UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Creación de un videojuego y un bot inteligente para el mismo

Presentado por Pablo Alejos Salamanca
en Universidad de Burgos — 28 de junio de 2017

Tutores: Dr. José Francisco Díez Pastor
Dr. César Ignacio García Osorio

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. José Francisco Díez Pastor y D. César Ignacio García Osorio, profesores del departamento de Ingeniería Civil, área de Lenguajes y sistemas informáticos.

Exponen:

Que el alumno D. Pablo Alejos Salamanca, con DNI 71302468, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado «Creación de un videojuego y un bot inteligente para el mismo».

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de los que suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 28 de junio de 2017

Vº. Bº. del Tutor:

D. José Francisco Díez Pastor

Vº. Bº. del tutor:

D. César Ignacio García Osorio

Resumen

El proyecto tiene como propósito la creación de un videojuego y, posteriormente, el desarrollo de una serie de agentes inteligentes que sean capaces de jugar al mismo. Mediante el análisis y explotación de técnicas de aprendizaje máquina y la utilización de la minería de datos, seremos capaces de dotar al agente de la capacidad de descubrir las reglas del juego a partir de un conjunto de datos y deberá ser capaz de jugar por imitación a partir de lo aprendido. Para el desarrollo del videojuego se utiliza Unity3D, un entorno de desarrollo de Videojuegos. Los agentes inteligentes se implementan en Python, un lenguaje de programación que, a día de hoy, está entre los tres más usados mundialmente.

Descriptores

Minería de datos, Videojuego, Aprendizaje máquina, Unity3D, Imitación, Python, Agente inteligente

Abstract

The purpose of this project is the creation of a video game, and then follows the development of a series of intelligent agents that are able to interact with the same. The agents will be able to discover the rules of the games by using Machine Learning and Data Mining techniques on a dataset obtained from a human player, and hence it will be able to learn by imitation. The development of the game is done with Unity3D, a video game framework. The intelligent agents are implemented in Python, a programming language that, to date, is among the three most used worldwide.

Keywords

Data Mining, VideoGame, Machine Learning, Unity3D, Imitation, Python, Intelligent Agent

Índice general

Índice de figuras

Índice de tablas

Introducción

Históricamente los agentes inteligentes (ya sea un robot físicamente presente, formado por un hardware o simplemente, un software capaz de hacer web scraping o capaz de jugar al ajedrez), necesitan la definición de rutinas para poder realizar la labor para la que han sido creados. Esta labor de introducir las instrucciones en su sistema de control recae sobre el programador. Programar estos agentes abarca desde los más sencillos, que realizan una actividad muy concreta, a complejos sistemas capaces de realizar tareas muy sofisticadas. Estos últimos requieren de un gran esfuerzo por parte del programador, pero, ¿y si los agentes se programasen a sí mismos? Gracias a técnicas como Machine Learning podemos hacer que los agentes aprendan por sí solos a realizar las tareas para las que fueron creados. Este proyecto tiene como objetivo final entrenar un agente inteligente capaz de aprender a jugar a un determinado videojuego mediante imitación del ser humano. Esto requiere un gran volumen de datos y una persona haciendo uso de juego para generar los datos de entrenamiento necesarios.

Objetivos del proyecto

- El objetivo del proyecto es diseñar y desarrollar un videojuego de tipo arcade. Para la implementación del juego que sirve como base para este proyecto, es necesario planear las mecánicas de juego, además de diseñar los modelos del jugador, los enemigos y otros elementos gráficos. Esta tarea también incluye el diseño de la interfaz, banda sonora y efectos de sonido.
- Diseñar e implenentar un sistema inteligente que posteriormente será utilizado para interactuar con el videojuego creado. Este proceso de divide en:
    - Establecer una comunicación entre el juego y el modelo del sistema inteligente. Dado que el juego y el script que hace uso del modelo serán implementados en diferente lenguaje de programación, se debe establecer un mecanismo de intercambio de datos para la comunicación.
    - Diseñar el conjunto de datos que posteriormente servirá para el entrenamiento del modelo. Al ser el punto de partida el aprendizaje por imitación, necesitaremos un gran volumen de datos de los cuales seleccionaremos todos o solo los más representativos para el entrenamiento.
    - Implementar, haciendo uso de las bibliotecas como scikit-learn (minería de datos), DEAP(algoritmos evolutivos) y Pandas y NumPy (procesamiento de datos a bajo nivel) el sistema inteligente que jugará al videojuego.

Conceptos teóricos

3.1. Introducción

Han pasado algunos años desde que se empezaron a utilizar los videojuegos como entorno de prueba para agentes inteligentes, y es que los videojuegos son un excelente campo de prueba para algoritmos de inteligencia artificial. Los videojuegos nos proporcionan un entorno controlado y predecible, lo que nos permite determinar con mayor eficacia la calidad del algoritmo. Para una mejor comprensión del trabajo realizado a continuación se exponen brevemente algunos conceptos relevantes.

3.2. Minería de datos

Vivimos rodeados de datos, cada vez que utilizamos nuestro smartphone para consultar el tiempo, realizamos un pago con tarjeta, tuiteamos... dejamos un rastro de datos. Esto es aplicable, no solo al ámbito privado, sino que es aplicable a cualquier ámbito de la industria, ya que allá donde exista un proceso, van a existir datos. Siempre se ha intuido que esa información podría resultar interesante, pero no ha sido hasta hace relativamente poco, gracias a los grandes avances en la computación, cuando hemos empezado a sacarle provecho, y sí, todo ese reguero de datos que dejamos ha resultado realmente valioso.

La minería de datos consiste en aplicar a esa gran cantidad de datos desorganizados de los que disponemos, una serie de métodos matemáticos que nos permitan transformarlos en información. Cuando hablamos de transformar datos en información, hablamos que que la minería de datos nos va a permitir extraer patrones que una persona a simple vista no sería capaz de descubrir.

3.3. Inteligencia Artificial

La inteligencia artificial (IA) es una ciencia en auge en las últimas décadas gracias al vertiginoso crecimiento del rendimiento en el campo de la computación. En realidad no hay una única definición para IA, ya que históricamente se le han dado diferentes enfoques,siendo aplicada a diferentes campos con diversos propósitos. Russel y Norvig [9] sintetizan en su libro las definiciones de diversos autores con cuatro enfoques diferentes.

Sistemas que piensan como humanos	Sistemas que piensan racionalmente
«[La automatización de] actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de definiciones, resolución de problemas, aprendizaje...(Bellman, 1978)	«El estudio de las facultades mentales mediante el uso de modelos computacionales». (Charniak y McDermott, 1985) «El estudio de los cáculos que hacen posible percibir, razonar y actuar!.(Winston, 1992)
Sistemas que actúan como humanos	Sistemas que actúan racionalmente
«El arte de desarrollar máquinas con capacidad para realizar funciones que cuando son realizadas por personas requieren inteligencia». (Kurzweil, 1990) «El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor». (Rich y Knight, 1991)	»La Inteligencia computacional es el estudio del diseño de agentes inteligentes».(Poole et al.1998) «IA... Está relacionada con conductas inteligentes en artefacto". (Nilson, 1998)

Tabla 3.1: Definiciones de inteligencia artificial

Para el caso que nos ocupa la que más me gusta es «El estudio de cómo lograr que los computadores realicen tareas que, por el momento, los humanos hacen mejor», ya que mediante las técnicas de aprendizaje que emplearé, nos debería costar diferenciar si es un humano o una máquina quien realiza las tareas.

Sistemas inteligentes

En 1995 comienzan a aparecer los sistemas inteligentes [10]. Un agente inteligente es una entidad capaz de recibir estímulos de su entorno, procesar esta información y actuar en consecuencia de manera racional, maximizando la eficacia del resultado esperado.

3.4. Aprendizaje Máquina

El concepto de aprendizaje máquina, o Machine Learning, se basa en la idea de utilizar una serie de estímulos o percepciones, no sólo para la toma de decisiones, sino para mejorar la respuesta y que estas decisiones sean más certeras. Hay múltiples formas de aprendizaje que podemos clasificar en tres grupos mas amplios: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

Aprendizaje supervisado

El aprendizaje supervisado se basa en la idea de aprender a partir de ejemplos o instancias, es decir, se le proporcionarán una serie de entradas (atributos), y sus correspondientes salidas esperadas.

Entrada	Salida
1	2
2	4
3	6
4	8
5	10

Tabla 3.2: Ejemplo de instancias de aprendizaje supervisado

En el ejemplo podemos ver un set de datos muy elemental que nos permitirá entender el aprendizaje supervisado. La entradas, o la entrada, es un único valor numérico y la salida otro valor numérico. La función que permite obtener la variable dependiente y a partir de la independiente x a simple vista se deduce que es: $y = 2x$. Podemos entrenar un modelo para generalizar cualquier valor mediante una sencilla regresión lineal.

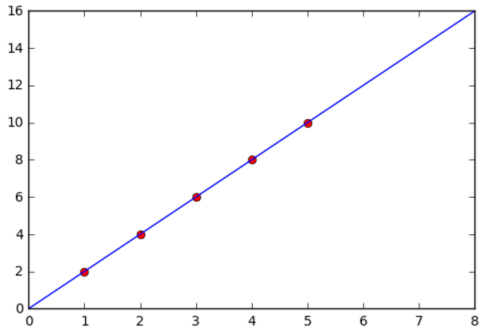

Figura 3.1: Representación de la regresión lineal

La regresión lineal permite ajustar una función a los valores proporcionados, es decir, gráficamente se debe poder trazar una línea que se encuentre a la menor distancia posible de todos los puntos. En la figura 3.1 se pueden ver en rojo los puntos que representan los valores en entrada en el eje x, y los valores de salida en el eje y, y en color azul la representación de la función, en este caso, $y = 2x$.

Llegados a este punto podríamos decir que ya tenemos un modelo «entrenado», puesto que si queremos predecir la salida que obtendríamos en, por ejemplo, $x = 6$, el modelo nos devolvería el valor 12.

En este ejemplo se ha utilizado una regresión lineal. Este modelo en ciertos casos se queda corto, pero hay otros modelos de aprendizaje supervisado muy conocidos que nos permitirán hacer mejores estimaciones. En la realización de este proyecto se ha empleado una combinación de varios de estos y otros modelos, a mencionar, Decision tree (arboles de decisión), Random Forest(una variante de los arboles de decisión), MLP (perceptrón multicapa) y una red neuronal convolucional.

Aprendizaje no supervisado

En contraposición al aprendizaje supervisado, tenemos el aprendizaje no supervisado. En el aprendizaje no supervisado no vamos a proporcionar una salida determinada, simplemente se le proporcionan unas entradas y se espera que consiga generalizar exclusivamente a partir de los datos. La idea básica de este tipo de aprendizaje es buscar patrones en las agrupaciones de datos. Normalmente es un tipo de entrenamiento que lleva mucho más tiempo. No se profundiza más en este caso, ya que no es relevante en este proyecto.

Aprendizaje por refuerzo

En este tercer tipo de aprendizaje no seremos nosotros los que le indiquemos cuál es la respuesta correcta. En este tipo de aprendizaje es el propio agente el que va a aprender a partir de una serie de refuerzos o recompensas. En este caso se toma una decisión y se observa la respuesta, si esta es favorable, se ajustará la función.

En el caso de mi proyecto, se podría hablar de una aproximación a este tipo de aprendizaje, sin tratarse exactamente del mismo. En el caso de mi algoritmo de entrenamiento, se utiliza una red neuronal cuya función de *fitness* es la puntuación obtenida. Se asemeja al aprendizaje por refuerzo en el aspecto de que para mejorar la velocidad de aprendizaje, e independientemente de la puntuación del jugador, se aplicarán de forma arbitraria recompensas (para un refuerzo positivo) y penalizaciones (para un refuerzo negativo). En mi caso, el refuerzo positivo es sumar a la puntuación final el tiempo que ha estado jugando sin morir. En el caso del refuerzo negativo, aplicaremos una penalización al morir.

3.5. Árboles de decisión

Los árboles de decisión son un tipo de clasificador basados en instancias, es decir, estaríamos hablando de aprendizaje supervisado. La instancias las conforman una serie de atributos de entrada y unas clases, salidas esperadas. Dadas estas instancias, el clasificador genera una estructura de árbol en base a éstas.

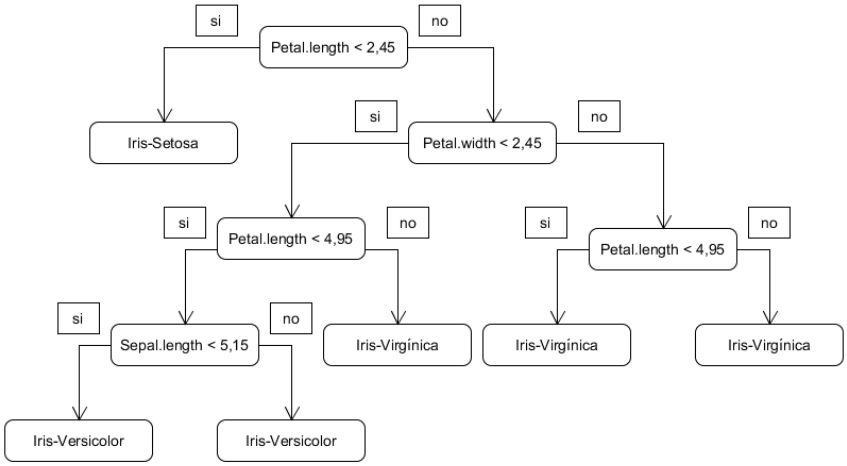

Figura 3.2: Ejemplo Árbol de decisión

3.6. Redes Neuronales

Las redes neuronales artificiales son una de las ramas de la computación de las denominadas «Bio-inspiradas", es decir, tratan de emular de algún modo alguna de las características propias de los seres vivos. Muchos de los seres vivos precisan de neuronas para poder llevar a cabo sus actividades motoras. Una neurona es un tipo de célula cuya función es transmitir impulsos al sistema nervioso [11]. En nuestro caso tendremos una representación digital de estos impulsos. Las neuronas de una red neuronal, al igual que ocurre en la naturaleza, se van activar conjuntamente de forma que, neuronas que se activan conjuntamente con mucha frecuencia, refuercen sus uniones para que, cuando se tengan que volver a activar a causa de un estímulo externo, lo vuelvan a hacer conjuntamente. Este proceso de reforzar (y debilitar) las uniones es lo que se conoce como aprendizaje.

A rasgos generales las redes neuronales realizan dos tipos de funciones:

- Clasificación: En la clasificación el agente inteligente que implementa la red neuronal ha de devolver una de las salidas ya especificadas anteriormente. Lo que es lo mismo, «Clasificar" nuestros datos de entrada en uno de los grupos esperados.

- Regresión: En este caso nos va a devolver un valor numérico que se ajusta a una función continua, por este motivo a la regresión lineal también se la denomina ajuste de funciones.

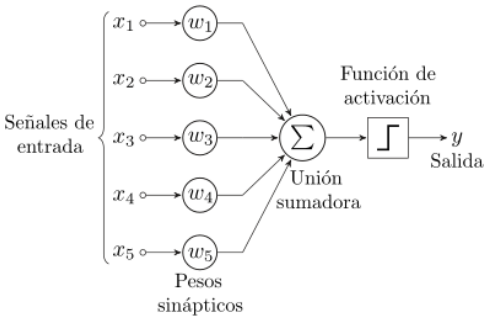

Figura 3.3: Diagrama de un perceptrón simple con cinco entradas

Perceptrón

El modelo más simple que podemos encontrar es el perceptrón. El perceptrón es un modelo neuronal de una sola neurona con una única salida [12].

Conectando entre sí varios de estos perceptrones podríamos comenzar a hablar de redes neuronales.

3.7. Algoritmos genéticos

Los algoritmos genéticos son una parte de la inteligencia artificial cuyo objetivo es optimizar los resultados intentando emular las leyes de la naturaleza, la evolución natural [1]. Es un tipo de algoritmo muy eficaz en la mayor parte de problemas dada su gran versatilidad.

La clave de un algoritmo evolutivos reside, como en la naturaleza, en la supervivencia de los mejor adaptados [5]. Para llegar a estos individuos mejor adaptados, siempre se suelen seguir los mismos pasos. Los paso a seguir son inicialización, evaluación, selección, cruce y, opcionalmente, mutación.

Antes de todo este proceso hay que analizar el problema y decidir cuál va a ser su representación, la representación de un individuo. Comúnmente este tipo de problemas se representan en forma de array unidimensional en el que cada posición de éste representará una característica del individuo. Los atributos o características del individuo son valores numéricos que, dependiendo del problema, pueden ser binarios, continuos, discretos, etc. Cuanto mejor se represente el problema, evidentemente, más probable es que consigamos alcanzar un resultado óptimo.

Inicialización: Para la inicialización se generan individuos aleatorios, esto es, rellenar con valores aleatorios los arrays que representan a nuestros individuos. Un array define las características de un individuo y un conjunto de

0	1	1	0	0	1

Figura 3.4: Representación del individuo

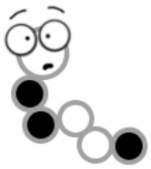

Figura 3.5: Individuo

individuos define una población. Esta población inicial, partiendo del absoluto azar será previsiblemente mala.

Evaluación: Para estimar la calidad de un undividuo debemos definir una función de *fitness*, es decir, una función que nos de un valor (*fitness*) comparable que nos indique lo bueno o malo que es frente al resto de la población.

Selección: Una vez ya hemos evaluado a todos los individuos se pasa al proceso de selección. En este punto se pueden descartar directamente aquellos individuos que cuyo *fitness* no supere un umbral mínimo definido por nosotros. Pasado este primer corte procedemos a elegir los mejores indivíduos, existen diferentes criterios de selección: selección Proporcional (o Ruleta), muestreo estocástico universal, selección por torneo, etc.

Cruce: Para poder seguir avanzando a lo largo de las generaciones necesitamos generar nuevos individuos. Uno de los operadores encargado de la generación de nuevos individuos es el operador de cruce. El operador de cruce establece una probabilidad determinada de que un individuo sea seleccionado para el cruce, si este supera ese umbral el individuo procederá a cruzarse. El cruce se realiza intercambiando secciones de su genotipo en uno o varios puntos aleatorios. En la figura 3.6 tenemos un ejemplo de cruce monopunto [2].

Figura 3.6: Ejemplo de cruce Monopunto

Mutación: Otro de los operadores más utilizados en la generación de nuevos individuos es el operador de mutación. Cuando se va a proceder a la mutación se establece una probabilidad de mutación y se va evaluando cada elemento

del genotipo, es decir, se saca un número aleatorio y si este supera el umbral de mutación, ese gen resulta mutado. Durante el proceso de mutación pueden surgir individuos peores que sus predecesores, por lo tanto habrá que ser cauto estableciendo el parámetro de mutación.

Reemplazo: Para finalizar, antes de comenzar de nuevo todo este proceso, debemos reemplazar los individuos descartados por lo nuevos para conformar la nueva población. Una vez hecho este reemplazo, se procede de nuevo con todos los pasos. Hay varios criterios de parada, los más comunes suelen ser por convergencia, por tiempo o por número de generaciones.

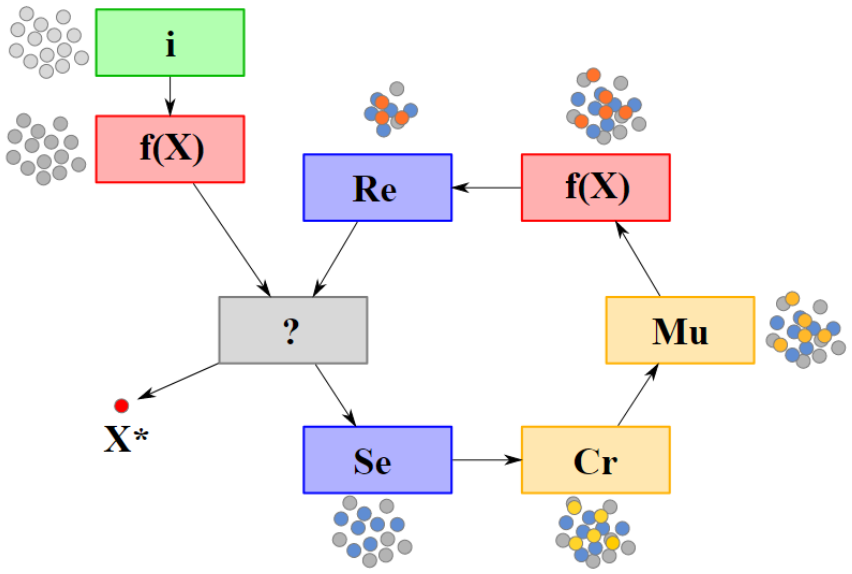

Figura 3.7: Esquema de algoritmo evolutivo simple

Técnicas y herramientas

Para la realización de este proyecto se han empleado las siguientes herramientas:

4.1. Photoshop

Photoshop es un software profesional de edición fotográfica. El mi proyecto ha sido utilizado para dibujar toda la interfaz gráfica de juego.

4.2. Visual Studio

Visual Studio es un IDE de Microsoft que sirve para desarrollar aplicaciones para múltiples plataformas. Este IDE ha sido utilizado para escribir el código en c correspondiente a la funcionalodad del videojuego.

4.3. Scikit-Learn

Scikit-learn es una biblioteca de herramientas de Machine Learning [3]. Esta biblioteca nos proporciona una serie de algoritmos de aprendizaje supervisado y aprendizaje no supervisado, todo esto implementado en python. Esta biblioteca se desarrolló a partir, o haciendo uso de SciPy, por lo que también debe ser instalada. Algunos enlaces de interés son un tutorial para crear un modelo sencillo rápidamente: http://scikit-learn.org/stable/tutorial/basic/tutorial.html y, por otro lado, la guía de usuario en la que se detallan todas la posibilidades de sikit-learn http://scikit-learn.org/stable/user_guide.html.

Como herramienta para crear modelos de regresión, clasificación, clustering... etc. es maravillosa, pero necesitamos también otra herramienta que nos permita hacer un pre-procesado de los datos de los que disponemos. Para ello necesitaremos Pandas.

4.4. Pandas

Para el manejo de la enorme cantidad de datos obtenidos se ha utilizado Pandas. Esta herramienta nos permite organizar y clasificar grandes cantidades de datos[6, 7]. Mediante el uso de dataframes podemos separar, unir, reorganizar, en fin, realizar casi cualquier tipo de operación sobre grandes bloques de datos. De esta manera, con poco esfuerzo podía elegir qué datos quería emplear para el entrenamiento, concatenar instancias... etc.

4.5. DEAP

DEAP ha sido otra de las bibliotecas utilizadas a lo largo de este proyecto. Llegados a cierto punto se requería que las predicciones fueran mas precisas, requqrían un nivel mas profundo de entendimiento [4] por este motivo, se introdujo el uso de DEAP. DEAP es una biblioteca de computación neuronal que te permite implementar de una formas rápida y relativamente sencilla modelos evolutivos.

4.6. Unity

Como base para el grueso de proyecto se ha utilizado el motor de desarrollo de videojuegos Unity.

4.7. GitHub

GitHub es un herramienta que nos permite llevar un control de versiones de nuestro código fuente, además de permitir el trabajo colaborativo.

Aspectos relevantes del desarrollo del proyecto

5.1. Descripción del juego

Introducción

Para entender todo el proceso de entrenamiento del bot inteligente que voy a implementar considero necesario entender el funcionamiento y reglas del juego así como ciertas particularidades relacionadas con la captura y uso de datos.

Diseño de reglas y desarrollo del juego

El juego va a ser un «mata-marcianos» de tipo arcade, un juego infinito en el que el jugador aspira a alcanzar la máxima puntuación de forma individual. Este tipo de juegos era típico de los salones recreativos y centros comerciales en la década de los años ochenta. En este caso nos ponemos a los mandos de una nave espacial que ha de evitar que los enemigos invasores traspasen la zona inferior de la pantalla (nuestro territorio). Los enemigos aparecen en oleadas y se limitan a avanzar oscilando hacia su objetivo. El jugador ha de eliminar el máximo número de enemigos posibles sin impactar contra ellos[1]. Cuanto menos se acerquen a la zona inferior de la pantalla mayor será la puntuación que otorgue su eliminación. Si los enemigos llegan a nuestro territorio restarán una cantidad fija de puntos, por lo que nuestra puntuación final se verá notablemente mermada.

En el apartado gráfico se optó por un aire retro, algo que recordase a las primeras máquinas arcade o consolas de videojuegos. En primer lugar me decanté por imitar el estilo de la *Game Boy*, consola que triunfó en los 90.

[1] Está era una limitación inicial ya que en futuras iteraciones se implementa un método que hace irrelevante el número de enemigos simultáneamente en pantalla.

Figura 5.8: Paleta de colores

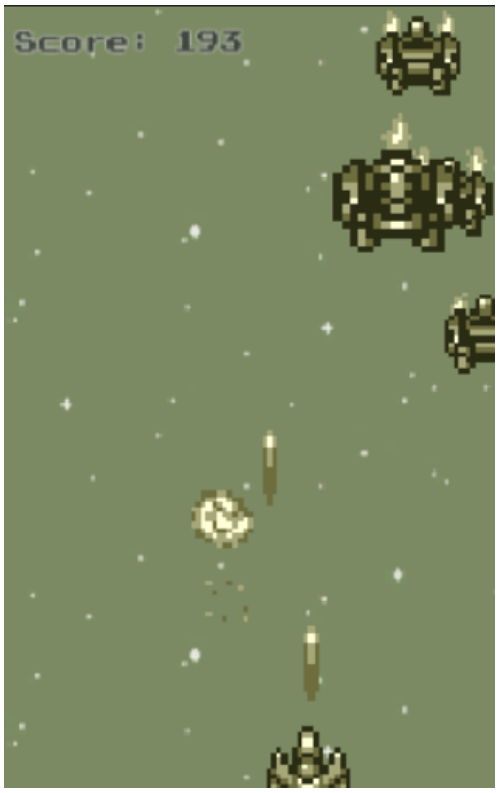

Figura 5.9: Interfaz. Versión 1

Para ello se seleccionó una paleta de colores que recordase a los gráficos de la época. Para el jugador y los enemigos se utilizó un pack de Sprites gratuito de la tienda de Unity3D[2].

Evolución de las reglas del juego

Ya tenía la primera versión lista para jugar. En este punto se decidió establecer unas reglas un poco más «especiales» para simplificar el proceso. En primer lugar, los enemigos no atacan al jugador con ningún tipo de proyectil, se limitan a avanzar oscilando a una velocidad constante. Por otro lado, se limitará el número de enemigos visibles simultáneamente en la pantalla con el

[2]Estos Sprites solo se utilizaron en la versión inicial, más adelante son sustituidos por unos nuevos hechos por mí.

fin de poder hacen una instancia de tamaño constante y no perder enemigos en la captura de datos. Para darle un podo de dinamismo y evitar que el jugador deje pulsada la tecla de disparo infinitamente, se ha implementado un sistema de calentamiento del arma. Este sistema hace que los disparos consecutivos calienten el arma y si no se dispara, se enfríe lentamente. Si el arma llega al máximo de temperatura dejará de disparar. El sistema de calentamiento del arma dio lugar a la idea de implementar unos «power-up» o potenciadores. Estos serán de dos tipos, uno enfría de golpe el arma una determinada cantidad y otro duplica en número de balas disparadas simultáneamente. Esto, como se describirá más adelante, condicionará la estructura de las instancias.

Como última aclaración, los «sujetos de prueba», a los que el agente inteligente deberá imitar, se aburrían jugándolo, por lo que se introdujo un jefe que tendría una enorme cantidad de vida. Este jefe aparecerá una vez por minuto jugado y restará muchos más puntos que los enemigos estándar. La eliminación de este tipo de enemigo conlleva un gran calentamiento del arma y, por tanto, un estrés añadido al jugador, que tendrá que estar listo en todo momento para este tipo de combate.

Evolución del interfaz gráfica

La primera versión de la interfaz del juego tenía una *aspect ratio* de 5:3, lo que no le hacía agradable a la vista, ni dejaba espacio para otros elementos de la interfaz. Por este motivo, se decidió revisar el diseño inicial.

Se diseñó un menú principal que dejaba elegir entre empezar a jugar o salir del juego. Esta pantalla fue diseñada por el ilustrador Luis G. Contreras[3]. Para la interfaz del juego se amplió la paleta de colores, dejando de lado el aspecto *Game Boy* pero manteniendo un aire retro.

5.2. Comunicación entre Scripts

introducción

Para el desarrollo del proyecto, se han utilizado dos lenguajes de programación. Por un lado c#, que ha sido uno de los lenguajes más utilizado para el desarrollo de videojuegos en la historia reciente, por otro, Python, que es el lenguaje en el que están las bibliotecas utilizadas para el entrenamiento de nuestro sistema inteligente. Por este motivo, era necesario encontrar la forma de que el juego pudiera ejecutar los scripts de Python, además de pasarle la correspondiente información. El proceso de comunicación ha sido uno de los puntos que más problemas me ha dado a lo largo del desarrollo del proyecto, ya que hubo que probar varias alternativas hasta dar con la correcta.

[3] http://www.borratajo.es/

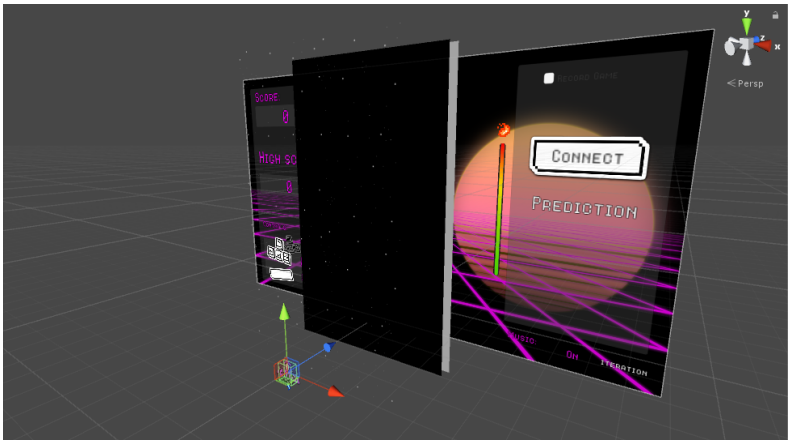

Figura 5.10: Interfaz de desarrollo Unity3D

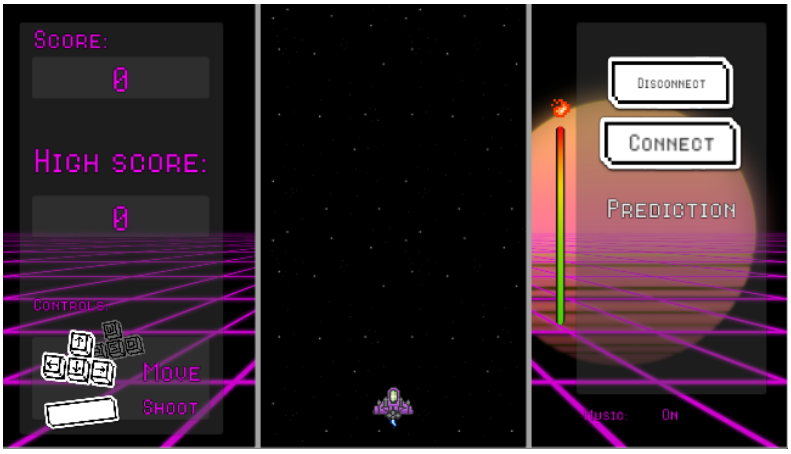

Figura 5.11: Evolución de diseño de la interfaz

Pythonet

En primer lugar estudié la posibilidad de integrar Python en c#, de tal forma que pudiese ejecutar pequeños scripts de Python dentro de mi código en c#. Para esta integración se probé a utilizar Pythonet. Pythonet, como pone en su pagina web[4], permitiría a Python interactuar con el CLR[5] de .net e incrustar código Python en .net. Tras númerosas pruebas no hubo forma de ejecutar una sola línea de Python desde un script Monobehaviour. MonoBehaviour es la clase maestra de la que heredan todos los scripts de unity. Esta clase proporciona las fuciones necesarias para la ejecución del juego. Las funciones

[4]https://github.com/Pythonnet/Pythonnet

[5]CLR: (Common Language Runtime) Máquina virtual de .net que controla la ejecución del le srcipts .net

elementales son Start() y Update(), pero disponemos de muchas más.

IronPython

La siguiente herramienta con la que se estuvo trabajando fue IronPython. IronPython prometía unas utilidades muy similares a Pythonet, o incluso mejores, pero tras varias horas de lectura de la documentación encuentro algo que tira por tierra mis expectativas, IronPython aún no es compatible al cien por cien con Scikit-learn.

Pipes o tuberías

Comienza una nuevas vía de investigación, era el turno de los pipes (o tuberías) de Python. En internet no había demasiada información y la poca que había no era muy clara, pero aún así me puse manos a la obra y a las pocas horas ya tenía dos scripts de Python comunicándose entre ellos. Bien, ahora solo quedaba que se comunicase con mi aplicación en .net. Este ultimo paso no hubo forma de llevarlo a cabo, no conseguí que ambas aplicaciones intercambiasen mensajes de forma correcta.

Sockets

Finalmente, me decanté por utilizar, o bien WebServices, o bien sockets. Como la opción de los sockets me parecía en principio más sencilla me puse manos a la obra. Primero hice un servidor de sockets y un cliente en Python, y funcionó. El script era sencillo, un simple eco, cuando alguien se conectaba enviaba de vuelta toda la información que recibiese. El siguiente paso lógico era hacer lo propio en .net, y funcionó, ya tenía las aplicaciones intercambiando mensajes.

El funcionamiento final consiste en un script de Python que al ejecutarse abre un socket anónimo que se queda a la espera de una conexión, cuando alguien se conecta empieza a escuchar y procesar todos los mensajes que le llegan pasándoselos al modelo del sistema inteligente, una vez procesados, por la misma vía responde con el resultado calculado.

5.3. Definiendo las instancias

Para dotar a la máquina de la capacidad de aprender es necesario, como en el aprendizaje de los seres humanos, una serie de estímulos. Estos estímulos, en mi caso, se traducen en una serie de estados o instancias. Para todas la pruebas iniciales se empezó utilizando un clasificador Random forest[6].

[6] Al poco tiempo se descartó la validación cruzada ya que en la teoría rozaba el 97 % de acierto y en la práctica se alejaba mucho de la realidad y , por lo tanto, no nos sirve para validar con qué eficacia imita al humano

Player - x	Player - y	Enem1-x	Enem1-y	Enem2-x	Enem2-y	Enem3-x	Enem3-y	Enem4-x	Enem4-y	Clase
0.00	0.00	0.25	8.75	999.00	999.00	999.00	999.00	999.00	999.00	RightArrow
0.56	0.00	0.50	8.02	999.00	999.00	999.00	999.00	999.00	999.00	None

Tabla 5.3: Primera versión Instancia

Player x	Player y	Enem1 x°	Enem1 y	Enem 1 Salud	Enem2 x	Enem2 y	Enem 2 Salud	Enem3 x	Enem3 y	Enem 3 Salud	Enem4 x	Enem4 y	Enem 4 Salud	Clase
0.00	0.00	0.25	8.75	3	999.00	999.00	3	999.00	999.00	3	999.00	999.00	3	RightArrow
0.56	0.00	0.50	8.02	3	999.00	999.00	3	999.00	999.00	3	999.00	999.00	3	None

Tabla 5.4: Segunda versión Instancia

Player x	Player y	Enem1 x	Enem1 y	Enem 1 Salud	Enem2 x	Enem2 y	Enem 2 Salud	Enem3 x	Enem3 y	Enem 3 Salud	Enem4 x	Enem4 y	Enem 4 Salud	Clase
0.00	0.00	0.25	8.75	3	999.00	999.00	3	999.00	999.00	3	999.00	999.00	3	RightArrow
0.56	0.00	0.50	8.02	3	999.00	999.00	3	999.00	999.00	3	999.00	999.00	3	None

Tabla 5.5: Tercera versión Instancia

Para empezar, se comenzó seleccionando los que parecían más relevantes, como la posición absoluta del jugador y la posición absoluta de los enemigos en el campo de batalla. Solo formaba parte de la clase la última tecla pulsada por el usuario, que podía ser Arriba, Abajo, Izquierda, Derecha, Disparo o No disparo. Para entrenar un agente en número de atributos de entrada debe ser inmutable, por lo que se estableció un máximo de cuatro enemigos en pantalla de forma simultánea. Dado que yo estaba programando el videojuego yo tenía el control absoluto de todas las variables.

Tras un primer entrenamiento, con un número no muy alto de instancias (4000) los resultados no eran muy satisfactorios. En siguiente iteración se observó que había otro parámetro más que nos podría ayudar a tomar decisiones, la vida del enemigo.

De forma paralela se iban realizado tests de jugabilidad con diferentes usuarios, dado que el arma se calentaba rápidamente, se optó por introducir en el juego los «power-up». Llegados a este punto se decidió seguir aumentando el número de atributos con el fin de extraer toda la información que podría tener un humano de la partida. Así que ahora la instancia tendrá también la posición absoluta de los «power-up».

La posición de los enemigos, como se puede ver en las ilustraciones adjuntas, venían dados por posiciónX, posicionY y vida restante. Para facilitar al modelo la clasificación lo que hago el crear un «heat-map» con las posiciones de los enemigos en una matriz. Este heat-map o mapa de densidad proporciona la información de la cantidad de enemigos que hay en un un sector concreto. Pudiendo establecer el tamaño...

Ya tenía definidas las instancias con las que empezarían mis primeros entrenamientos. Esta vez comencé con un dataset con menos de 1000 entradas y

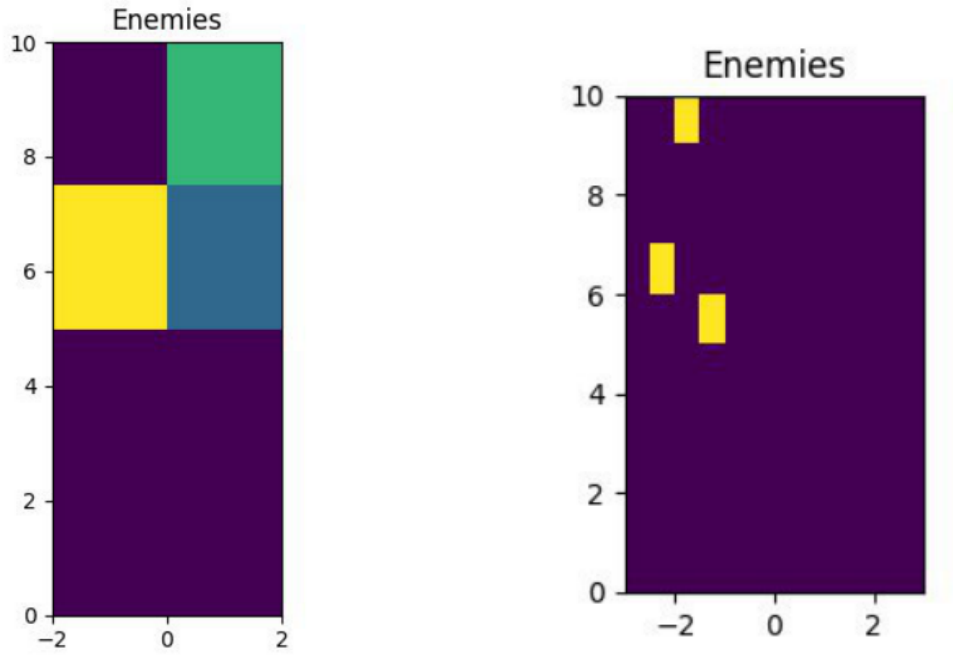

Figura 5.12: Heat-maps con diferentes garnularidades

los resultados seguía siendo muy mejorables, la aletoreidad de los movimientos era evidente. ¿Qué parámetros estaba teniendo yo en cuenta que no le estaba transmitiendo la instancia? La respuesta es «el tiempo». Yo, cuando jugaba, era consciente de varios atributos que no eran plasmables en una «Fotografía», que es lo que venía siendo la instancia hasta ahora. Así que para esto se seleccionaron grupos de n instancias consecutivas y se concatenaron para formar una nueva instancia n veces la inicial. Los atributos de mi nueva instancia serían ahora:

[instanciaT-n...instanciaT-2 instanciaT-1 instanciaActual][clases actuales]

En esta primera concatenación los resultados mejoraron notablemente. El juego se movía por fin de una forma aceptable. Aún así, se decidió incluir en la instancia los movimientos realizados en las acciones anteriores, ya que se pensó que podría ser que también fuesen relevantes.

Se realizaron entrenamientos de prueba con varios dataSet y se observó un problema común en todas la iteraciones. Cuando se acercaba a los bordes de la zona de juego, dejaba de responder correctamente, realizando movimientos aleatorios, llegando incluso a dejar de actuar totalmente. El problema residía en la forma de jugar del los usuarios.

Los jugadores humanos tienden, como es lógico, a jugar en el centro de la pantalla, por lo que en el centro de la pantalla se tienen muchas instancias de

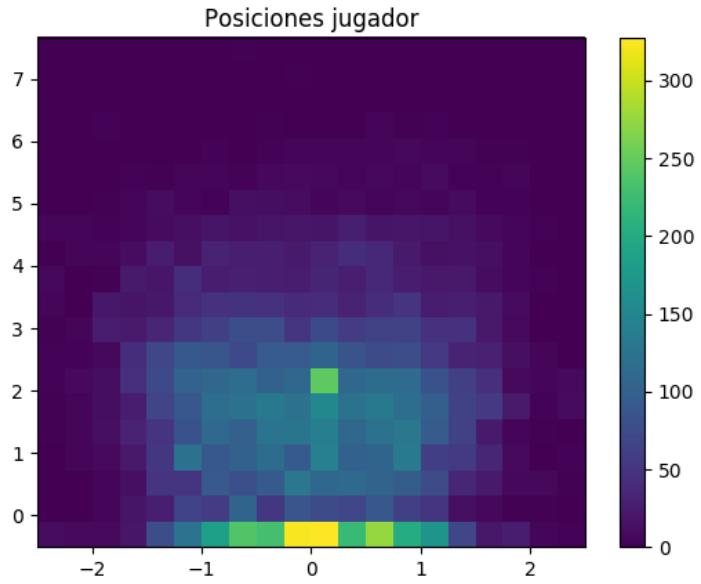

Figura 5.13: Posiciones del jugador

las que aprender en esa zona, pero en los bordes, las instancias eran casi nulas. La figura 5.3 representa la cantidad de muestras (instancias) en relación a la posición absoluta del jugador en el terreno de juego.

En este punto hay un cambio radical en la estructura de las instancias. Ya que una posición absoluta del jugador y los enemigos da lugar a estados únicos muy amplios. Decidí que una buena forma de reducir esta gran cantidad de estados únicos era trabajar con la distancia relativa del jugador a los diferentes enemigos.

5.4. «visión» del bot

La representación por la que me decanté finalmente fue la que almacenaba la distancia relativa a los diferentes enemigos. Para obtener una cantidad de atributos constante, lo que se hizo fue definir una serie de rayos procedentes del jugador como se ve en la ilustración 5.14.

Para obtener la información, se irán haciendo barridos cada 0.25 segundos. Si el rayo impacta contra un enemigo se almacenará la distancia en línea recta al mismo. Si el rayo no impacta contra nada, se insertará un valor numérico especial para indicar la ausencia de información. Ésta última aproximación tiene la ventaja de que nos va a ser indiferente la cantidad de enemigos en pantalla, ya que lo que almaceno el la presencia o no de los mismos en mi

Tipo	Atributos	Total
Información del jugador	PosisiónX, PosiciónY, Temperatura del arma	3
Información del enemigo	PosX 1-6 PosY 1-6 Vida restante 1-6	18
Información Power-Up	Power-Up1 PosX Power-Up1 PosY Power-Up2 PosX Power-Up2 PosY	4
Otra información	TimeStamp Score	2
Clases	Arriba-Abajo Izquierda-Derecha Dispara-NoDispara	3
	Total	30

Tabla 5.6: Cuarta versión Instancia

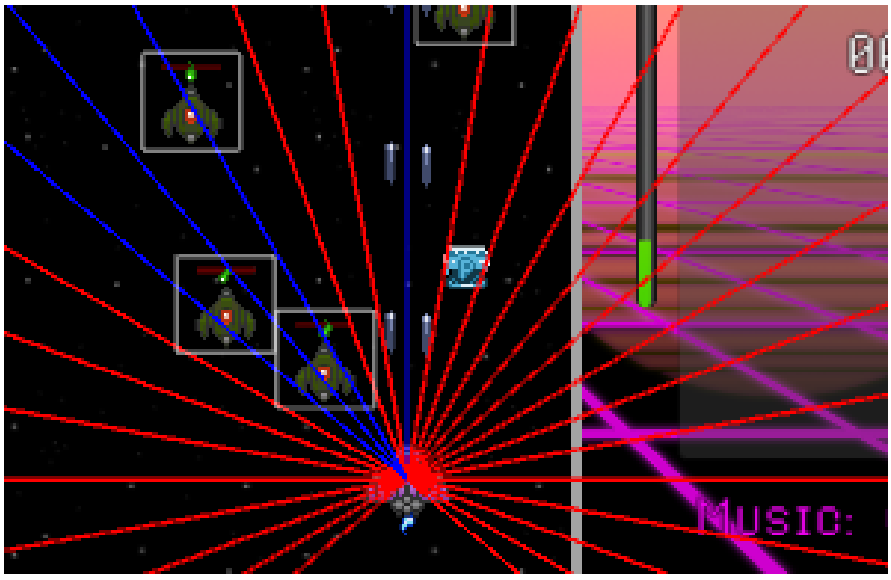

Figura 5.14: Visión del Bot

Tipo	Atributos	Total
Información del jugador	PosisiónX, PosiciónY, Temperatura del arma	3
Información del enemigo	PosX 1-6 PosY 1-6 Vida restante 1-6 Raycast 1-27	45
Otra información	TimeStamp Score	2
Clases	Arriba-Abajo Izquierda-Derecha Dispara-NoDispara	3
	Total	53

Tabla 5.7: Quinta versión Instancia (Final)

campo visual. Por otro lado tiene el inconveniente de que un enemigo me puede ocultar la presencia de otro que se encuentre detrás y que, hasta que este primero no sea destruido, no nos va a ser posible conocer más información.

Finalmente tenemos una instancia con mucha información, de la cual podemos seleccionar qué atributos vamos a utilizar y cuales no.

5.5. Entrenamiento

Una vez tenemos definido el tipo de instancia y el conjunto de datos con los que vamos a trabajar procedemos a entrenar el modelo del agente inteligente. Se optó por empezar entrenando un clasificador sencillo, un clasificador que nos permitiese saber hasta dónde podría llegar, ya que en un principio desconocía por completo las herramientas de aprendizaje máquina.

5.6. Entrenamiento con árboles de decisión

La primera opción barajada fueron los árboles de decisión. Este es uno de los muchos clasificadores que nos proporciona la biblioteca sicikit-learn. Se utilizaron dos tipos de clasificador, un de árbol de decisión (Decission Tree) y un Random-forest.

La implementación de este modelo consta de varios scripts de Python que realizarán las tareas de entrenamiento, procesado de instancias y predicción. En el apartado de entrenamiento ambos tienen una implementación idéntica,

sólo vamos a cambiar la invocación al un método u otro a la hora de generar el modelo.

Para utilizar este modelo de entrenamiento he implementado dos scripts. El primero de ellos, denominado «Trainer.py» tiene la función de generar y guardar el modelo. El segundo, «Predictor.py» , mucho mas simple, se limita a llamar a la función predict del modelo. A pesar de ser un script muy elemental, se decidió dejarlo por separado por si era necesario un preprocesado antes de devolver el resultado, cosa que finalmente sí que ocurrió.

5.7. Entrenamiento con algoritmo evolutivo

En primer lugar, nuestra idea era evolucionar un modelo entrenado para que funcionase por imitación, por este motivo se comenzó utilizando conjunto de datos utilizado en los casos anteriores para entrenar un perceptrón multicapa (MLP).

Todo este proceso lo vamos a realizar utilizando un *pipeline*. Un *pipeline* nos permite realizar varios pasos para crear un modelo de forma automática. En este caso nuestro *pipeline* consta de dos pasos. El primero es un *Standard Scaler*, que nos va a tomar los elementos del conjunto de datos y nos los va a normalizar entre 0 y 1. Este paso se realiza para reducir el ruido que nos podemos encontrar, haciendo que los valores más alejados del conjunto principal tengan menos efecto en la toma de decisiones. El segundo paso es el clasificador, que va a conformar el modelo. Este clasificador, que nos proporciona la biblioteca scikit-learn, tiene unos parámetros definidos por defecto, pero podemos indicar los nuestros para conseguir que se adapte mejor a nuestro problema.

Los primeros entrenamientos se realizaron con los parámetros por defecto del MLP que nos proporciona la biblioteca sckit-learn. Los resultados de estos entrenamientos eran desastrosos por lo que se procedió a realizar un calibrado de parámetros del MLP. Para el calibrado de parámetros se utilizó una de las herramientas que nos ofrece scikit-learn. De las dos fórmulas que nos ofrece me decanté por el Randomized Parameter Optimization. Ésta herramienta lo que hace es que, dados unas posibilidade de determinados parámetros, él los va combinado y probando aleatoriamente, de manera que finalmente te indique cuál ha dado mejores resultados. No podemos tener la certeza de que esa combinación de parámetros vaya a ser la mejor, pero ahorra tiempo.

Una vez calibrados los parámetros del MLP se procede a la neuroevolución de la red. Para este proceso se utilizaron las herramientas que nos proporciona DEAP[7].

[7]Se habla de esta biblioteca en el apartado Técnicas y herramientas

En el proceso de neuroevolución hay que preparar el entorno en el que vamos a trabajar. Como ya se introdujo en el apartado de *Conceptos teóricos*, esta preparación consta de: definir el individuo, indicar el tipo de cruce y especificar los porcentajes tanto de mutación como de cruce.

Para la definición de los individuos se utiliza un array conformado por los pesos de la red neuronal del MLP. Dado que los pesos de la red neuronal no tienen estructura de array, ha sido necesario implementar una función que realice esa tarea. Ahora, cuando la red neuronal genere un individuo lo hará en forma de array pero, cuando se proceda a probar el modelo, se volverá a convertir para que pueda ser utilizado por la red neuronal.

La función de fitness viene dada por la media de los n intentos por cada partida jugada. La media de las partidas jugadas se calcula de la siguiente manera: Cada individuo juega una partida. Una partida se compone de n intentos. En cada partida, el juego genera un csv en el que se va almacenando la puntuación de cada intento. Al final de todos los intentos, la función de fitness lee el fichero y hace la media. Esa media es el valor de fitness que buscamos.

Hay que definir, también, una función que genere la población inicial de acuerdo a nuestras demandas. En este caso la función generará n arrays con valores aleatorios acotados a nuestro gusto.

De todos los entrenamientos realizados creo que es interesante analizar los resultados del modelo finalmente entregado:

Para la primera fase del entrenamiento se generó un clasificador MLP con seis neuronas[8]. Las poblaciones se generaron con un tamaño de diez individuos de los cuales, cuatro serían idénticos al MLP generado a partir de los datos tomados. El hecho de introducir individuos no generados aleatoriamente, nos permite «forzar» a nuestros individuos a evolucionar en una dirección, concretamente, hacia los resultado obtenidos de los jugadores humanos. Además, introducir estos individuos agiliza notablemente el aprendizaje, ya que partimos de un modelo parcialmente entrenado, lo que nos evita todo el proceso de ensayo error del principio.

Como se puede observar en la tabla 5.8, la función converge bastante rápido, esto es observable en la cuarta generación, pues a partir de esta no cambian los resultados. Los resultados, llegados a este punto eran bastante satisfactorios, la puntuación era alta, pero estaba seguro de que eso se podía mejorar.

Realicé un segundo entrenamiento, esta vez, el modelo inicial sería el mejor modelo procedente del primer entrenamiento. Del mismo modo que se procedió la primera vez, diez generaciones, poblaciones de diez individuos, ect. En la

[8]Este número no es un número arbitrario, surge de la evaluación mediante validación cruzada de varios modelos generados con diferentes número de neuronas

gen	nevals	avg	std	min	max
0	10	-376515	527862	-1124.18	601929
1	5	601929	0	601929	601929
2	7	601929	0	601929	601929
3	8	601929	0	601929	601929
4	8	1626.53	2,27374E-08	1626.53	1626.53
5	9	1626.53	2,27374E-08	1626.53	1626.53
6	6	1626.53	2,27374E-08	1626.53	1626.53
7	9	1626.53	2,27374E-08	1626.53	1626.53
8	8	1626.53	2,27374E-08	1626.53	1626.53
9	6	1626.53	2,27374E-08	1626.53	1626.53
10	5	1626.53	2,27374E-08	1626.53	1626.53

Tabla 5.8: Resultados primer entrenamiento

gen	nevals	avg	std	min	max
0	10	1443.61	2013.27	-1181.88	4802.25
1	7	4712.18	156006	4441.97	4802.25
2	6	4802.25	0	4802.25	4802.25
3	8	4802.25	0	4802.25	4802.25
4	7	4802.25	0	4802.25	4802.25
5	6	4802.25	0	4802.25	4802.25
6	6	4802.25	0	4802.25	4802.25
7	7	4802.25	0	4802.25	4802.25
8	5	4802.25	0	4802.25	4802.25
9	5	4802.25	0	4802.25	4802.25
10	4	4802.25	0	4802.25	4802.25

Tabla 5.9: Resultados segundo entrenamiento

tabla 5.9 vemos que ahora converge más rapido incluso que antes y, además, incrementa la puntuación media hasta casi cuatro veces la mejor obtenida en el anterior entrenamiento.

Bot de Telegram

Fueron muchas la pruebas que se realizaron en el apartado de algoritmos evolutivos. Estas pruebas requerían mucho tiempo y cierta supervisión, así que, por mera curiosidad, quise ver si podía programar un bot que me enviase mensajes a través de la aplicación de mensajería Telegram. En teoría un bot de Telegram debería estar sempre activo e ir almacenando las id de los usuarios que quieren interactuar con el. Yo no quería dedicarle demasiado tiempo a ello, ya que no era materia fundamental de este proyecto, por lo que me limité

a obtener mi id de usuario, almacenarlo y utilizar únicamente la función de «Enviar mensaje» para avisarme de determinados eventos.

5.8. Mini-proyectos Unity3D

A lo largo de la tarea de investigación, hubo que realizar pequeños proyectos en Unity con una funcionalidad muy concreta. Los proyectos tenían como finalidad probar si se podían realizar determinadas tareas en el entorno de desarrollo de Unity. El primero que se creó, fue el que sirvió como campo de pruebas para testear la conexiones entre los scripts de Unity y los scripts de Python. Por otro lado, tuve que hacer otro mini-proyecto para probar cómo podía lanzar mi juego desde la terminal y qué capacidad tenía para pasarle parámetros al mismo. Este último surgió a raíz de la necesidad de lanzar el juego desde Python, ya que el algoritmo evolutivo tenía que tener la capacidad de lanzar el juego para calcular el fitness del individuo.

Trabajos relacionados

Una de las motivaciones que me llevaron a proponer este tipo de proyecto fue el trabajo realizado por Katharina Muelling, Jens Kober y Jan Peters *Learning Table Tennis with a Mixture of Motor Primitives*[8]. En este trabajo entrenan un brazo robótico para jugar al ping-pong. Este entrenamiento parte, como en mi proyecto, de un aprendizaje por imitación. Puede verse un vídeo sobre esto en https://www.youtube.com/watch?v=SH3bADiB7uQ

Conclusiones y Líneas de trabajo futuras

Para este proyecto se entrenaron numerosos modelos hasta quedarme con los detallados anteriormente. Creo no equivocarme si digo que la mejor forma de evaluar estos modelos, es de forma empírica, es decir, observando cómo juegan y qué resultados obtienen.

Por un lado tenemos los clasificadores. Se entrenaron tres modelos, uno con Decision Tree simple, otro con Random Forest y un tercero que se trataba de un Decision Tree aleatorio (con datos de entrada barajados). Estaba seguro de que el que mejor iba a funcionar iba a ser el Random Forest, un tipo de clasificador mas sofisticado que su contrincante. Contradiciendo a mi hipótesis inicial, el que mejor funcionó fue el Decision tree y analizándolo detenidamente tiene sentido. Cuando tenemos un conjunto de datos no muy grande, como fue mi caso, no conviene crear estados muy concretos si no que conviene generalizar mucho mas y dejar hueco a la improvisación. En otras palabras, el random forest sobreajustaba la función. Para mi sorpresa, los dos clasificadores mencionados no distaban mucho del modelo entrenado aleatoriamente, lo que hace plantearme varias cosas. ¿Son el random forest y el Decision tree malos algoritmos para el caso que nos ocupa?, ¿El conjunto de datos era insuficiente y le hace comportarse como si de un aleatorio se tratase? ¿Mi aleatorio es tremendamente afortunado? Para resolver estas dudas había que probar con un dataset mucho mas grande y generar varios modelos más de clasificadores aleatorios, pero había que pasar a la parte evolutiva.

Por otro lado tenemos el perceptrón multicapa con una red neuronal convolucional, el claro vencedor. Este algoritmo comienza también con un set de datos de entrada que le servirán como referencia para la tarea de imitación y a continuación se le hace evolucionar. Bien es cierto que ésta neuroevolución ha empleado más de cuatro horas para llegar donde ha llegado, pero ha merecido la pena. En las primera generaciones se podría decir que imitaba bien, de una

forma aceptable, al humano pero no se diferenciaba notablemente de los clasificadores empleados en la etapa anterior. Según fueron pasando generaciones el modelo se fue alejando de ese estilo de juego para pasar a una estrategia más mecánica, descubriendo un error que cometí a la hora de generar los enemigos. Este error hace que el punto de partida de mi generador de enemigos los genere siempre en un punto aleatorio de la mitad izquierda de la pantalla, por lo que en el lado derecho nunca se generan. El modelo final centra todos sus disparos precisamente en esta zona, dejando la zona derecha prácticamente sin visitar.

Por lo tanto, una vez analizados los resultados, podemos concluir que teniendo en cuenta la habilidad del modelo entrenado el vencedor es el Perceptrón multicapa, ya que ha obtenido las máximas puntuaciones. Si tenemos en cuenta la forma de comportarse del modelo ante los estímulos, los del perceptrón se ven muy mecánicos, no parecen humanos, por lo que yo me decantaría por del Decision tree, que el mas «humano» aunque mucho mas torpe.

Cuando se comenzó a trabajar en este proyecto ya tenia algún conocimiento de ciertos aspectos del aprendizaje máquina, pero desconocía totalmente cómo se había de llevar a cabo y qué herramientas existían para ello. Fue una gran noticia descubrir las bibliotecas de scikit-lean y su relativa simplicidad de uso. Debido a este desconocimiento, tuve que dedicar mucho tiempo a investigación y, por desgracia, me quedé con las ganas de, ahora que tenía los conocimientos, realizar algún proceso más exhaustivo.

El juego tiene implementados dos tipos de «power-up», o potenciadores que mejoran temporalmente las capacidades de la nave del jugador. La intención era que fuesen apareciendo aleatoriamente durante la partida y el jugador interactuase ellos. Finalmente solo de dejaron los «power-up» que aparecen al eliminar al jefe de final de nivel. Otro punto interesante es dotar a los enemigo de cierto poder ofensivo, ya que actualmente solo avanzan de forma pasiva. Creo que el hecho de que los enemigos ataquen da un punto interesante de complejidad.

Bibliografía

[1] L. Araujo and C. Cervigón. *Algoritmos Evolutivos: Un enfoque práctico.* 2009.

[2] B. Baruque. Conceptos básicos de los algoritmos evolutivos.

[3] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. C. Muller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. Vanderplas, A. Joly, B. Holt, and G. Varoquaux. Api design for machine learning software: Experiences from the scikit-learn project, 2003. https://arxiv.org/pdf/1309.0238.pdf.

[4] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(Jul):2171–2175, 2012.

[5] J. Holland. *Adaption in Natural and Artificial Systems.* 1975.

[6] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[7] Wes McKinney. pandas: a foundational python library for data analysis and statistics. In Stéfan van der Walt and Jarrod Millman, editors, *Python for High Performance and Scientific Computing*, pages 1–9, 2011.

[8] K. Muelling, J. Kober, and J. Peters. Learning table tennis with a mixture of motor primitives, 2010. https://pdfs.semanticscholar.org/ffa8/9e2d70c7b12e42b12923ebc45a46fb7798a9.pdf.

[9] Stuart Russell and Peter Norvig. *Inteligencia Artificial: Un enfoque moderno.* Pearson, 2004.

[10] Wikipedia. Agente inteligente (inteligencia artificial) — wikipedia, la enciclopedia libre, 2016. "https://es.wikipedia.org/w/index.php?title=Agente_inteligente_(inteligencia_artificial)&oldid=90911327".

[11] "Wikipedia". "neurona — wikipedia, la enciclopedia libre", "2016". https://es.wikipedia.org/wiki/Neurona".

[12] "Wikipedia". "perceptrón — wikipedia, la enciclopedia libre", "2016". "https://es.wikipedia.org/wiki/Perceptr%C3%B3n".