Redes Neuronales
Memoria Grupal

4DVJU Concept Art: Personajes, Escenarios y Props

Por Alfredo Pérez Pastor y Pablo Rodríguez Zurro

Tabla de contenidos

Sumario

En este documento se tratará la memoria de la implementación de una red neuronal expuesta en clase para la asignatura Concept Art: Personajes, Escenarios y Props.

La intencionalidad del proyecto es crear una inteligencia artificial basada en procesamiento de los valores y pesos de las conexiones entre diferentes neuronas. La red tiene que ser apta de responder a acciones regidas por patrones y cierta impredecibilidad, a la vez que "evolucionar" mediante diferentes entrenamientos.

Introducción

Descripción del programa

En esta aplicación se ha desarrollado una implementación del juego *"Whac-A-Mole"* para ser controlada por un jugador humano o una inteligencia artificial basada en una red neuronal de tres capas, una de inputs, escondida y outputs.

Objetivos

- Implementación de una Inteligencia Artificial basada en algoritmos de Redes Neuronales con 3 capas
- Aplicación de la Red Neuronal en un juego que pueda ser manejado por un ser humano.
- Aplicar y guardar diferentes entrenamientos para tener Redes Neuronales con diferente grado de inteligencia.

Motivación

- Integrar una Red Neuronal en un juego simple como es "*Whack-A-Mole"*
- Aprendizaje de otra herramienta para añadir Inteligencias Artificiales en videojuegos.

Descripción técnica

Red Neuronal

Diseño

- ANN_Layer: Las capas contienen las Neuronas y los valores de estas (pesos, bias,..), las relaciones entre otras capas, las operaciones básicas, definición de funciones y derivadas de estas. Operaciones como los ajustes de pesos y obtención de errores para el BackPropagation, obtener valores para el FeedForward.
  También tiene un sistema de Debugging para poder ver el ajuste de pesos y la información de esta.

- ArtificialNeuralNetwork: Se encarga de gestionar las layers, y gestionar los entrenamientos, los Inputs y los Outputs.

- ANN_Controller: se encarga sobre todo de encapsular y simplificar acciones para los usuarios que quieran manejar la IA, ya que muchas funciones internas de la IA no deberían ser tocadas fuera del control general de la IA.

- TrainingDatabase: Simplemente guarda dos arrays, que contienen los Inputs, y los Outputs deseados para el entrenamiento.

Implementación

- ANN_Layer: Sobre todo separe todo lo posible las funciones, aunque fueran sencillas, para que fuera más entendible para las personas que lo van a leer, a pesar de que se hagan llamadas a funciones muy sencillas. Esto se hizo sobre todo con las funciones de la región ANN_Layer/MathFunctoins, ya que permite ver incluso ver que tipo de función se está utilizando dentro del algoritmo y cuando.

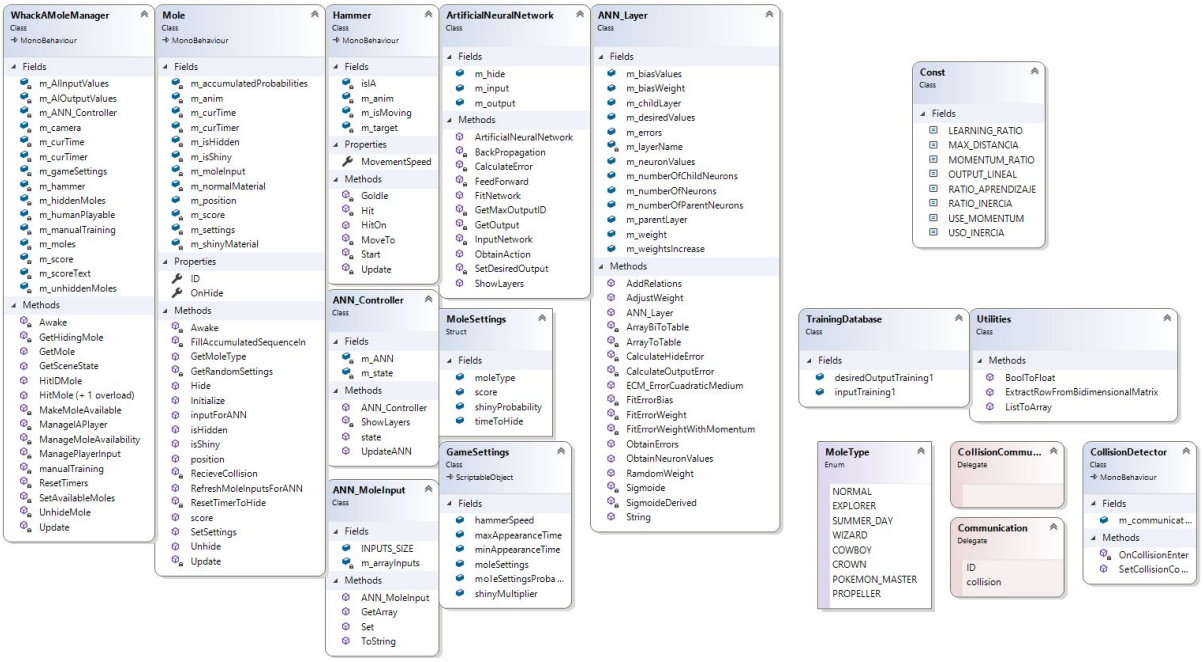
- ArtificialNeuralNetwork: Mucho más fácil, debido a que solo se encarga de gestionar capas, llamando a las funciones necesarias en caso de entrenar, de hacer BackPropagation o FeedFordward.

- ANN_Controller: Una simplificación de ArtificialNeuralNetwork para el usuario.

Observaciones

- La unica observacion que tengo, es que me hubiera gustado parametrizar las cantidad de capas, pero al no ser necesaria para esta práctica, y por el tiempo, no terminó implementando,

Diagrama de clases



- **WhackAMoleManager**

  Núcleo del programa, se encarga de gestionar la input de los jugadores, sean humanos o IAs, ordena y actualiza diferentes objetos, lógica del juego, guardar los topos y controlarlos, guardar el martillo y controlarlo, etc...

- **Mole**

  Clase encargada de gestionar animaciones, actualizar los inputs necesarios para la inteligencia artificial (extraída de sus miembros), seleccionar aleatoriamente a qué tipo de topo (con sus atributos) va a cambiar al salir del agujero y comunicar mediante un delegado si se esconde por recibir colisión o por finalización de su tiempo descubierto.

- **Hammer**

  Gestiona las animaciones del martillo y se mueve hasta un objetivo dado.

- **ANN_Controller**

  Crea una *ArtificialNeuralNetwork* para encapsularla y hacer de “interfaz” o bridge de algunos miembros que esta contiene de cara a implementación y gestión externa.

Explicación complementaria en el Diseño de la Red Neuronal.

- **ANN_Layer**

  Clase explicada en el Diseño de la Red Neuronal.

- **ArtificialNeuralNetwork**

  Clase explicada en el Diseño de la Red Neuronal.

- **CollisionDetector**

  Clase que guarda un delegado para comunicar colisiones con otros objetos.

- **GameSettings**

  ScriptableObject que guarda valores necesarios para el flujo de juego y diferentes ajustes.

- **TrainingDatabase**

  Archivo con propósito de guardar cadenas de texto con datos de entrenamientos anteriores.

- **Declarations**

  Archivo con clases de utilidades y declaración de estructuras y clases complementarias.

Memorias individuales

Pablo Rodríguez Zurro

Organización y tareas

- Diseño e implementación de las clases centrales necesarias para la gestión y desarrollo del gameflow.
- Diseño inicial de la red neuronal junto a Alfredo: Inputs necesarios, procesamiento de los outputs, modelo de datos, etc...
- Herramientas para uso fácil: ScriptableObjects, funciones de utilidades,etc...
- Asistencia a la implementación de la red neuronal e integración con el juego.

Errores y obstáculos

No ha habido demasiados errores por mi parte en el desarrollo de la práctica, se ha realizado con un ritmo constante y sin complicaciones.

Cooperación

Debido a la imposibilidad del desarrollo sincronizado ya que tenemos un repositorio basado en git, no hemos podido editar y revisar el código para ayudar al contrario en ciertas ocasiones.

Ha habido ciertos momentos en los que no nos hemos entendido, ya sea teniendo ideas iguales o contrarias, pero resueltas satisfactoriamente.

El desarrollo y publicación de las tareas ha sido excelente, solo hizo falta explicar la mecánica de publicación de commits en el repositorio al inicio y Alfredo se adaptó muy rápido.

Reflexión

Esta inteligencia artificial me ha costado más entender por su razonamiento matemático pero al fin y al cabo he entendido bien su funcionamiento y podría hacer mejores implementaciones (y más específicas) en un futuro.

Alfredo Pérez Pastor

Organización y tareas

- Descargar modelos 3D y adaptarlos para Unity e integrarlos
- Animaciones en Unity, máquina de estados y lógica de los scripts para controlarlas, junto con la ocultación y desocultación de los elementos.
- Logica Estetica de Mole.

- Lógica de movimiento de Hammer.
- Sistema de obtención de valores/casos y los Outputs esperados mientras juegas.
- Neural Network, Layer, gestión de Inputs y Outputs de datos con el juego, y tabla de entrenamiento.

Errores y obstáculos

Sin errores ni obstáculos aparentes, salvo la falta profunda de conocimiento de las IA, que tuvimos que estudiar concienzudamente.

Cooperación

En parte del trabajo hemos podido trabajar juntos, y hemos podido entender las tareas del otro, aunque en algunos momentos hemos trabajado por separado y hemos tenido que mirar el codigo del otro.

Reflexión

El concepto de las Redes Neuronales es muy interesante y aplicable a una infinidad de cosas, aunque para ello tienes que meterle un entrenamiento largo que puede tardar bastante en función de la cantidad de neuronas que tiene, y es más duro de lo que parece.

Me gustaria aprender mas sobre las diferencias que implica aumentar en número de capas ocultas y el número de neuronas que contiene cada una ya que para optimizar los resultados y el tiempo de la IA, seguramente sea necesario un número concreto de estas para cada aplicación. Ya que el aumento de neuronas y el entrenamiento puede aumentar de forma exponencial.