Universidad de La Laguna

PIPO

Program for the Investigation of Pursuing Objectives

(in different creatures)

Por:

Pamela Jiménez Rebenaque *alu0100894325@ull.edu.es*
Airam Manuel Navas Simón *alu0100618426@ull.edu.es*
Kevin Díaz Marrero *alu0100880625@ull.edu.es*
Jorge Sierra Acosta *alu0100896282@ull.edu.es*

Link al proyecto

Índice

Propuesta

Descripción

PIPO (Program for the Investigation of Pursuing Objectives in different creatures).

El proyecto se basa en la creación de un juego de realidad virtual controlado con un mando, donde el jugador deberá recorrer distintos escenarios (similares a laberintos), evadiendo trampas y enemigos, y recogiendo vidas y puntos.

En concreto, nos perseguirán distintas criaturas que contarán con dos sentidos para analizar su entorno: vista y oído y posteriormente se ha barajado la posibilidad de añadir el sentido del olfato.

Dispondrán, de varias redes neuronales que no serán creadas de forma arbitraria, sino a través de algoritmos genéticos (Ver NEAT, más adelante). En el caso de la vista, la red será entrenada para explorar el terreno evadiendo obstáculos y perseguirnos en caso de que nos vea. Por otro lado, en el caso del oído, las criaturas podrán escucharnos a través del micrófono, de forma que, si oyen ruido proveniente de cierta zona, se acerquen para comprobar si estamos ahí, utilizando en este caso otra red neuronal entrenada para moverse a un punto en el espacio 2D, ese punto es determinado de la siguiente forma:

Entre el punto emisor del sonido (el jugador) y el punto receptor del sonido (el propio enemigo) se buscará un camino utilizando A*, ese camino estará compuesto por una serie de nodos, se pedirá después a la red que se mueva hacia el primero de los nodos, de esta forma, si el sonido es repetitivo, el enemigo eventualmente alcanzará la posición de emisión, si el sonido fue puntual, el enemigo se moverá hacia el sonido, pero no tendrá más información.

Que red utilizar en cada momento será decidido mediante las variables externas, es decir:

- Se escucha un ruido: Utilizar la red que relacionada con el sistema auditivo.
- Se ve al jugador: Utilizar la red de visión para seguir.
- Otro caso: Utilizar la red de visión para explorar

De igual forma y al disponer de varios sentidos, se propone una diferenciación de criaturas, dotandolas de diferentes parámetros de sensibilidad para sus sentidos y/o eliminando alguno de ellos (En este caso, quedándose sólo con visto o con oido).

Este documento se centrará en el aspecto ya comentado del desarrollo de la inteligencia artifical para el juego.

Recursos utilizados

Los recursos utilizados en el proyecto son los siguientes:

- **Unity3D -** Para la creación de la parte gráfica del videojuego (escenarios, personajes, menús, etc). Además, se utilizan los recursos disponibles para Realidad Virtual. Se utiliza Unity3D por la posibilidad de implementar Realidad Virtual y por la facilidad para modelar e implementar modelos 3D que respondan a la conducta requerida mediante los scripts asociados.

- **Blender -** Para el modelado en 3D de los escenarios, objetos y personajes. Se utiliza Blender por la facilidad (comparado con Unity3D) para modelar formas complejas

.

- **Google VR SDK for Unity -** SDK de Google utilizado en Unity3D para la implementación de la Realidad Virtual. Se utiliza por su compatibilidad con un amplio abanico de dispositivos móviles.

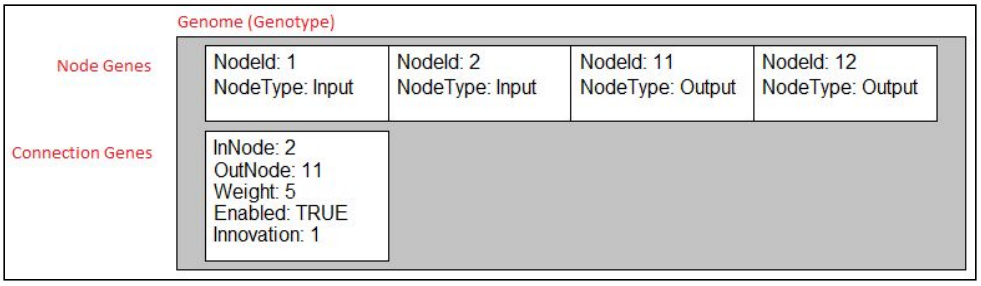
- **UnityNEAT -** Framework para la utilización del algoritmo NEAT sobre redes neuronales en Unity.

- **C# -** Para la creación de scripts así como para la programación de NEAT. Utilizamos C# por su compatibilidad con el entorno de Unity3D. su similaridad con C++ (lenguaje con el que el equipo cuenta con más experiencia) y por su flexibilidad.

Tecnologías de Inteligencia Artificial

NEAT - NeuroEvolution of Augmenting Topologies

El algoritmo NEAT es un algoritmo genético que se aplica sobre redes neuronales, y es el algoritmo que hemos utilizado para enseñar a los enemigos a explorar y perseguirnos.



En este algoritmo, la información de la red es representada por un genoma.

El genoma contiene genes nodo o neuronas y genes conexión o enlaces. Los genes nodo o neuronas pueden ser de input, output o estar ocultos. Los genes conexión o enlaces unen las neuronas y tienen un peso asociado a ellos, un flag que indica si el enlace está activo o no, y un número de innovación.

El algoritmo comienza con una piscina genética (*n* topologías de redes neuronales creadas de forma aleatoria).

A continuación, se calcula el valor objetivo de cada genoma de la piscina genética, aplicando dicha red neuronal al problema y obteniendo un valor de fitness.

Luego, se asigna cada genoma a una especie concreta. Los genomas pertenecientes a la misma especie tendrán características similares. De esta forma, en lugar de luchar para sobrevivir (no ser eliminados de la piscina genética) contra todos los demás, luchan contra su misma especie. Así, las especies que se forman de una innovación topológica que pueden no tener un valor objetivo alto debido a que sus pesos no ha sido optimizados, pueden sobrevivir.

Para cada especie, se eliminan los genomas más débiles (menor valor objetivo) y tras eso, se seleccionan genomas de cada especie para mutarlos o cruzarlos.

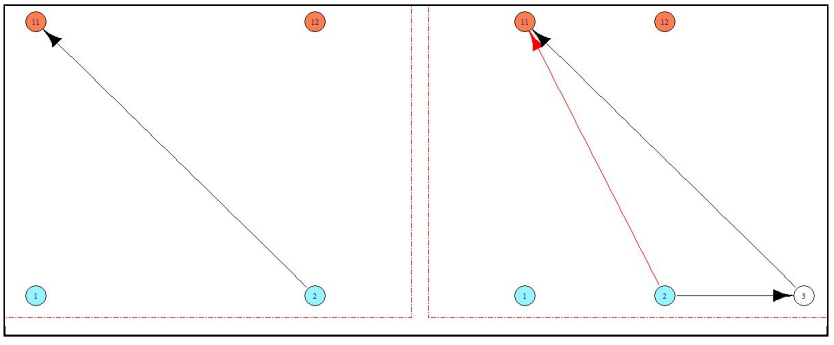
Se pueden dar diversas mutaciones. Cada genoma dispone de un índice de probabilidad para cada una de las posibles mutaciones, que puede ser incrementado o decrementado a lo largo del proceso. Las diversas mutaciones posibles son:

- **Mutación de peso:** se selecciona un enlace de forma aleatoria y se cambia su peso por un valor aleatorio.

- **Mutación de enlace:** añade un enlace nuevo entre dos nodos con un peso aleatorio.

- **Mutación de nodo:** añade un nuevo nodo a la red, sustituyendo un enlace entre dos nodos por un nodo intermedio que une con uno de los primeros nodos con un enlace de peso 1 y con el otro con un enlace del mismo peso que el enlace sustituido

.



- **Mutación de activación:** elige un enlace aleatoriamente y lo activa o desactiva.

Para mutar un genoma, se determina mediante el índice de probabilidad si se aplica o no esa mutación.

También se puede realizar un cruce entre genomas. Cada vez que se crea un gen mediante mutación (mutación de enlace o mutación de nodo), se incrementa un número de innovación global que es asignado a ese gen. Si dos genes tienen el mismo número de innovación deben representar la misma topología (aunque los pesos sean diferentes). De esta forma, se puede seguir el origen histórico de cada gen.

Al realizar el cruce entre dos genomas padres, se crea un genoma hijo, que tendrá los genes más fuertes de cada padre y cualquier innovación topológica encontrada. Para ello, se alinean los genes de ambos padres según su número de innovación. Para cada número de innovación, se coge el que tiene mejor valor objetivo (si son iguales, se elige uno al azar). Si uno de los padres tiene un gen con un número de innovación que no tiene el otro padre, se considera una innovación topológica y el hijo la hereda.

Los nuevos genomas son añadidos entonces a la piscina genética.

Repitiendo este proceso durante varias generaciones, se consiguen topologías de redes neuronales óptimas.

Desarrollo

Lista de tareas y cronograma

Cronograma

		30 Oct	6 Nov	13 Nov	20 Nov	27 Nov	4 Dic	11 Dic	18 Dic	25 Dic	1 Ene	8 Ene	15 Ene
1.1	Estudio y diseño de los personajes iniciales												
1.2	Diseño de otro tipo de personajes												
2	Estudio del rigging y control del personaje												
3	Estudiar las posibles redes neuronales y algoritmos genéticos a utilizar.												
4	Aplicación y entrenamiento de las NN y/o AG a los personajes												
5	Aplicar algoritmos de búsqueda a los movimientos												
6	Colocar a los personajes en el escenario (A persigue a B)												
7	Estudio del movimiento de un personaje (controlado por el jugador) en el escenario												
8	Colocar a un personaje controlado por el jugador												
9	Diseño de mapas y obstáculos												
10	Preparación de la exposición												
11	Opcional: Entrenar red neuronal para toma de decisiones de movimiento												
12													
13													
14													
15													
16													
17	(Tiempo de margen)												

Recursos

		Unity	Blender	Google Drive & Github	Google Slides	Redes Neuronales	Algoritmos Genéticos	Algoritmos de Búsqueda	Dispositivo VR	Airam	Jorge	Kevin	Pamela
1.1	Estudio y diseño de los personajes iniciales												
1.2	Diseño de otro tipo de personajes												
2	Estudio del rigging y control del personaje												
3	Estudiar las posibles redes neuronales y algoritmos genéticos a utilizar.												
4	Aplicación y entrenamiento de las NN y/o AG a los personajes												
5	Aplicar algoritmos de búsqueda a los movimientos												
6	Colocar a los personajes en el escenario (A persigue a B)												
7	Estudio del movimiento de un personaje (controlado por el jugador) en el escenario												
8	Colocar a un personaje controlado por el jugador												
9	Diseño de mapas y obstáculos												
10	Preparación de la exposición												
11	Opcional: Entrenar red neuronal para toma de decisiones de movimiento												

Proceso y funcionamiento

Proceso de entrenamiento

Como ya se ha descrito con anterioridad, se utilizarán diferentes redes que serán entrenadas utilizando el algoritmo NEAT. El proceso de entrenamiento se ha divido en las redes que responderán al sistema visual y en aquellas redes que responderán al sistema auditivo.

Para facilitar el entrenamiento, se han construido escenarios simples adecuados a cada una de las necesidades y los enemigos se han simplificado con un *cubo* con capacidades visuales y/o auditivas.

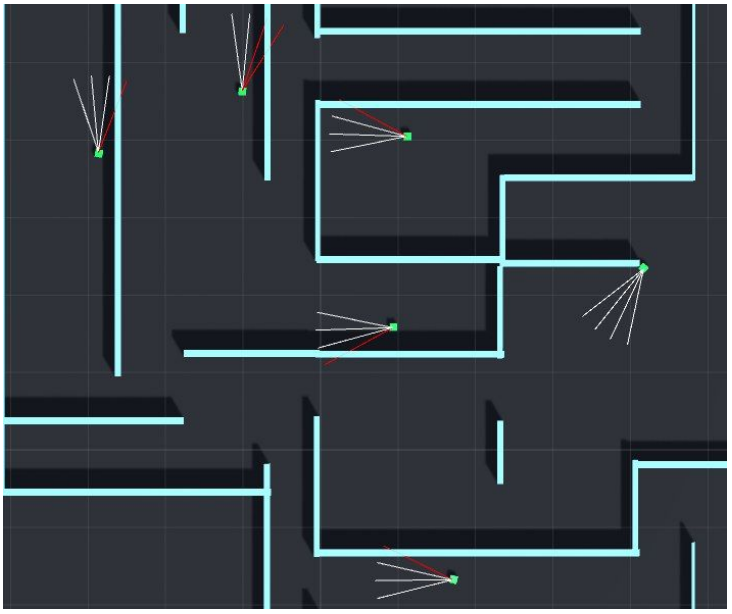
Sistema visual

Para el sistema visual se ha diseñado un pequeño laberinto y se ha dotado a cada *cubo* de un sensor visual. Los sensores visuales están formados por un pequeño punto a partir del cual se lanzan varios rayos (Raycasting) a una distancia máxima determinada en forma de cono. Cada rayo contiene la siguiente información.

1. Distancia a la que el rayo detectó un objeto
2. Tipo de objeto que ha detectado el rayo.

Utilizando esta información, se le ha dotado por defecto a cada sensor visual de un número de cuatro rayos, en un cono con una abertura total de aproximadamente 45 grados. En consecuencia se decidió dotar a la red de las siguientes entradas:

- Por cada rayo
  - Distancia a la que se detectó un objeto normalizada entre [0, 1], siendo 1 la distancia máxima del sensor y 0 tener el objeto detectado inmediatamente delante. Cuando no se detecta ningún objeto, también se devuelve el valor de 1.
  - Valor booleano que indica si se ha detectado un muro y obstáculo
  - Valor booleano que indica si se ha detectado un jugador



Cada punto verde representa una criatura, los cuatro rayos se pueden ver con facilidad

Se escogió como función objetivo para el algoritmo genético manejado por NEAT una función que describe la cantidad de casillas únicas por las que el *cubo* ha pasado. Con el objeto de entrenar la red primero con el objetivo de que realizara una exploración adecuada. Sin embargo, los resultados fueron bastante propensos a mostrar un comportamiento que se basaba en moverse de forma paralela al muro derecho, de esta forma el conjunto de casillas únicas que alcanzaba era bastante alto, pero no así mostraba un comportamiento adecuado para un "individuo que explora una zona", ya que muchas veces dejaba atrás salas completas y/o evitaba siempre explorar el centro de las salas más grandes.

Esto se resolvió añadiendo dos nuevas entradas a la red neuronal. Estas entradas son:

- Un número entero aleatorio, entre 0 y 1 (Inclusive).
- Un booleano que representa si la casilla actual ya ha sido explorada con anterioridad.

Después de un tiempo de entrenamiento moderado, y utilizando la misma función objetivo, los *cubos* mostraban un comportamiento ejemplar a la hora de la exploración, y aún mejor, no repitiendo el mismo patrón siempre que se iniciaba el algoritmo.

Para realizar un entrenamiento ya no sólo con respecto a la exploración exitosa, sino también con el objetivo de perseguir a nuestro jugador una vez lo encuentren, se ha

desarrollado otra función objetivo cuyo comportamiento es exactamente igual que el de la anterior pero sólo hasta que el *cubo* haya completado una exploración de las primeras x casillas únicas (x varía en valor entre las 200 y 400 según la prueba). Una vez el *cubo* realice rápidamente exploraciones eficaces, la función objetivo incluirá también el tiempo total que el propio *cubo* es capaz de seguir a un jugador. Este "jugador" ha sido simulado utilizando la propia red de exploración, es decir: **Una vez se completó el entrenamiento de explorador, se crearon simulaciones de jugadores reales con esos exploradores y por otro lado se crearon nuevos objetivos para evolucionar.**

Después de otro tiempo razonable de entrenamiento se crearon redes neuronales que satisfacen de forma muy adecuada los requisitos y simulaban con creces el comportamiento de un animal o insecto explorando un área.

Sistema auditivo

El sistema auditivo sufrió complicaciones debido a problemas con la temporización del proyecto y no llegó a ser implementado de forma completa. Sin embargo, si se entrenó con éxito una red neuronal encargada de (y con ayuda de los sensores de visión ya explicados), moverse del punto A al punto B.

La premisa del sistema auditivo utilizando esa red es la siguiente:

- Detectar un sonido
- Mediante A* es calculado el camino entre el emisor y el receptor de ese sonido. Este apartado también llegó a ser completado satisfactoriamente.
- Mover el *cubo* hacia el primer nodo del camino encontrado.

Problemas encontrados

- **Idea inicial: enseñar a caminar:**
  - **Físicas de Unity3D.**

La idea inicialmente era diseñar unos modelos de personajes los cuales estarían dotados de un esqueleto con el cual mediante las físicas y fuerzas aplicadas en cada articulación, poder moverse en el entorno de Unity 3D y, mediante el uso de las redes neuronales (UnityNEAT), poder llegar a enseñar a dichos personajes a caminar.

El problema surgió cuando quisimos simular as "físicas" de un personaje que se quiere "arrastrar" por un plano, pues al no existir ningún tipo de fuerza de rozamiento entre los colliders de las patas del personaje y el plano, el personaje "patinaba" sobre sí mismo en vez de intentar avanzar. Tras muchos intentos de arreglar dicho problema, acabamos dándonos cuenta que con el motor de Unity3D que nos ofrece, no era posible realizar tal tarea y decidimos cambiar el objetivo de la enseñanza a los personajes, pasando de enseñarles a caminar, a enseñarles a explorar un entorno desconocido mediante el uso de la vista, oído y olfato (simulados).

➢ **Grados de libertad para red neuronal.**

Es cierto también que llegamos a conseguir arreglar de cierta manera el problema de las físicas en Unity3D, pero ello requería que el personaje en cada articulación pudiese ejercer una fuerza aleatoria, en una articulación aleatoria, en un eje de rotación aleatorio. Lo cual, ofrecía un grado de libertad a la hora del aprendizaje demasiado extensa, lo cual podría hacer que el personaje llegara a aprender a caminar pero tardando demasiado tiempo en la fase de entrenamiento.

❖ **Combinación de algoritmos de pathfinding y RN.**

Tras comprobar que la red neuronal funcionaba a la perfección a la hora de enseñarles a los personajes a explorar y perseguir a su objetivo con el sentido de la vista, decidimos intentar que hiciera algo similar con el oído mediante las RN, pero nos acabamos percatando que se volvía demasiado complejo pues estaba muy limitado y decidimos hacerlo mediante el algoritmo A*. El proceso era que cuando el objetivo realizara cualquier tipo de ruido, el personaje trazaba el camino dado por el A* pero éste sólo avanzaría hasta el primer nodo del camino, para así no volver la dificultad demasiado grande al objetivo.

❖ **Problemas de colaboración en Unity.**

Uno de los mayores problemas que tuvimos a la hora de trabajar en grupo con Unity3D, es que la versión para poder colaborar en un mismo proyecto era de pago, y era bastante pesado el tener que estar comprimiendo el proyecto continuamente, recompilando el mismo, etc.

❖ **Retrocompatibilidad de versiones Unity.**

A la hora de pasarle el proyecto a otro miembro del grupo de trabajo para que realizara cambios, por las distintas versiones que cada uno podría tener, para que dicho miembro pudiese abrir el proyecto, primero se tendría que transformar las versiones para poder ejecutar el proyecto, y la demora se llegaba a extender por 20 minutos lo que hacía muy difícil la colaboración en grupo en el mismo proyecto.

❖ **Tamaño del proyecto para Github.**

El proyecto final acabó ocupando unos 3.5 GB, y en Github el límite de subida está en los 2GB, por lo tanto tampoco podíamos llegar a compartir el código por dicha plataforma.

Conclusiones

Las conclusiones del proyecto pueden ser resumida en el conjunto de hitos conseguidos, sin embargo cabe destacar también la premisa de que después de realizar el proyecto se ha

demostrado que los algoritmos genéticos ofrecen soluciones sencillas de implementar con respecto a las redes neuronales que requieren de un diseño de su topología, pero como contrapunto requieren de una gran cantidad de tiempo (y generaciones) para producir buenos resultados.

La combinación de ambos, ofrece una alternativa bastante eficaz como se puede ver en los resultados de la IA utilizada en el juego.

Es importante destacar además que se ha observado como las variaciones de la cantidad de especies e individuos por especie afectan a los resultados. Las soluciones que se prevén sencillas se benefician de muchos individuos con pocas especies, generando una mayor capacidad de variación. Sin embargo, las soluciones más complicadas se desarrollan mejor alcanzando un equilibrio donde hay menor número de individuos por especie, donde diferentes especies exploran una mayor variedad de caminos que puedan llevar a una solución.

Hitos conseguidos

- Diseño de tres escenarios propios (granero, hospital y nave) con menús (inicio, fin de partida y.pantalla de carga).
  - Sonidos ambientales (música y personajes) y efectos.
  - Utilización de sistemas de partículas (fuego, puntos y vidas).
- Diseño de personajes y animaciones.
- Movimiento del personaje para exploración y persecución mediante la red neuronal.
- Sensor de vista y oído.
- Utilización de sensores de Android.
  - Acelerómetro: el personaje que controlamos se agacha si la persona jugando también lo hace, reduciendo su velocidad y sonido emitido.
  - Micrófono: se detecta el sonido por micrófono y su rango en el escenario.
- Eventos implementados mediante controlador.
  - Recogida de vidas y puntos.
  - Activación de trampas.
  - Muerte del jugador.
  - Jugador atrapado por enemigo.
- Realidad Virtual en Google Cardboard.
  - Selección de opción mediante retícula.
  - Control de cambio de luces.
  - Control de disparidad en la percepción entre el mundo real y el juego.

Bibliografía

Google VR SDK:
https://github.com/googlevr/gvr-unity-sdk/releases

UnityNEAT:
https://github.com/lordjesus/UnityNEAT

NEAT:
http://gekkoquant.com/2016/03/13/evolving-neural-networks-through-augmenting-topologies-part-1-of-4/
https://youtu.be/qv6UVOQ0F44
http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf