Grado en Ingeniería Informática ULL

Inteligencia Artificial Avanzada

Inteligencia Artificial en Juegos

Manuel Andrés Carrera Galafate
Aarón José Cabrera Martín
Rafael Cala González

Grupo G4

ÍNDICE

Introducción

En este trabajo comenzaremos por explicar distintos aspectos de la IA en los videojuegos, como son el contexto histórico, qué es una IA, cómo ha evolucionado con el paso del tiempo... A continuación hablaremos de la aplicación de redes bayesianas en este ámbito y de sus ventajas sobre otro tipo de modelos para representar IIAA.

En último lugar, modelizaremos una red bayesiana simple para un bot sencillo y resumiremos el trabajo realizado.

Introducción a la IA en juegos.

Hoy en día los videojuegos están presentes en el día a día en una parte importante de las población, incluso siendo independiente de la edad, sexo, etc. La complejidad de los videojuegos puede variar desde pocas líneas de código a un proyecto multimillonario, pero estos no habrían tenido la expansión actual si no hubiese sido por un factor determinante: la introducción de la inteligencia artificial en ellos.

Historia y contexto

El origen de la IA en videojuegos se remonta a mediados siglo pasado (1959) por Arthur Samuel en su libro "Some studies in machine learning using the game of checkers" con un simple juego de damas en el que veríamos por primera vez uso de algo de inteligencia artificial. Esto serviría de base para otros juegos en el futuro que tendrían IIAA más desarrolladas en videojuegos como Pac-man en el que cada enemigo tiene una personalidad y patrón de movimientos único, o Half-life, donde los enemigos son capaces de apoyarse mutuamente... Gradualmente pasaríamos de tener videojuegos que implementarían la inteligencia como un simple patrón de movimientos para los enemigos, visibles en juegos como Donkey Kong, Mario Bros, etcétera a videojuegos donde realmente el software se sentiría como un ente inteligente.

Las ventajas que ofrecían las IIAA a los videojuegos pronto ganaron terreno sobre juegos basados en simples patrones, y con el desarrollo de los microprocesadores pronto se convirtieron en un elemento esencial en la mayoría del mercado ya que el usuario promedio ya no se contentaría con IIAA tan básicas tras la existencia de sistemas más sofisticados que brindarían experiencias de juego mejores que las de sus antecesores.

Hoy en día la IIAA es una parte imprescindible en un videojuego que quiera consolidarse como un producto exitoso o interesante para un sector de la población. Tenemos ejemplos de grandes IIAA implementadas en videojuegos como el GTA V, donde cada ciudadano del mundo tiene independencia propia, emociones, acciones en base al entorno... Juegos como el Far Cry 3 donde incluso los animales tienen comportamientos similares a los de seres vivos reales, o los enemigos pueden experimentar miedo, recuerdan la localización del jugador y atacan en base a esta información (y no simplemente la IA trabaja con la información de saber dónde realmente está el jugador), etc...

Definición de IA en videojuegos.

La definición formal de IA se basa en el comportamiento y pensamiento de manera racional o humana. Es decir, podemos decir que un sistema es inteligente si bien se comporta de manera racional o humana (comportarse humanamente no siempre es racional) o si actúa de cualquiera de estas 2 maneras. Uno de los mecanismos para decidir si un sistema es inteligente o no se basa en el Test de Turing, que postula que si a un juez lo ponemos frente a un sistema de IA y una persona, le hacemos realizar preguntas a estos dos donde estos tienen la opción de mentir o decir la verdad y el juez no es capaz de diferenciar al humano de la IA, podemos concluir que el sistema es inteligente.

Esta definición formal es útil y es buena base para trabajar, pero no funciona exactamente igual en el mundo de los videojuegos. En un videojuego, el Test de Turing se sigue aplicando, pero ahora el juez será el jugador y se considerará que el test se pasa o no en base a la experiencia del jugador. La clave está en encontrar el equilibrio donde el sistema no se sienta obtuso o carente de cualquier tipo de inteligencia, pero que tampoco se comporte de manera perfecta, como si "fuera una máquina", sino encontrar el balance justo donde parezca que el rival sea realmente una persona.

Aspectos de la implementación

Implementar un buen diseño de IA para un videojuego puede tener una solución aún más compleja de lo que pueda parecer en un principio, porque no se trata simplemente de buscar un comportamiento o gameplay perfecto por parte de la máquina, o una forma de actuar que sea capaz de ganar siempre a los humanos, sino de hacer que esta se sienta como un reto para el jugador (A menos que estemos buscando un ejemplo académico en el que queramos una IA que juegue de manera perfecta o que se trate de un reto en sí conseguir este juego perfecto, en casos como los del ajedrez o la IA que venció al campeón del mundo del Go).

La clave está en que este reto debe ser posible superarlo, con mayor o menor dificultad para que sea "divertido". Para lograr eso hay que conseguir que actúe como haría un humano. Es decir, puede cometer errores, reaccionar en un tiempo factible para un humano y no contar con más información de la que a priori un jugador podría conseguir

sin usar "cheats". Para lo previamente mencionado, es importante para mejorar la experiencia del jugador, ya que dar información del sistema a la IA no se siente justo, humano ni satisfactorio. Un ejemplo podría ser algún juego de estrategia en tiempo real en el que la IA tiene información de dónde hay recursos ocultos para la visión de un jugador convencional, pero que la máquina por ser parte del sistema tiene el privilegio de conocer la información. Esto sería lo que denominamos "dificultad artificial".

Para lograr nuestro objetivo tendremos que tener en cuenta varios aspectos, como son el movimiento NPC's, el dinamismo del sistema, el objetivo y estrategia (que no será igual para todos los NPC's, ya que habrá algunos que directamente no tengan objetivos, como transeúntes en un juego de mundo abierto, y habrá otros que directamente quieran acabar con el jugador, como podría ser un jefe final), sistemas de diálogos, comportamientos alineados con las personalidades y sentimientos del sistema, etc... Incluso deberemos tener en cuenta la dificultad variable de un videojuego, durante esta variabilidad puede representar un cambio en un simple parámetro, como pueda ser la vida de los enemigos, hasta una nueva inteligencia artificial completa para un enemigo determinado.

Otro aspecto a considerar es que el fin de la inclusión de la IA en el videojuego no va de conseguir el "mejor código" ni la mayor optimización de este, sino de lograr un código funcional que se asemeje en la medida de lo posible a los esperado por el jugador. Nunca hay que olvidar que la inteligencia del sistema es un aspecto que complementa y ayuda al juego a funcionar, pero el fin último es la experiencia del jugador. En definitiva que el conjunto del sistema se sienta orgánico en la tarea que debe realizar por simple que sea, ya que un fallo en estas puede ocasionar una pérdida importante en la experiencia del jugador.

En conclusión, a la hora de implementar una IA hay que tener en cuenta dos aspectos fundamentales: el comportamiento de nuestra IA debe asemejarse en la medida de lo posible al de un humano que esté jugando y el hecho de que toda la implementación debe basarse en la experiencia del jugador. Sin estos aspectos considerados, el sistema de IA en el videojuego está condenado al fracaso.

Aplicaciones de las redes bayesianas en juegos

Con el objetivo de conseguir una solución eficiente para lograr el comportamiento de un bot en un videojuego se propone un sistema basado en redes bayesianas.

Ventajas

- La ventaja clara para los programadores es que es mucho más fácil de leer e implementar basado en preguntas tipo "si estoy atacando, ¿cuánta vida me queda?" frente a por ejemplo un típico script lleno de if else. Este aspecto ayuda a la hora de corregir bugs y a la hora de trabajar en equipo.

- Se basa en un modelo fuertemente matemático y probabilístico.
- La complejidad a la hora de inferir comportamientos está prácticamente en lineal tanto en el número de variables como en el número de posibles estados.
- A la hora de añadir nuevas variables al sistema simplemente consiste en añadir una nueva tabla.

- El sistema se vuelve fácil de analizar y controlar el comportamiento para el diseñador. Es una manera muy natural de formalizar el comportamiento esperado del bot, para cambiarlo sólo hay que modificar entradas concretas en la tabla.

- No hacen falta grandes conocimientos de programación ni de probabilidad para implementar el modelo debido a la propia naturaleza de la red.

Desventajas

- Puede resultar complejo cuantificar la probabilidad de las decisiones de los jugadores humanos para poder rellenar las tablas con criterio y exactitud.

- Dependencia del diseño que se realice a alto nivel, porque si se eligen mal las variables de interés o las dependencias entre ellas podemos obtener comportamientos no deseados o incluso complicar demasiado el modelo.

Modelización del bot

En nuestro caso hemos modelizado el comportamiento del bot de la siguiente forma:

Como se puede observar las dependencias de las variables en nuestro modelo son todas del tipo 1 a 1 donde la mayoría de variables pueden tomar solo 2 valores, lo cual crea varias tablas de tamaño (n variables de la tabla 1 * m variables de la tabla 2) y si quisiéramos añadir una nueva variable al modelo tan solo tendríamos que crear una tabla extra y rellenar los valores. Sin embargo, si la aproximación fuese en un modelo donde todas las variables menos 1 infieren la restante, tendríamos una tabla que tendría tantos valores como el producto de cada una de las tablas por el número de variables que puede tener. Esto hace que este hipotético modelo fuese inmanejable, y más teniendo en cuenta que podríamos añadir nuevas variables en el futuro. Por ejemplo, si tenemos una tabla con 100 entradas, meter una variable nueva que pueda tomar 2 valores implicaría que pasariamos a tener 200 entradas.

Segunda parte del informe.

En esta segunda parte del trabajo hemos utilizado la API de Nética en C y C++ para transcribir nuestra red diseñada en la primera semana a código nuestro sin usar la aplicación de nética directamente.
El primer paso que hemos seguido ha sido instalar las librerías, descargándolas de la página oficial de Nética y hemos modificado el Makefile (fichero Compile.sh) para poder hacerlo funcionar (hay cierto problema con las dependencias originales del script). Además, la versión de libnetica.a que venía por defecto era la de 32 bits, por lo que fue necesario sustituirla por la de 64 bits.

Tras arreglar las dependencias del compile.sh y cambiar la versión de libnetica, procedimos a arrancar el script. Una vez arrancado se generó un código de ejemplo que ejecutamos y funcionaba correctamente.

El siguiente paso consistió en tomar ese código como base y convertir la red de ejemplo en la nuestra, cambiando los nodos de la red, las dependencias entre las variables, añadimos los atributos necesarios, etc...

Una vez transcribimos nuestra red, ejecutamos el programa y comprobamos con éxito que los resultados fueran iguales a los de la red modelizada en la aplicación de Nética.

El último paso consistió en ampliar el código para darle interacción con el usuario, para que este pueda introducir los datos manualmente, o, si simplemente quiere probar el programa también tiene un modo de prueba.

Ejemplo (Vida baja, enemigo cerca, y St = atacar):

ll

Porcentajes de trabajo

Nombres	Porcentajes	Influencia en el trabajo
Manuel Andrés Carrera Galafate	33.3%	Redacción principal del informe.
Aarón José Cabrera Martín	33.3%	Desarrollo del modo interactivo en el código.
Rafael Cala González	33.3%	Mayor parte del desarrollo del código base para modelizar la red.

El desarrollo de esta práctica se ha dado de manera similar al de la anterior, nos hemos reunido los tres integrantes del grupo en dos sesiones cortas telemáticas para hacer la práctica, donde el primer día discutimos cómo enfocar la práctica y desarrollamos el código y en la segunda nos hemos enfocado más en el desarrollo del informe.

Webgrafía

Netica: https://www.norsys.com/netica.html

Errores varios durante el desarrollo: https://es.stackoverflow.com/