Facultad de Ciencias

Uso del algoritmo NEAT aplicado a videojuegos TPS

(Use of the NEAT algorithm applied to TPS videogames)

Trabajo de Fin de Grado
para acceder al titulo de
Graduado en Ingeniería Informática

Autor: Juan Becerro Campos
Director: Jose Luis Montaña Arnaiz
Co-Director: Camilo Palazuelos Calderon
Curso 2022-2023

A Belén por su apoyo y cariño.

Resumen

En la actualidad, cada vez es más común el uso de técnicas de inteligencia artificial en los videojuegos. Esto se hace tanto para facilitar su desarrollo, como para modelar el comportamiento de los personajes que participan en ellos.

En este trabajo se ha implementado un algoritmo basado en computación evolutiva, concretamente en un concepto conocido como neuroevolución. Esto consiste en una estrategia de inteligencia artificial basada en el proceso evolutivo de la naturaleza con la que se intenta generar una red neuronal capaz de resolver un problema determinado. El algoritmo implementado, conocido como *Neuro Evolution of Augmenting Topologies* (NEAT) es un algoritmo basado en la neuroevolución que se caracteriza por no adherirse a una estructura fija de red neuronal, de manera que puede obtener mejores resultados que otros algoritmos más tradicionales.

Adicionalmente se ha desarrollado un prototipo de videojuego del género "third-person shooter"(TPS) en el que las características de las entidades enemigas del jugador son generadas mediante la red neuronal generada con NEAT.

Palabras clave: computación evolutiva, neuroevolución, red neuronal, NEAT, Inteligencia Artificial, TPS, videojuegos.

Abstract

Nowadays, the use of artificial intelligence techniques in video games is becoming more and more common. This is done both to facilitate their development and to model the behavior of the characters that participate in them.

In this work we have implemented an algorithm based on evolutionary computation, specifically on a concept known as neuroevolution. This consists of an artificial intelligence strategy based on the evolutionary process of nature with which an attempt is made to generate a neural network capable of solving a given problem. The implemented algorithm, known as Neuro Evolution of Augmenting Topologies (NEAT) is an algorithm based on neuroevolution that is characterized by not adhering to a fixed neural network structure, so that it can obtain better results than other more traditional algorithms.

Additionally, a prototype of a video game of the "third-person shooter"(TPS) genre has been developed in which the characteristics of the player's enemy entities are generated by means of the neural network generated with NEAT.

Keywords: evolutionary computing, neuroevolution, neural network, NEAT, Artificial Intelligence, TPS, video games.

Índice general

Índice de figuras

1. Introducción

En los últimos años, ha quedado evidente que la inteligencia artificial puede crear adversarios más capaces en el campo de los videojuegos que cualquier ser humano. Esto se ha demostrado a través de varios ejemplos emblemáticos, como la partida de ajedrez entre el supercomputador Deep Blue y el campeón mundial Garry Kaspárov en 1997 [1], el enfrentamiento entre AlphaGo y Lee Sedol en 2016 [2] y, más recientemente, la victoria de OpenAI Five en 2019 contra OG, que en ese momento era el equipo campeón del mundo en el videojuego Dota 2 [3]. Todos estos casos demuestran de manera contundente que la inteligencia artificial puede alcanzar niveles de habilidad y destreza mucho más ambiciosos que los métodos tradicionales.

La utilización de técnicas de inteligencia artificial en el campo de los videojuegos ha demostrado ser altamente efectiva y en general son campos que continúan evolucionando a un ritmo acelerado y es importante estar atentos a los desarrollos y discutir sus implicaciones.

El objetivo principal de este trabajo es implementar una versión del algoritmo NEAT, desarrollar un prototipo de videojuego y utilizar la implementación de NEAT para entrenar una red neuronal que equilibre las entidades enemigas adaptandose a las acciones del jugador, mejorando así la experiencia del usuario.

1.1. Antecedentes

En esta sección se van a listar algunos trabajos de investigación relacionados o similares al tema.

- *Towards adaptive online RTS AI with NEAT* [4]: Este trabajo presenta una aplicación del algoritmo NEAT para crear agentes inteligentes en videojuegos RTS[1].
- *A Minimal Training Strategy to Play Flappy Bird Indefinitely with NEAT* [5]: En este trabajo se utilizó NEAT para encontrar una red neuronal mínima que puede jugar el videojuego Flappy Bird a la perfección.
- *EvoCommander: A Novel Game Based on Evolving and Switching Between Artificial Brains* [6]: Este trabajo describe como en el videojuego EvoCommander, el jugador evoluciona una ANN que define el comportamiento de un sencillo robot el cual tiene que pelear contra el robot de otros jugadores.
- *Deduction of fighting game countermeasures using Neuroevolution of Augmenting Topologies.* [7]: En este trabajo se crea una ANN mediante el uso de NEAT para modelar el comportamiento de adversarios inteligentes en peleas de luchas, y comparan su eficiencia con un modelo ya existente entrenado mediante kNN[2].
- *Estudio del algoritmo NEAT aplicado al videojuego Pac-Man.* [8]: Este trabajo realiza un estudio sobre la eficiencia del algoritmo NEAT para jugar a una implementación del videojuego clásico Pac-Man.

Dado el alto grado de similitud con el trabajo [8], ya que ambos trabajos usan el algoritmo NEAT y lo aplican a un videojuego, se listan a continuación las principales diferencias con este.

En este trabajo se ha realizado una implementación de cero del algoritmo NEAT, adaptado para su funcionamiento en el programa Unity, a diferencia del trabajo mencionado, que utiliza una implementación de una librería.

Otra diferencia se puede encontrar en el objetivo principal del trabajo, este intenta utilizar NEAT para entrenar un modelo que equilibre a los adversarios de un prototipo de videojuego, el objetivo del trabajo mencionado es realizar un análisis de eficiencia del algoritmo NEAT para compararlo con otras estrategias.

Finalmente, en este trabajo se ha creado un prototipo de videojuego de cero, a diferencia del otro trabajo que utiliza una implementación del Pac-Man programada en python.

[1] Real Time Strategy
[2] Un algoritmo de Inteligencia artificial llamado k-Nearest Neighbour

2. Objetivos

El objetivo de este proyecto consiste en desarrollar una implementación del algoritmo NEAT, crear de un prototipo de videojuego, y finalmente utilizando el algoritmo NEAT entrenar un modelo que equilibre a los adversarios y estos se adapten al jugador así consiguiendo una mejor experiencia de juego.

El objetivo de este proyecto se puede dividir en tres partes:

- Creación del videojuego: Planificación, diseño y desarrollo de un prototipo de videojuego.
- Implementación del algoritmo NEAT: Desarrollar una implementación del algoritmo NEAT adaptable al objetivo del trabajo.
- Despliegue del algoritmo sobre el videojuego y experimentos: Entrenar el modelo utilizando la implementación y realizar experimentos para comprobar su funcionalidad.

3. Estado del arte

En esta sección se explica la base teórica sobre la cual partimos en este proyecto. Las bases teóricas relacionadas al algoritmo NEAT, son: redes neuronales, algoritmos genéticos, neuroevolución y NEAT. Por otro lado, se va a desarrollar qué es un videojuego, el estado de la industria de los videojuegos, los diferentes tipos de géneros existentes, el género Third Person Shooter (TPS) y finalmente la utilización hoy en día de la IA en videojuegos.

3.1. Redes Neuronales

Una red neuronal artificial es una estructura matemática que representa el procesamiento de información en sistemas biológicos [9]. Está compuesta por nodos o unidades interconectadas por enlaces direccionales. Un enlace desde el nodo i al nodo j sirve para transmitir la activación a_i del nodo i al nodo j. Cada enlace también tiene asociado un peso numérico $w_{i,j}$, que determina la intensidad y dirección de la conexión [10].

Se puede representar como un grafo dirigido formado por múltiples capas. Hay tres tipos de capas, la capa de entrada conocida como *input*, solo puede haber una y no contiene conexiones entrantes. La capa de salida, conocida como *output*, que al igual que la capa *input* solo hay una, pero esta no contiene conexiones salientes. Y finalmente la capa oculta, conocida como *hidden*, puede haber tantas capas de este tipo como se necesiten y tienen tanto conexiones de entrada como de salida.

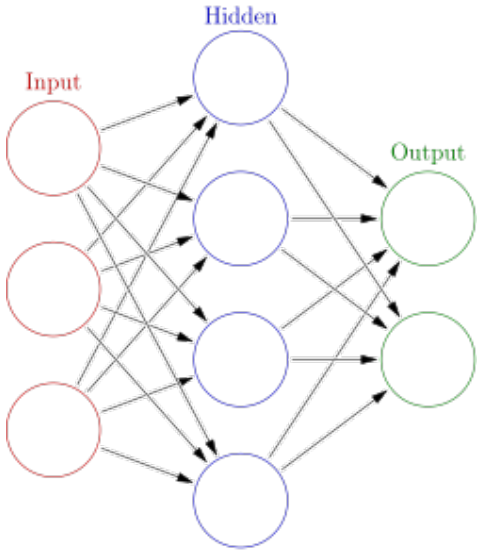

Figura 1: Imagen de una red neuronal. [11]

3.1.1. Redes Neuronales Profundas

Una red neuronal profunda es un modelo de aprendizaje automático que se inspira en la estructura de las redes neuronales. Está formada por 3 o más capas ocultas de neuronas interconectadas, donde cada capa procesa la información de entrada y envía la información procesada a la siguiente capa. Las primeras capas de la red neuronal profunda suelen aprender características más simples, como bordes y formas, mientras que las capas más profundas aprenden representaciones de la información más complejas y abstractas.

Una de las principales ventajas de las redes neuronales profundas es su capacidad para aprender características a partir de grandes cantidades de datos sin tener que ser programadas explícitamente. Esto las hace muy útiles para tareas que involucran patrones complejos y abstractos, como el reconocimiento de imágenes y el procesamiento del lenguaje natural. [12]

3.2. Algoritmos Genéticos

Un algoritmo genético es una variante del *stochastic beam search* con la diferencia de que las soluciones sucesoras son generadas al combinar dos soluciones padres en vez de modificando una sola. Estos algoritmos tienen cierto parecido con el proceso de selección natural ya que los sucesores (hijos) de las souciones (organismos) pueblan la siguiente generación en función de su valor (fitness), la diferencia es que con los algoritmos genéticos se utiliza una reprodución sexual en vez de una asexual. [13]

El algoritmo tiene varias fases. Primero se genera un set de k estados aleatorios llamado *población*. Después, se evalua cada estado de la población segun la *función de fitness*, la cual debe de retornar un mejor valor para los mejores estados. En función de este valor, cada estado tiene una probabilidad de ser elegido para la reproducción. A continuación, se eligen parejas de forma aleatoria utilizando los porcentajes calculados, estas parejas se cruzan y asi se generan dos nuevos estados hijos, se repite este paso hasta tener una nueva población con los hijos. Finalmente, cada estado de la nueva población tiene una pequeña probabilidad de mutar ligeramente.[13]

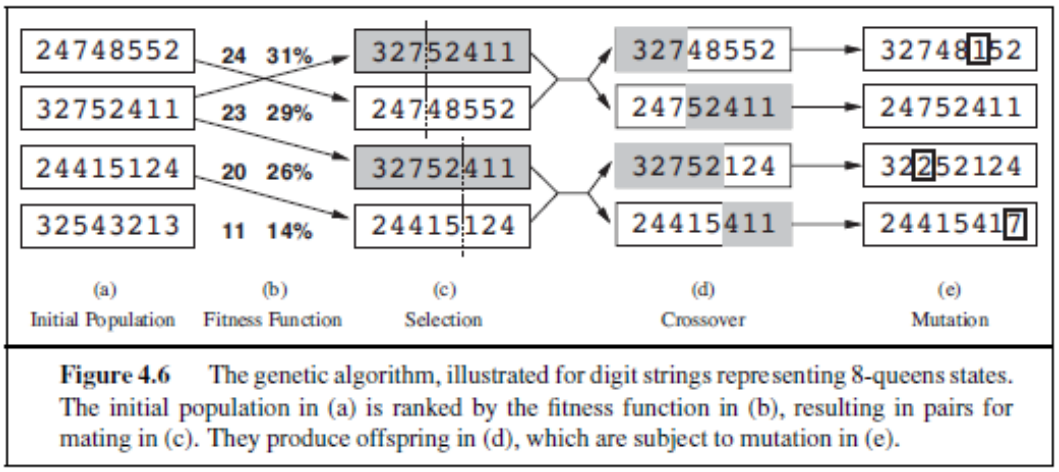

Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Figura 2: Ejemplo de fases de un algoritmo genético con el problema de las 8 reinas. [13]

Para explicar cómo y porqué funciona este algoritmo se usa la idea de un *esquema*, que es una representación parcial de un estado, un estado que lo contiene se denomina *instancia* de dicho esquema. Se puede demostrar que si el fitness medio de las instancias de este esquema es mayor que la media en la población, el número de instancias crecerá con el tiempo. [13]

3.3. Neuroevolución

La neuroevolución es una técnica de aprendizaje automático que aplica algoritmos genéticos para construir redes neuronales artificiales, inspirándose en la evolución de los sistemas nerviosos biológicos en la naturaleza. En comparación con otros métodos de aprendizaje de redes neuronales, la neuroevolución es muy general; permite el aprendizaje sin objetivos explícitos, con una retroalimentación escasa y con modelos neuronales y estructuras de red arbitrarios. La neuroevolución es un método eficaz para resolver problemas de aprendizaje por refuerzo y se aplica sobre todo en la robótica evolutiva y la vida artificial. [14]

Se puede distinguir entre los algoritmos que evolucionan sólo los pesos para una topología de red fija (a veces llamados neuroevolución convencional), frente a los que evolucionan tanto la topología de la red como sus pesos (llamados TWEANN, por Topology and Weight Evolving Artificial Neural Network algorithms). [14]

3.4. Utilización de la IA en videojuegos

La utilización de la inteligencia artificial en videojuegos es un campo de invesigación establecido que sigue creciendo y evolucionando rápidamente. Se estudia cómo automatizar jugar, diseñar, entender o adaptar juegos usando gran variedad de métodos de inteligencia artificial. [15]

Un ejemplo muy común de para qué tareas se utiliza la IA en videojuegos es para modelar el comportamiento de los *NPC*[3], lo cual se hace de las siguientes maneras: para navegar en el entorno, de manera que se utilizan algoritmos de búsqueda de caminos como A* para que un *NPC* pueda trasladarse por un espacio definido de manera natural, para encontrar un equilibrio en la dificultad del videojuego, de manera que el jugador no pueda adoptar una misma estrategia para "ganar"haciendo así que el juego deje de ser una experiencia gratificante, también se utiliza para que los *NPC* adquieran instinto de supervivencia o de cazador, es decir, no solo demostrando habilidad ofensiva y defensiva, sino de analizar el entorno y tomar decisiones como buscar a una presa o esconderse ante un potencial peligro.

Otros ejemplos son para la remasterización de juegos antiguos, donde se utilizan GANs para adaptar los gráficos de videojuegos antiguos a consolas más modernas, o también para la generación automática de recursos útiles para la creación de videojuegos, como texturas, terreno procedural, mapas, etc. [16]

[3]Non Player Character

4. Planificación

A continuación, se describe la planificación seguida para el desarrollo del proyecto. Adicionalmente, también se ha utilizado un software de gestión de versiones para monitorizar y gestionar el proyecto.

- Diseño del videojuego a implementar.
- Evaluación de soluciones software a utilizar en el desarrollo del proyecto.
- Desarrollo del videojuego.
- Estudio del funcionamiento del algoritmo NEAT.
- Implementación de NEAT.
- Desarrollo de la integración del videojuego con NEAT.
- Testeo de la implementación.
- Entrenamiento del modelo.
- Análisis de resultados.

4.1. Herramientas Utilizadas

En cuanto al hardware, se ha utilizado el ordenador de mesa del autor, las especificaciones relevantes del mismo son:

- CPU: AMD Ryzen 7 5700G
- GPU: Nvidia GeForce RTX 3070
- RAM: 32G, 3600 MHz

Para determinar las herramientas software a utilizar se han tenido que evaluar diferentes partes del proyecto, como el lenguaje de programación donde implementar el algoritmo o la *game engine* a utilizar para el videojuego.

En cuanto a la game engine, se consideraron como posibles motores Unreal Engine, Godot o utilizar librerías de creación de videojuegos de python como pygame, sin embargo se ha escogido Unity3D debido tanto a la gran documentación existente y a los tutoriales online como a la experiencia previa del autor utilizando este programa.

Respecto al lenguaje de programación para el algoritmo se ha optado por C#, debido a la preferencia de usar OOP y también debido a la game engine utilizada para el proyecto que usa C#.

Para la gestión de versiones inicialmente se iba a utilizar Git pero se ha optado por la utilización de PlasticSCM, ya que este está integrado en Unity3D y es entonces más sencillo de utilizar.

La documentación de este proyecto se ha realizado con Overleaf, que es una herramienta online para el procesamiento de texto LaTeX.

Finalmente el sistema operativo utilizado es Windows 10.

5. Videojuego

Al tener como objetivo mostrar las capacidades de NEAT para generar adversarios más inteligentes y adaptables al jugador, más que un juego, es un prototipo sobre el cuál en el futuro se puede considerar construir un videojuego.

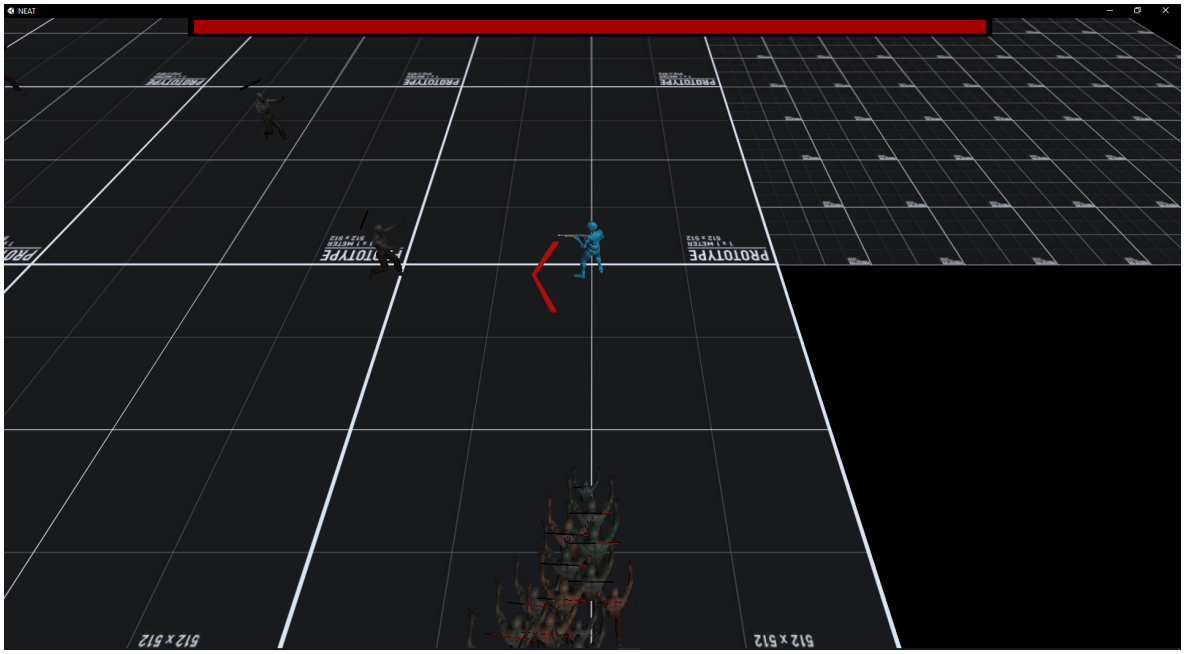

Figura 3: Captura del videojuego desarrollado.

5.1. Concepto

Este es un juego de supervivencia por oleadas, donde el jugador tendrá que luchar contra hordas de enemigos que a cada ronda que avanza se hacen más fuertes.

5.2. Características Principales

- Planteamiento sencillo: las reglas del juego son relativamente sencillas debido a la naturaleza del juego en sí, ya que su propósito no es ser un juego comercial sino una prueba de concepto. Las reglas sencillas hacen que el jugador aprenda las mecánicas importantes más rápido.
- Dinamismo: se pretende que el juego sea dinámico y produzca una sensación de tensión en el jugador.

5.3. Género

- Acción-Supervivencia: Subgénero de los videojuegos de acción ambientado en un ambiente hostil, intenso y de mundo abierto, donde los jugadores generalmente comienzan con equipos mínimos y se les exige que recolecten recursos, herramientas de artesanía, armas y refugio, y sobrevivan el mayor tiempo posible [17]. En este caso no tenemos un mundo abierto pero sí que se pone al jugador en un ambiente hostil e intenso donde el jugador recolecta recursos para sobrevivir.
- Disparos en tercera persona: subgénero del género de Acción, shooter (o disparos) en los que la mecánica principal es disparar. Es similar al subgénero de disparos en primera persona, con la diferencia de que el personaje del jugador esta visible. [18]

5.4. Jugabilidad

En cada ronda aparecerán un número fijo de zombies, el jugador tendrá que sobrevivir utilizando los siguientes recursos:

- Escopeta: Dispara en un área delante del jugador y cada perdigón hace x daño al/los zombie/s que golpea.
- Subir de nivel: Cada ronda el jugador podrá subir de nivel y mejorarse una habilidad.
- Movimiento: El jugador podrá moverse con libertad por el mapa evadiendo los zombies y así posicionarse mejor.

5.5. Personajes

Aquí se describen los personajes implicados en el juego.

5.5.1. Jugador

El objetivo del jugador es el de sobrevivir a los zombies. Para ello puede moverse por todo el mapa mientras dispara a los zombies para acabar con ellos, a cada ronda que pasa el jugador sube de nivel pudiendo mejorar una de las siguientes stats haciendose más fuerte:

- Vida
- Daño
- Velocidad
- Rango de disparo
- Cooldown de disparo
- Cantidad de perdigones por disparo

5.5.2. Zombies

Los zombies son los enemigos del jugador, su objetivo es atacar el jugador, y finalmente acabar con él, así terminando el juego.

Las posibles stats de los zombies son las siguientes:

- Vida: Vida del zombie.
- Daño: Daño del ataque del zombie.
- Velocidad: Velocidad de movimiento del zombie.
- Tamaño: Tamaño del zombie.
- Rango de visión: Rango de visión del zombie, si el jugador no se encuentra en este rango, el zombie se mueve de manera aleatoria.
- Cooldown de ataque: Tiempo que tarda en volver a poder atacar.
- Precisión de ataque: Precisión que tiene el zombie al atacar.
- Velocidad de ataque: Velocidad con la que el zombie ejecuta la acción de atacar.

Estas stats se generan cuando se crea el zombie por una red neuronal, y como entrada tiene la información de las stats del jugador. A medida que avanza el juego, los zombies se van haciendo más fuertes, no solo adaptandose a las stats del jugador, sino que por cada ronda los zombies ganan 5 puntos extra para gastar en stats.

5.6. Controles

Los controles son sencillos y se ajustan al estándar de un juego de ordenador de hoy en día; el jugador puede utilizar las teclas w, a, s y d para moverse hacia adelante, izquierda, derecha y atrás respectivamente, mientras que apunta con el ratón, el clic izquierdo del ratón se utiliza para disparar en la dirección en la que se apunta. Adicionalmente, si se presiona el shift izquierdo mientras se anda, el jugador en vez de andar, corre.

5.7. Implementación del Videojuego

Para el desarrollo del proyecto se ha optado por la utilización de recursos externos gratuitos principalmente sacados de la Unity Asset Store y Mixamo para la parte estética, es decir; modelos 3D, animaciones y sonidos. El resto se ha implementado con ayuda del software de creación de videojuegos Unity3D.

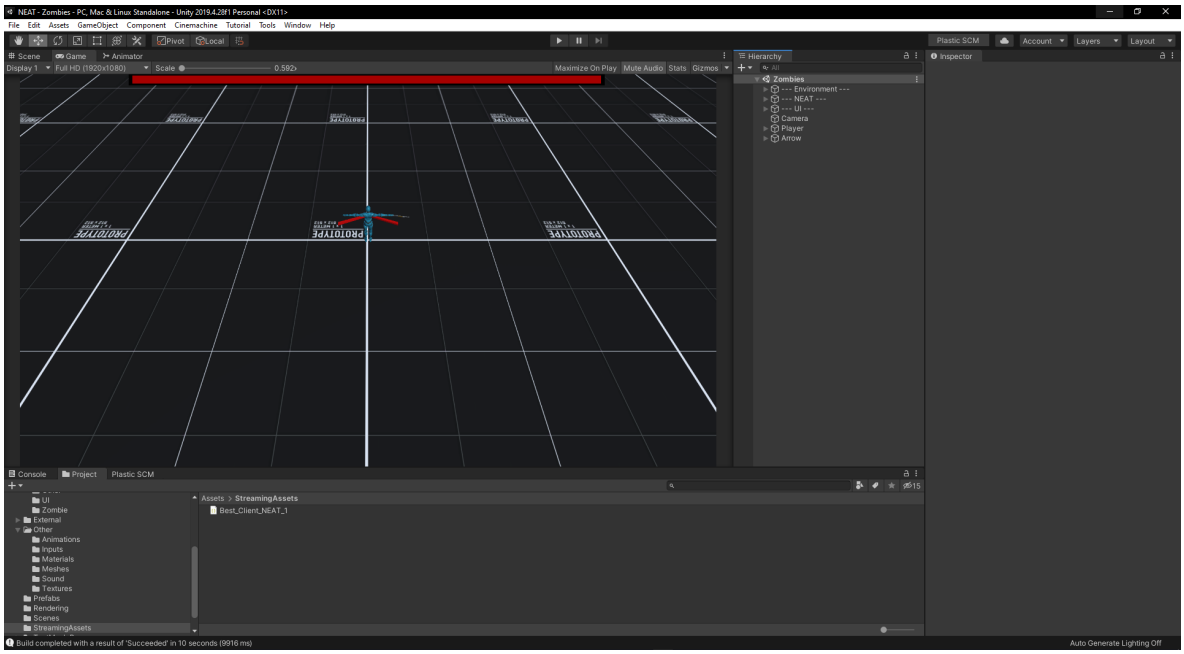

Figura 4: Captura del videojuego desarrollado en Unity.

5.7.1. Estructura

En este apartado se van a describir las clases principales correspondientes al videojuego explicando su objetivo y su relación entre ellas.

Neat Player

La clase player es la que se encarga de todo lo relacionado con el jugador, dado a que solo existe uno y para el prototipo no se planea implementar ningún tipo de multijugador, se ha optado por utilizar el patrón singleton.

Para los controles, se utiliza la libreria InputSystem de UnityEngine para cazar eventos que se llaman cuando el jugador interactua con los controles asignados. Desde este script además se controla cuando se hacen las animaciones y sonidos del jugador.

Neat Zombie

Esta es la clase que representa a los adversarios del jugador, se encarga de establecer las stats del mismo a partir de la red neuronal generada por NEAT.

Utiliza las stats generadas y las utiliza para el control del zombie. El algoritmo que sigue este control es muy sencillo, si tiene al jugador en rango de visión, le sigue, si puede atacarle, le ataca y sino, camina de forma aleatoria.

Neat Tester

Esta es la clase que sirve de conexión entre NEAT y el videojuego, se encarga de definir las funciones necesarias para el funcionamiento de NEAT.

ZombieNeatManager

Clase hija de Neat Tester, implementa las funciones abstractas del Neat Tester para aplicar el testeo de la red neuronal a el caso concreto.

6. NEAT

En esta sección se explica NEAT, el algoritmo seleccionado y algún detalle relacionado a su implementación.

6.1. ¿Qué es NEAT?

NEAT, o *NeuroEvolution of Augmenting Topologies*, es un método neuroevolutivo que supera los mejores métodos similares de topología fija. Esto es gracias a la implementación de un mejor método de cruce entre individuos con diferente topología, una protección de la innovación estructural usando especiación y finalmente aplicando el crecimiento a partir de la estructura mínima. [19]

6.1.1. El Cruce en NEAT

El método de cruce que utiliza NEAT fué desarrollado con el fin de evitar el problema conocido como *Competing Conventions*, el cual consiste en que en el contexto de la neuroevolución, cuando dos individuos representan una solución al problema con soluciones diferentes, el cruce entre estos individuos puede concluir en una pérdida de información. [20]

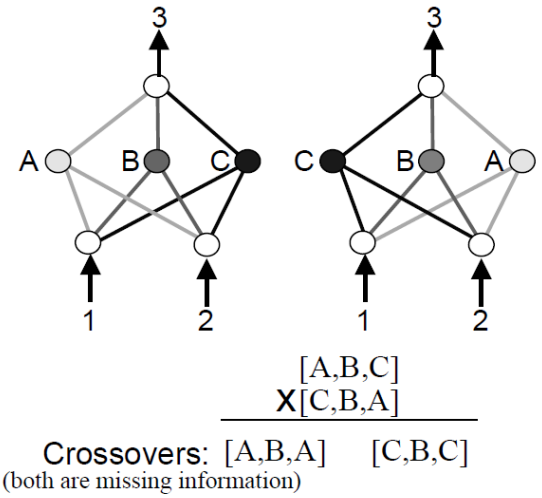

Figura 5: Competing Conventions. [20]

Para evitar este problema, NEAT realiza un seguimiento de mutaciones asignandole a cada mutación un identificador llamado número de innovación. De esta manera, a la hora de realizar el cruce entre dos individuos, se puede utilizar este índice para alinear los genes y evitar la pérdida de información.[21]

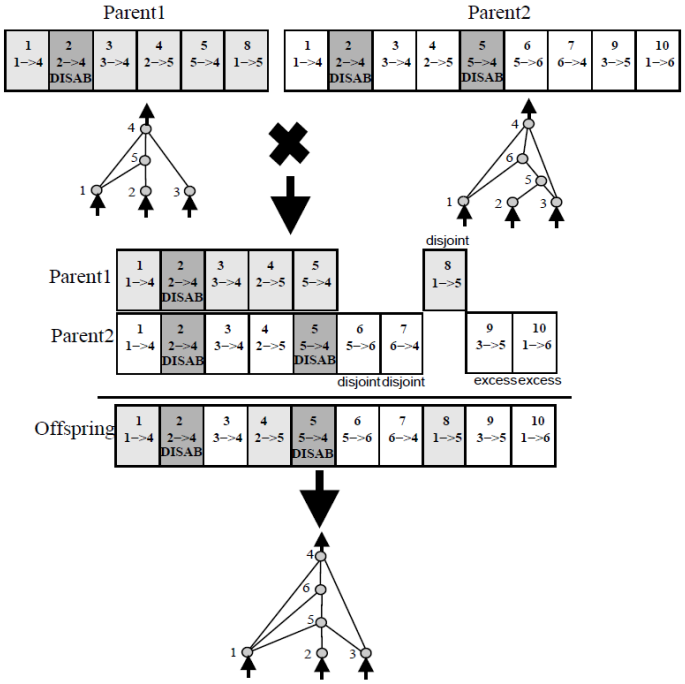

Figura 6: Crossover en NEAT. [21]

6.1.2. El uso de la especiación en NEAT

Otro problema a resolver es que cuando un individuo muta y da a la creación de una nueva topología, el fitness de este individuo tiende a disminuir debido a que se puede añadir no linearidad y los pesos de las conexiones tienen que optimizarse, haciendo así que no sea capaz de sobrevivir y se *extingue* antes de poder optimizarse. Por eso se utiliza un sistema de especiación, mediante el cual se divide a la población en *especies* o *nichos* utilizando una función de compatibilidad que utiliza como información los indices de innovación previamente descritos para dividir correctamente los individuos. Estos agrupamientos de especies comparten fitness y por eso están protegidos y pueden optimizarse y explorarse antes de que se extingan. [22]

6.1.3. Crecimiento a partir de la estructura mínima

Finalmente, la última manera en la que NEAT mejora su eficacia es mediante el desarrollo de las soluciones partiendo de una topología mínima, esto se hace debido a que en otros TWEANNs, que se comienza con topologías aleatorias, suelen surgir varios problemas; por ejemplo, puede ocurrir que no exista un camino entre las neuronas de input y output, formando así una red no funcional que tardará en extinguirse. Sin embargo, el motivo principal por el cual es mejor comenzar con una topología mínima es porque de esta manera también se minimiza el número de parámetros a buscar, ya que con topologías aleatorias existen gran cantidad de parámetros innecesarios que hay que evaluar y explorar en vano. [23] [24]

6.1.4. Codificación de genomas en NEAT

NEAT utiliza una codificación de genomas diseñada para que se pueda de forma sencilla alinear los genes a la hora de realizar el crossover entre dos genomas. De esta forma, cada genoma se compone de una lista de conexiones donde cada conexión contiene las dos neuronas que conecta especificando cual es la de entrada y cual la de salida, el peso. [25]

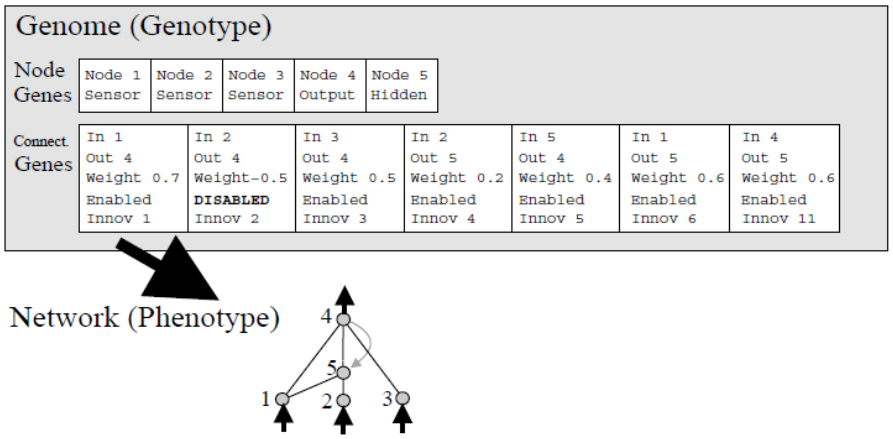

Figura 7: Genoma en NEAT. [26]

6.1.5. Mutación en NEAT

La mutación en NEAT, como indica su nombre, puede modificar tanto los pesos como la topología de la red neuronal. Los pesos de las conexiones mutan igual que otros métodos de neuroevolución, sin embargo, las mutaciones estructurales pueden ocurrir de dos maneras, añadiendo una nueva conexión entre dos nodos, donde se escogen dos nodos no conectados y se conectan con un peso aleatorio, y añadiendo un nuevo nodo, donde se escoge una conexión existente entre dos nodos y se divide en dos, para que el funcionamiento de la red se afecte en lo mínimo, de las dos nuevas conexiones la que viene del anterior nodo input se le asigna un peso de 1 y a la segunda conexión se le asigna el peso de la conexión inicial. [25]

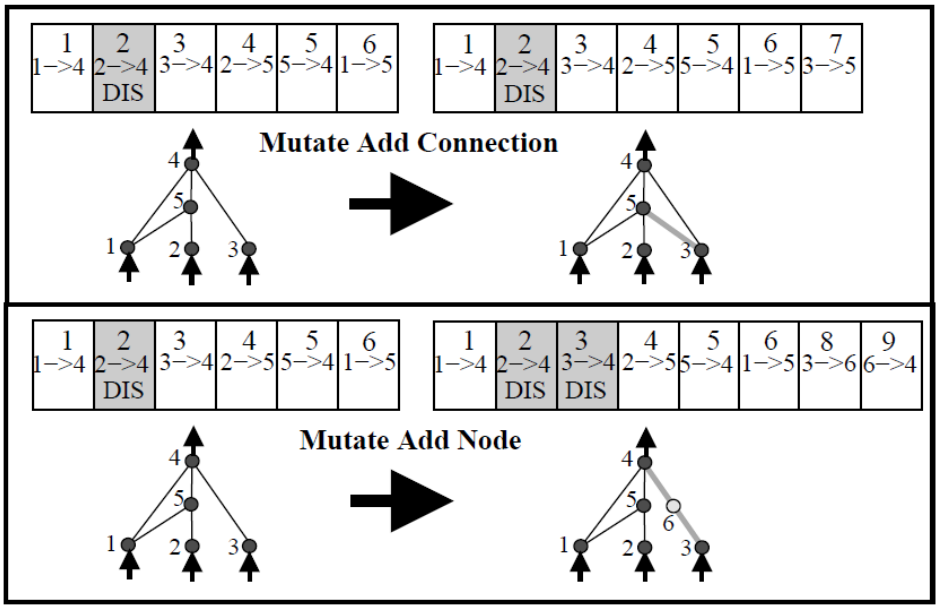

Figura 8: Mutación en NEAT. [27]

6.2. Implementación

Como se ha mencionado previamente, se ha llevado a cabo una implementación propia del algoritmo, por ello, en esta sección se va a mostrar un diagrama de la estructura seguida en la implementación y se van a explicar las partes más relevantes.

6.2.1. Diagrama de Clases

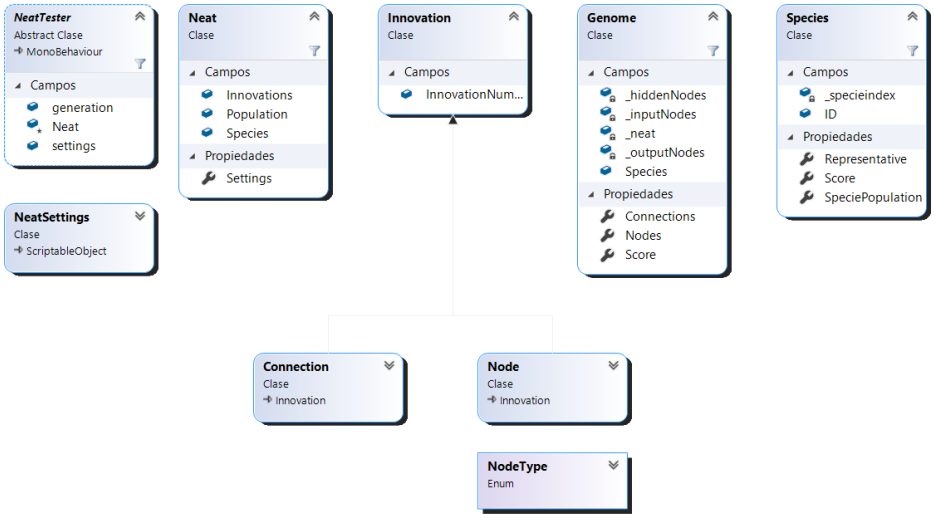

Figura 9: Diagrama de clases de NEAT

Para realizar esta implementación, se ha utilizado como base la publicación original del algoritmo, y se han investigado varias implementaciones ya existentes del algoritmo, como la página de usuarios de NEAT [28], una implementación en Java de Finn Eggers en github [29], o el libro AI Techniques for Game Programming [30].

6.2.2. Genoma

La clase Genoma contiene la codificación de cada solución, en este caso ha sido implementada tal y como la describen en el algoritmo original, una red completamente conexa con una capa de entrada y otra de salida y pesos aleatorios. En este caso, hay 7 nodos de entrada y 8 de salida.

Los 7 nodos de entrada corresponden con:

- Bias: el bias es un nodo que siempre está activado cuyo objetivo es proporcionar un grado de libertad al modelo.
- Vida maxima del jugado: Como el nombre indica, es la vida máxima del jugador, comienza a con un valor de 100 y tiene un máximo de 500.
- Velocidad del jugador: Es la velocidad a la que se mueve el jugador, comienza con un valor de 3 unidades/seg y tiene un máximo de 10 unidades/seg.

- Daño del jugador: Es el daño por bala del jugador, comienza siendo 1 y tiene un máximo de 20.
- Rango de disparo del jugador: La distancia máxima que recorren las balas disparadas por el jugador, comienzan siendo 15 unidades y tienen un máximo de 50.
- Cooldown de disparo del jugador: El tiempo que tiene que esperar el jugador entre disparos, comienza siendo 1000 milisegundos y tiene un mínimo de 500.
- Balas por disparo del jugador: El número de balas que dispara el jugador, son 40 al comienzo y pueden ser un máximo de 80.

Los 8 nodos de salida controlan:

- La vida máxima del zombie: Puede ser cualquier valor entre 1 y 20, decidido por el modelo.
- La velocidad del zombie: Similar a la vida máxima, puede ser cualquier valor entre 1 y 50.
- La fuerza/daño del zombie: También puede ser cualquier valor entre 1 y 20.
- El tamaño del zombie: determina el tamaño del zombie, este valor está entre 0,75 y 2,75.
- El rango de visión del zombie: determina el rango en el que el zombie detecta al jugador y le persigue, este valor está entre 5 y 75.
- El Cooldown de ataque del zombie: determina el tiempo que tiene que esperar el zombie entre ataque y ataque, puede ser cualquier valor entre 0,5 segundos y 2 segundos.
- La precisión de ataque del zombie: determina la probabilidad de que el ataque del zombie acierte, puede ser cualquier valor entre 0,3 y 1.
- La velocidad de ataque del zombie: determina la velocidad en la que el zombie realiza el ataque, este valor está entre 1, que sería la velocidad por defecto y 3 que significa que se realiza el triple de rápido.

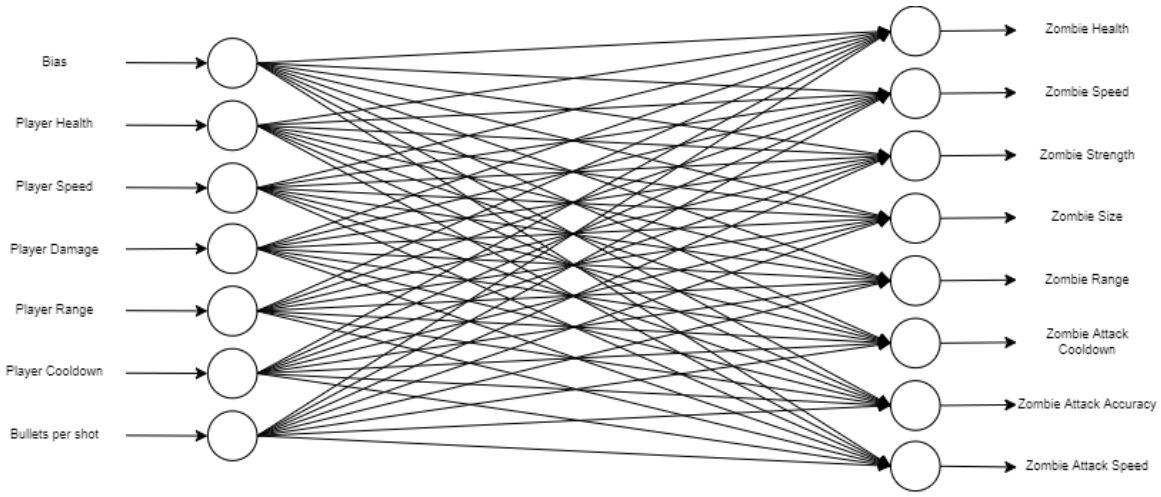

Figura 10: Genoma Inicial

6.2.3. Evaluación de soluciones

La función de evaluación ha ido cambiando a lo largo del desarrollo del proyecto. Se llegó a la conclusión de que la manera de evaluarlos sería recompensándolos por el daño que le hacen al jugador, y en caso de morir, depende del estado en el que mueren y la distancia al jugador de la que se encontraban. Así que la función de fitness sería:

Dado un individuo a evaluar i.

Dada una generación g.

Dado el número de ataques dados en la generación hasta acabar con el jugador $H_{i,g}$

Dado el número de ataques fallidos en la generación $F_{i,g}$

Dada la fuerza del individuo S_i

Dado el multiplicador de la fuerza M_1

Dado un valor representativo de la diferencia de cambio entre las stats del jugador y las del zombie entre la ronda previa y la actual de ambos $V_{i,g,r}$.

Dado un multiplicador del valor previo M_2.

$$f(i,g) = H_{i,g} * M_1 * S_i + F_{i,g} + M_2 * V_{i,g,r}$$

La razón por la cual se utiliza el valor $V_{i,g,r}$ es para intentar motivar la adaptación de la red a las estadísticas del jugador, y evitar que el modelo optimize una mejor manera de jugar independientemente de las estadísticas del jugador.

7. Experimentación y Resultados

Para el entrenamiento del modelo, se ha concluido que dada la dependencia del fitness a factores aleatorios lo mejor es que por cada generación se ejecuten varias rondas y así conseguir un fitness que represente mejor la solución. Por eso se ha decidido hacer 6 rondas por generación con una población de 200. También se ha desarrollado un programa que controle el papel de jugador, de esta forma se puede acelerar el proceso de entrenamiento. Los datos de entrada son normalizados y estos son los parámetros aplicados al algoritmo:

- Bias: 1
- Probabilidad de mutación de enlace: 0.01
- Probabilidad de mutación de nodo: 0.03
- Probabilidad de mutación de variar peso de enlace: 0.04
- Probabilidad de mutación de randomizar peso de enlace: 0.02
- Probabilida de activar o desactivar enlace: 0.0001
- C1: 1
- C2: 1
- C3: 1

Inicialmente se ha entrenado el modelo durante 1000 generaciones con las stats del jugador randomizadas cada generación.

Estos son los resultados del entrenamiento:

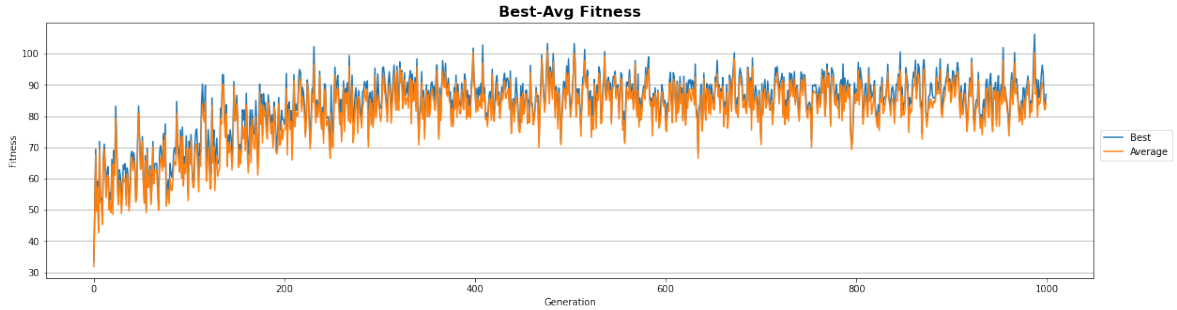

Figura 11: Mejor fitness y fitness medio de la población.

Se puede observar el mejor individuo en cada generación (azul) y la media de la población (naranja).

Como se observa, tras el entrenamiento, se ha logrado alcanzar un nivel de fitness que oscila entre alrededor de 80 y 90. A pesar de que claramente hay un visible aumento del fitness, se aprecia una notable variabilidad en el rendimiento de una generación a otra.

Esto se debe a que, en cada generación, las estadísticas del jugador son generadas de forma aleatoria, lo que puede dar lugar a variaciones considerables en la capacidad del jugador, dificultando así el proceso de aprendizaje.

Con el fin de entrenar un modelo que sea capaz de adaptarse a las stats del jugador, se ha realizado un segundo entrenamiento con una estrategia diferente. En vez de cambiar de manera aleatoria las stats del jugador, cada generación se va a componer de 6 rondas donde en cada ronda va a estar maximizada una de las stats del jugador, de esta manera el fitness del zombie representará lo bien que lo ha hecho en esas 6 rondas y asi beneficiando a los que mejor se adapten.

Estos son los resultados del entrenamiento:

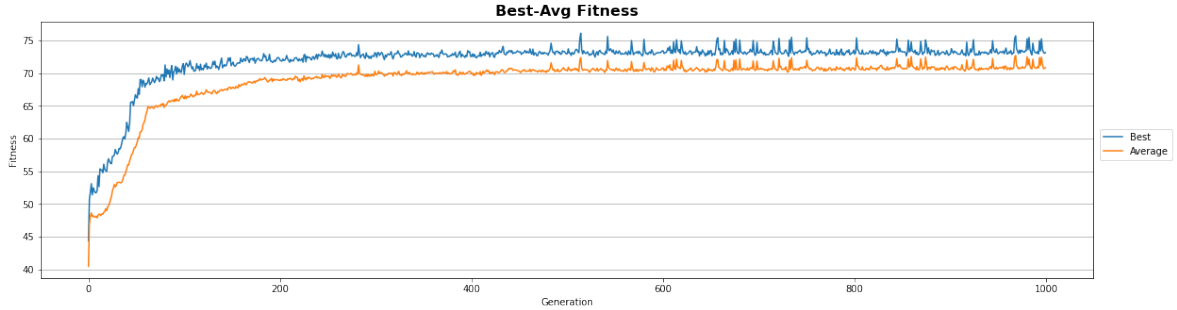

Figura 12: Mejor fitness y fitness medio de la población.

Al usar esta nueva estrategia, se ha reducido la variedad que teníamos antes, pero el fitness que hemos logrado no es tan alto como el que obteníamos con la estrategia anterior. Debido a esto se va a utilizar el primer modelo para la siguiente fase.

Ahora, para comprobar la validez del modelo para el objetivo de adaptarse a las estadísticas del jugador, se ha realizado un experimento que consiste en probar el modelo entrenado incrementando cada estadística del jugador por separado y observando el comportamiento del modelo.

Estos son los resultados del experimento:

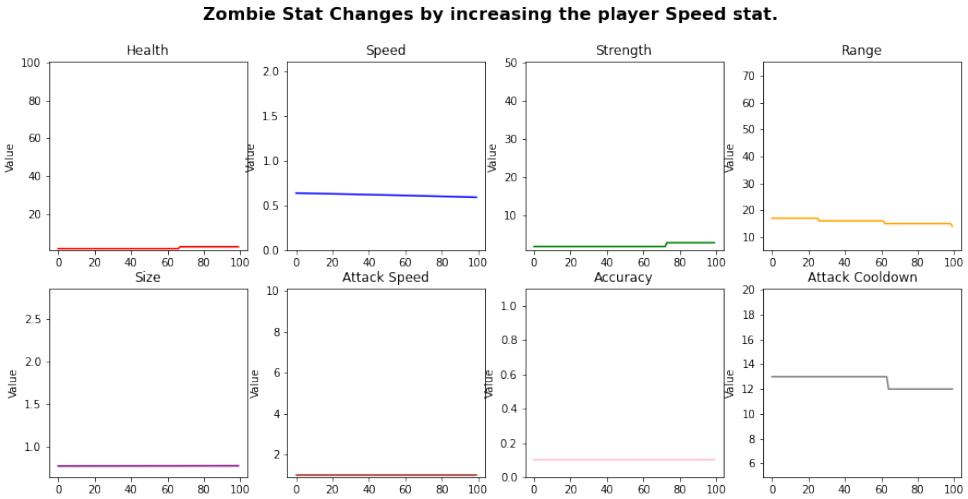

Figura 13: Cambio en las estadísticas de los Zombies al incrementar la vida del jugador

Se observa que, al aumentar gradualmente la velocidad del jugador, el modelo entrenado reduce la velocidad, el rango y el tiempo de enfriamiento de ataque, a cambio de aumentar ligeramente su vida y su fuerza.

Lamentablemente, este comportamiento solo se ha observado en este caso, mientras que en el resto, la mayoría sino todas las características parecen permanecer estáticas. Debido a esto, las gráficas que representan estos datos se muestran en el Anexo 1.

Observando estos resultados, se llega a la conclusión de que el modelo no ha logrado adaptarse de manera adecuada a las estadísticas del jugador, y si lo ha hecho, ha sido de manera no significativa. Esto se debe a que el modelo ha encontrado una aparente solución óptima para el problema planteado, lo que limita su capacidad de adaptación.

Para abordar este problema en el futuro, se podría intentar llevar a cabo una experimentación más exhaustiva en el entrenamiento del modelo, con el fin de identificar los parámetros óptimos y la estrategia más adecuada para el entrenamiento. Sin embargo, es posible que un mejor diseño del videojuego o de la función de fitness resulte más efectivo, ya que de esta manera se podría recompensar la adaptabilidad del modelo de una manera más efectiva.

8. Conclusiones y trabajos futuros

Este proyecto ha logrado alcanzar dos de los tres objetivos principales que se plantearon: en primer lugar, se ha diseñado un prototipo de videojuego; y en segundo lugar, se ha implementado NEAT en Unity3D y en el prototipo de videojuego desarrollado. A pesar de ello, no se ha logrado de forma satisfactoria el tercer objetivo de entrenar un modelo que se adapte al jugador.

No obstante, con una investigación más profunda de los factores que influyen en el rendimiento del algoritmo, un diseño mejorado del juego y de la función de fitness, y un mayor tiempo dedicado al entrenamiento del modelo, se pueden obtener resultados más significativos e interesantes.

Otro problema que se puede mejorar sería la eficiencia del algoritmo, es cierto que existen implementaciones que son seguramente más eficiente que la implementada en este trabajo por varios factores, entre ellos el lenguaje de programación utilizado. No obstante, la ventaja de implementarlo para Unity 3D es la facilidad de crear entornos de entrenamiento y implementación del modelo en un videojuego. Hay dos maneras que se pueden investigar para mejorar la eficiencia del algoritmo en Unity 3D; por un lado se pueden intentar utilizar *ComputeShaders* para ejecutar el algoritmo en la GPU, o se puede utilizar el sistema DOTS de unity que, aunque está todavia en desarrollo, ha demostrado su eficiencia superando en gran medida la forma tradicional de programación en Unity.

Algunas ideas para avanzar en base a este trabajo sería con el desarrollo de un mejor diseño del concepto, podría ser interesante que la ANN controlase el comportamiento del adversario de una manera más dinámica, eligiendo cuándo atacar, cuándo retirarse, hacia dónde moverse... O, por ejemplo, creando la capacidad de cambiar la morfología de la entidad teniendo una serie de posibles partes con ventajas y desventajas. También sería interesante la aplicación de este algoritmo a otros tipos de videojuegos o la utilización de otros algoritmos neuroevolutivos similares.

Para finalizar, mencionar que a pesar de no haberse superado satisfactoriamente el último objetivo planteado, este proyecto se suma a los ya existentes en la investigación actual sobre la aplicación de técnicas de IA en videojuegos, y también se han adquirido importantes conocimientos sobre la neuroevolución, NEAT, Inteligencia artificial y sus aplicaciones a videojuegos.

Bibliografía

[1] Wikipedia. *Deep Blue versus Garry Kasparov.* URL: https://en.wikipedia.org/wiki/Deep_Blue_versus_Garry_Kasparov.

[2] Wikipedia. *AlphaGo versus Lee Sedol.* URL: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol.

[3] OpenAi. *OpenAi Five defeats dota 2 world champions.* URL: https://openai.com/blog/openai-five-defeats-dota-2-world-champions.

[4] Jason M. Traish y James R. Tulip. "Towards adaptive online RTS AI with NEAT". En: *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. 2012, págs. 430-437. DOI: 10.1109/CIG.2012.6374187.

[5] Matheus G. Cordeiro y col. "A Minimal Training Strategy to Play Flappy Bird Indefinitely with NEAT". En: *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. 2019, págs. 21-28. DOI: 10.1109/SBGames.2019.00014.

[6] Daniel Jallov, Sebastian Risi y Julian Togelius. "EvoCommander: A Novel Game Based on Evolving and Switching Between Artificial Brains". En: *IEEE Transactions on Computational Intelligence and AI in Games* 9.2 (2017), págs. 181-191. DOI: 10.1109/TCIAIG.2016.2535416.

[7] Teofebano Kristo y Nur Ulfa Maulidevi. "Deduction of fighting game countermeasures using Neuroevolution of Augmenting Topologies". En: *2016 International Conference on Data and Software Engineering (ICoDSE)*. 2016, págs. 1-6. DOI: 10.1109/ICODSE.2016.7936127.

[8] Dario Blasco Millan, Enrique Romero Merino y René Alquézar Mancho. "Estudio del algoritmo NEAT aplicado al videojuego Pac-Man". Tesis doct. 2019.

[9] Christopher M. Bishop. "Neural Networks". En: *Pattern Recognition and Machine Learning*. Springer Verlag New York, 2006.

[10] Peter Norvig Stuart J. Russell. "Learning from examples". En: *Artificial intelligence : a modern approach*. Pearson Education, 2014.

[11] Glosser.ca. *Own work, Derivative of File: Artificial neural network.svg, CC BY-SA 3.0.* URL: https://commons.wikimedia.org/w/index.php?curid=24913461.

[12] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. MIT Press, 2016.

[13] Peter Norvig Stuart J. Russell. "Beyond Classical Search". En: *Artificial intelligence : a modern approach*. Pearson Education, 2014. Cap. 4.1.

[14] Scholarpedia. *Neuroevolution.* URL: http://scholarpedia.org/article/Neuroevolution.

[15] Sebastian Risi y Julian Togelius. "Neuroevolution in Games: State of the Art and Open Challenges". En: *IEEE Transactions on Computational Intelligence and AI in Games* 9.1 (2017), págs. 25-41. DOI: 10.1109/TCIAIG.2015.2494596.

[16] IAT. *INTELIGENCIA ARTIFICIAL EN VIDEOJUEGOS: UNA MIRADA AL PASADO Y FUTURO DE LA INDUSTRIA*. URL: https://iat.es/tecnologias/inteligencia-artificial/videojuegos/.

[17] Wikipedia. *Videojuego de Supervivencia*. URL: https://es.wikipedia.org/wiki/Videojuego_de_supervivencia.

[18] Wikipedia. *Third-person shooter*. URL: https://en.wikipedia.org/wiki/Third-person_shooter.

[19] Kenneth O Stanley y Risto Miikkulainen. "Abstract." En: *Evolutionary computation*. MIT Press, 2002. Cap. Evolving neural networks through augmenting topologies, pág. 99.

[20] Kenneth O Stanley y Risto Miikkulainen. "Competing Conventions." En: *Evolutionary computation*. MIT Press, 2002. Cap. Evolving neural networks through augmenting topologies, pág. 103.

[21] Kenneth O Stanley y Risto Miikkulainen. "Tracking Genes through Historical Markings." En: *Evolutionary computation*. MIT Press, 2002. Cap. Evolving neural networks through augmenting topologies, págs. 108-109.

[22] Kenneth O Stanley y Risto Miikkulainen. "Tracking Genes through Historical Markings." En: *Evolutionary computation*. MIT Press, 2002. Cap. Evolving neural networks through augmenting topologies, págs. 104-105.

[23] Kenneth O Stanley y Risto Miikkulainen. "Initial Populations and Topological Innovation." En: *Evolutionary computation*. MIT Press, 2002. Cap. Evolving neural networks through augmenting topologies, págs. 105-106.

[24] Kenneth O Stanley y Risto Miikkulainen. "Minimizing Dimensionality through Incremental Growth from Minimal Structure." En: *Evolutionary computation*. MIT Press, 2002. Cap. Evolving neural networks through augmenting topologies, pág. 111.

[25] Kenneth O Stanley y Risto Miikkulainen. "Genetic Encoding." En: *Evolutionary computation*. MIT Press, 2002. Cap. Evolving neural networks through augmenting topologies, págs. 107-108.

[26] Kenneth O Stanley y Risto Miikkulainen. "Figure 2." En: *Evolutionary computation*. MIT Press, 2002. Cap. Evolving neural networks through augmenting topologies, pág. 106.

[27] Kenneth O Stanley y Risto Miikkulainen. "Figure 3." En: *Evolutionary computation*. MIT Press, 2002. Cap. Evolving neural networks through augmenting topologies, pág. 107.

[28] Kenneth O. Stanley. *The NeuroEvolution of Augmenting Topologies (NEAT) Users Page*. URL: https://www.cs.ucf.edu/~kstanley/neat.html.

[29] Finn Eggers. *NEAT*. https://github.com/Luecx/NEAT. 04/2022. 2019.

[30] Mat Buckland y Mark Collins. *AI Techniques for Game Programming*. Premier Press, 2002. ISBN: 193184108X.

Anexo 1: Gráficas de los resultados de la experimentación

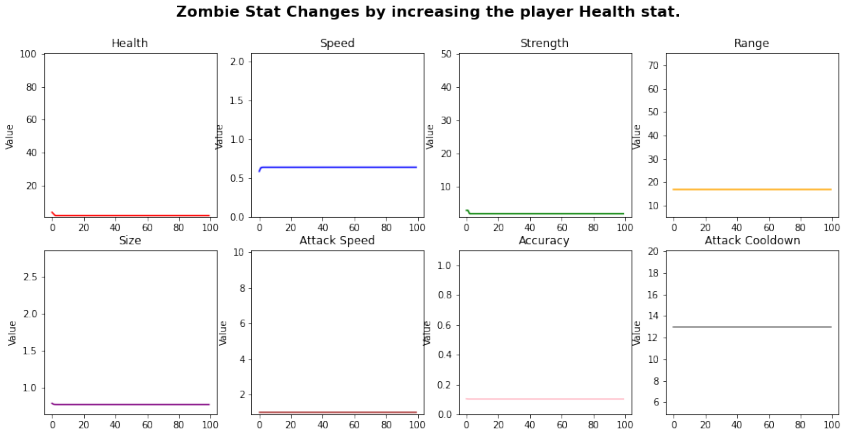

Figura 14: Cambio en las estadísticas de los Zombies al incrementar la vida del jugador

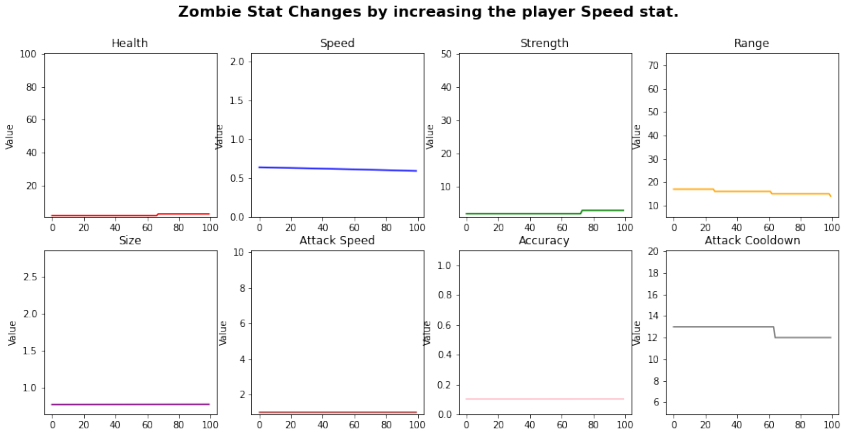

Figura 15: Cambio en las estadísticas de los Zombies al incrementar la velocidad del jugador

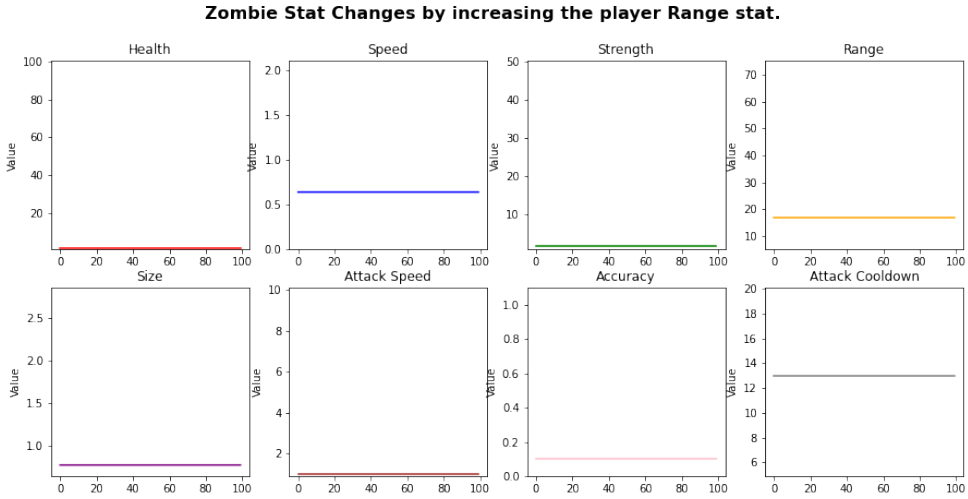

Figura 16: Cambio en las estadísticas de los Zombies al incrementar el rango del jugador

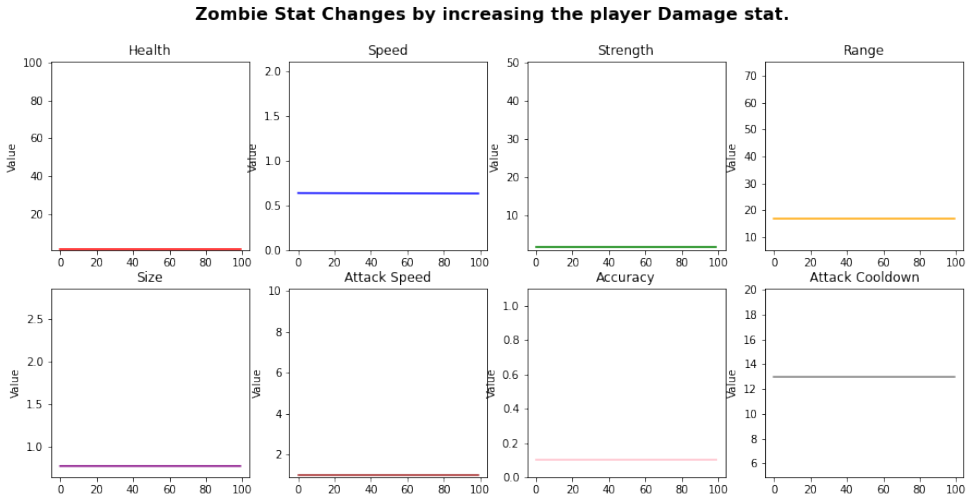

Figura 17: Cambio en las estadísticas de los Zombies al incrementar el daño del jugador

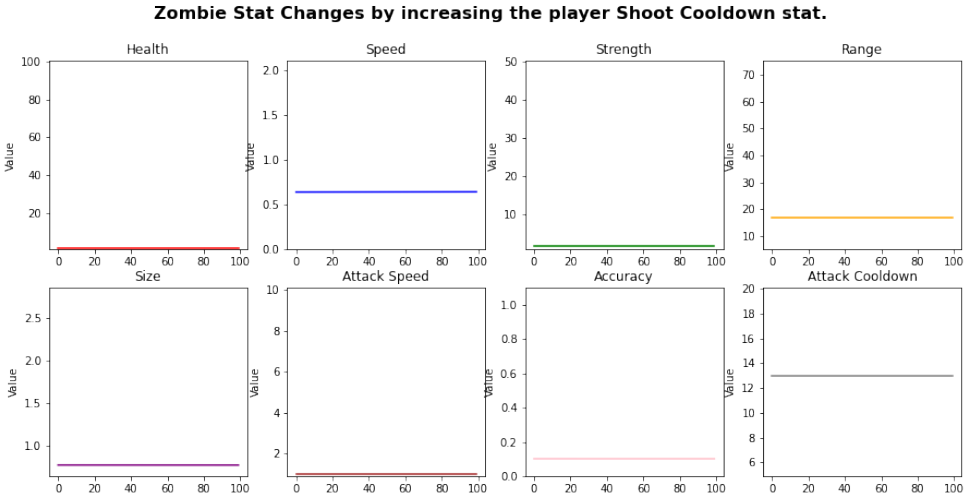

Figura 18: Cambio en las estadísticas de los Zombies al incrementar la cadencia de disparo del jugador

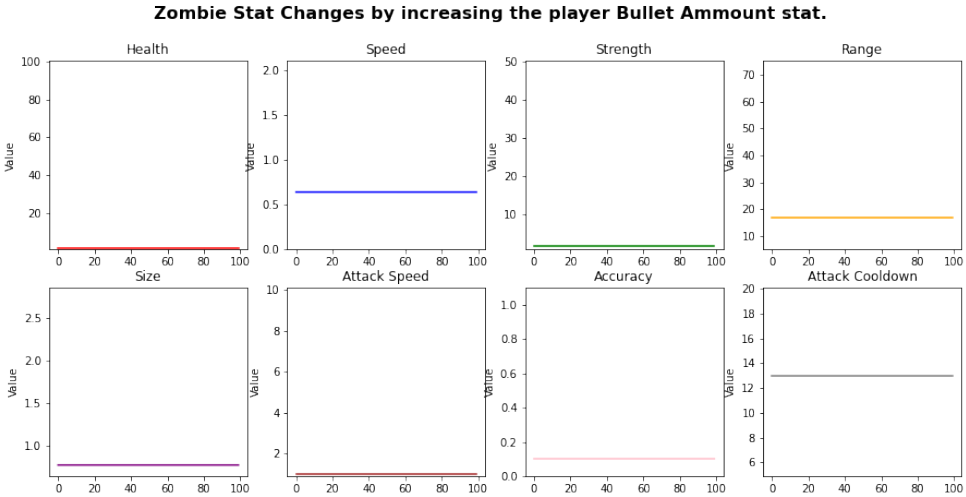

Figura 19: Cambio en las estadísticas de los Zombies al incrementar la cantidad de balas del jugador