grid[jrow][jcol] >
        (self.distances[irow][icol].grid[krow][kcol]+self.distances[krow][kcol].grid[jrow][jcol])):
            self.distances[irow][icol].grid[jrow][jcol] =
                (self.distances[irow][icol].grid[krow][kcol]+self.distances[krow][kcol].grid[jrow][jcol])
```

B Ejemplo de un fichero de configuración para *NEAT-Python*

```
[NEAT]
pop_size              = 1000
# Note: the fitness threshold will never be reached because
# we are controlling the termination ourselves based on simulation performance.
no_fitness_termination = True
fitness_criterion     = max
fitness_threshold     = 1000.0
reset_on_extinction   = 0

[Pacman_Genome]
num_inputs              = 28
num_hidden              = 0
num_outputs             = 4
initial_connection      = full_direct
feed_forward            = True
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient   = 1.0
conn_add_prob           = 0.15
conn_delete_prob        = 0.1
node_add_prob           = 0.15
node_delete_prob        = 0.1
activation_default      = sigmoid
activation_options      = sigmoid
activation_mutate_rate  = 0.0
aggregation_default     = sum
aggregation_options     = sum
aggregation_mutate_rate = 0.0
bias_init_mean          = 0.0
bias_init_stdev         = 1.0
bias_replace_rate       = 0.02
bias_mutate_rate        = 0.8
bias_mutate_power       = 0.4
bias_max_value          = 30.0
bias_min_value          = -30.0
response_init_mean      = 1.0
response_init_stdev     = 0.0
response_replace_rate   = 0.0
response_mutate_rate    = 0.1
response_mutate_power   = 0.01
response_max_value      = 30.0
response_min_value      = -30.0

weight_max_value        = 30
weight_min_value        = -30
```

```
weight_init_mean        = 0.0
weight_init_stdev       = 1.0
weight_mutate_rate      = 0.8
weight_replace_rate     = 0.02
weight_mutate_power     = 0.4
enabled_default         = True
enabled_mutate_rate     = 0.0

[DefaultSpeciesSet]
compatibility_threshold = 3.0

[DefaultStagnation]
species_fitness_func = mean
max_stagnation       = 15
species_elitism      = 1

[DefaultReproduction]
elitism            = 20
min_species_size   = 10
survival_threshold = 0.2
```