ESAT INNOVATION SCHOOL | ESCUELA SUPERIOR DE ARTE Y TECNOLOGÍA

Inteligencia Artificial en Videojuegos

Por: Lucas Merenciano Martínez

Nombre Asignatura: Inteligencia Artificial
Modulo Asociado: Unit 27: Artificial Intelligence (L5)
Profesor: Antonio Barella
Curso: 2019/2020
Autor: Lucas Merenciano Martínez

Índice

Inteligencia artificial en videojuegos

Este documento está centrado en inteligencia artificial para videojuegos, la cual tiene algunas diferencias respecto a la inteligencia artificial empleada en otros campos, siendo la principal el hecho de que un videojuego debe de ser entretenido, por lo tanto si utilizamos inteligencia artificial en los enemigos, no queremos maximizar su eficiencia, o acabaríamos teniendo un videojuego imposible de completar por un humano.
Otra diferencia viene marcada por las limitaciones de tiempo que presenta cualquier videojuego: no podemos emplear técnicas que requieran de muchos ciclos de procesador para llevar a cabo sus cálculos.

Es por este motivo que muchas de las técnicas que serán nombradas en este documento tienen como objetivo entretener al jugador utilizando el menor número de cálculos posible.

Cambios en la IA en la industria del videojuego

Obviando algoritmos como MiniMax en videojuegos de ajedrez y similares, la inteligencia artificial comenzó en forma de algoritmos básicos que permitían dar comportamiento a un elemento del videojuego, sin necesidad de que ningún humano lo controlase. Uno de los primeros ejemplos lo tenemos en el Pong, donde la raqueta rival se movía sola teniendo en cuenta la posición de la pelota.
Este tipo de comportamientos se fueron haciendo más complejos, pero sin dejar de ser movimientos o acciones básicas en relación a ciertas variables del juego, un claro ejemplo son los fantasmas del PacMan.
Este tipo de inteligencia artificial es la más común en videojuegos, y a día de hoy se sigue utilizando (en conjunto con nuevas técnicas), ya que bien utilizada, permite crear comportamientos “humanos” con costes de computación extremadamente bajos.

Un poco más adelante surgieron videojuegos como el Rogue o Elite, que generaban los niveles utilizando algoritmos (simulando la función de un Game Master).

Pero con el paso de los años, al igual que la generación de niveles ha evolucionado y ya no solo se basa en aleatoriedad con parámetros, sino que son usados una gran cantidad de algoritmos para crear mundos infinitos, también ha evolucionado el comportamiento de los enemigos, dando paso a complejos árboles de comportamiento con gran variedad de tareas y algoritmos, por ejemplo para la búsqueda de caminos o dificultad adaptativa.

Este comportamiento realista se ha extendido también a los elementos del entorno, como los npc secundarios o animales y pájaros. Normalmente son dotados de una rutina, empleando para ello sistemas como las máquinas de estado finitas y algoritmos como el “flocking”.

En la actualidad se utilizan técnicas de IA de formas nunca vistas en los años anteriores, por ejemplo para la remasterización de videojuegos antiguos o incluso el propio diseño completo del videojuego. Ejemplos de videojuegos creados por una IA: Games by ANGELINA [1].

Optimizando A* con niveles de granularidad

El algoritmo de búsqueda de caminos A* en sí tiene poco margen de optimización, por lo tanto las optimizaciones deben hacerse a nivel de optimización.
Por ejemplo sustituyendo las listas por “heaps”, pre-calcular caminos que sabemos que serán utilizados, o el tema que voy a tratar: mapas a diferente granularidad.

Durante la búsqueda de un camino, se tienen en cuenta un gran número de posibles posiciones intermedias entre el inicio y el final del trayecto, muchas más de las que finalmente se utilizan, y ese número crece exponencialmente junto al tamaño del mapa.
Por ese motivo, una gran optimización consiste en tener dos versiones del mapa, la que se muestra en pantalla y la que se utiliza para extraer su información y utilizarla en el algoritmo. De esta manera, el mapa es el mismo, pero el algoritmo hace sus cálculos teniendo en cuenta muchas menos posiciones, ya que hemos aumentado su granularidad.
Con esta optimización se aumenta bastante el rendimiento y el camino encontrado es válido, pero prestando atención, se ve claramente que la granularidad del camino encontrado no se corresponde con el mapa que vemos en pantalla. Este es un problema que puede aceptarse sin más, pero tiene una solución: ir reduciendo la granularidad del primer camino encontrado.

Para ello, nuestra estructura LevelMap dispondrá de dos niveles de granularidad: se calculará el camino primero utilizando la granularidad más alta, lo que nos dará un camino bastante tosco, pero empleando poco tiempo de computación. Una vez tenemos el camino tosco, volvemos a calcular el camino utilizando la granularidad baja, pero esta vez sólo tendremos en cuenta las posiciones contenidas en el camino descrito por el primer cálculo, o contemplando también los bordes del camino, si se permite el movimiento en diagonal.

De esta manera, el camino obtenido responde a una granularidad mucho más baja, con un coste mucho menor al de haber calculado el camino a esa granularidad desde el principio.

Pero ahora que hemos aplicado esta optimización, podemos volverla a aplicar creando un nivel intermedio de granularidad entre la inicial y la final, de esta manera:

1. Se calcula el camino utilizando la granularidad más alta.
2. Se calcula el camino utilizando la granularidad intermedia, teniendo en cuenta únicamente las posiciones contenidas en el camino obtenido en la granularidad alta, añadiendo los bordes para los avances en diagonal.
3. Se calcula el camino utilizando la granularidad más “fina”, teniendo en cuenta únicamente las posiciones contenidas en el camino obtenido en la granularidad intermedia, añadiendo los bordes para los avances en diagonal.

Como es lógico, de esta manera se reducen enormemente las posibles posiciones del camino en la granularidad más baja, reduciendo así el tiempo de computación requerido.
Esta optimización también nos permite varias mejoras, ya que una vez calculado el camino a granularidad alta, podemos ir calculando el camino a granularidad más baja por segmentos, mientras la entidad recorre el camino, o incluso no calcular la granularidad más baja si el equipo está teniendo problemas para mantener el “frame rate”, siempre que nuestra aplicación utilice la técnica del bucle de simulación.

Desventajas

El cálculo de caminos a diferente granularidad puede dar lugar a fallos de precisión, ya que al calcular el primer camino a granularidad alta, es fácil que se pasen por alto posibles caminos por zonas estrechas que no se perciben a granularidad alta.
Si queremos evitar esto a toda costa para aprovechar cualquier hueco por el que pueda caber la entidad, deberemos tener en cuenta su tamaño a la hora de escoger la granularidad de partida, de esta manera jamás tendremos la sensación de que un ratón está recorriendo un camino pensado para un orco enorme.

Diseño de un sistema de flocking

La técnica conocida como “flocking” es utilizada para dar comportamiento de bandada o manada a un grupo de entidades, normalmente aves, animales terrestres o patrullas.
Para crear este comportamiento, debemos de asegurar tres conceptos: separación, cohesión y alineamiento.

Para empezar a diseñar el sistema, es necesario pensar en una clase llamada Flock, la cual tendrá referencias a cada una de las entidades de la “bandada” y calculará el vector “velocity” de cada una de ellas. Tendrá además una posición actual de la bandada en general y una posición objetivo.

Para tener un comportamiento que cumpla con los tres principios del flocking (separación, cohesión y alineamiento) sin invertir mucho tiempo, podemos abusar del sistema de colisión del motor de físicas creando para cada entidad un collider que la envuelva cuyo tamaño variará dependiendo de la distancia mínima que determinemos, de esta manera se cumple con la separación. La cohesión y alineamiento vienen solos si para calcular el vector “velocity” de la entidad nos fijamos en si está cerca del resto de la bandada. En caso de estar lejos, la posición objetivo de la entidad sería la posición global de la bandada, y si por el contrario ya se encuentra dentro del grupo, la posición objetivo será la marcada como objetivo en la clase Flock. De esta manera, aseguramos el alineamiento ya que todo el grupo persigue la misma posición a la misma velocidad, y aseguramos la cohesión porque cuando una entidad está lejos del grupo, su objetivo principal pasa a ser la posición del grupo y se aumenta su velocidad (speed) para que se incorpore pronto.

Aunque esta implementación es fácil si disponemos de motor de físicas, no es la más adecuada ya que podría notarse que la separación se debe a colisiones invisibles y no a la voluntad de la entidad. Para solucionar esto, calcularemos la dirección de forma distinta.

Lo primero será asegurar la cohesión de igual forma que en la implementación anterior, fijando como objetivo la posición del grupo y aumentando la velocidad si la entidad está lejos.
El alineamiento nos lo proporciona el hecho de que todo el grupo está unido y persigue el mismo objetivo.

La separación la conseguiremos realizando una última comprobación y ajuste en la dirección de la entidad: se comprobará para cada una de las entidades del grupo si la dirección actual causará que se acerque demasiado a otra entidad, y en caso afirmativo, se desviará la dirección lo suficiente como para que no se acerque demasiado a ninguna otra. En el caso de que no se pueda evitar acercarse demasiado a otra entidad sin realizar un cambio de dirección demasiado brusco, la velocidad de esta entidad disminuirá temporalmente.

Con este diseño aumentará el realismo, ya que será visible que cada entidad realiza pequeños cambios en su dirección y velocidad con el fin de mantener la separación.

```
1  Flock::Update
2  {
3    foreach Entity in Flock
4    {
5      if Distance(Entity.position, Flock.position) < Flock.group_radius
6        Entity.velocity = Flock.target - Entity.position
7        Entity.velocity.normalize()
8      else
9        Entity.velocity = Flock.position - Entity.position
10       Entity.velocity.normalize()
11
12     Flock.checkSeparation(Entity)
13   }
14 }
```

Implicaciones del uso de machine learning en el diseño de videojuegos

El uso del machine learning en el ámbito del diseño de videojuegos sin duda dará lugar a herramientas muy potentes para los diseñadores, pero esas herramientas no se crean solas. Es muy probable que las empresas grandes contraten programadores específicos para esta tarea, con experiencia en machine learning y big data. Las empresas también tendrán que conseguir acceso a todos esos datos y sobretodo, obtener y almacenar datos del propio juego en desarrollo para ser utilizados para ir mejorando el diseño.
Por parte de los diseñadores, no creo ni que se reduzca el número de ellos ni que aumente, ya que por una parte estas herramientas van a hacer más efectivo su trabajo, pero por otra parte serán indispensables para poder utilizarlas.

Desde el punto de vista de los jugadores, creo que no habrá una diferencia notable, pero el juego gustará más sin saber exactamente por qué. Ya que estas herramientas habrán permitido a los diseñadores tener la información necesaria para saber ajustar el “timing” de los niveles, reducir la frustración, saber lo que el jugador quiere y ofrecérselo… en definitiva, hacer un juego mejor diseñado.

Sin embargo, creo que los desarrolladores de videojuegos independientes tardarán bastante tiempo en poder beneficiarse de esta tecnología, ya que no es viable contratar a un programador de machine learning solo para producir herramientas para ayudar a los diseñadores, y lo más importante: una empresa pequeña no suele tener acceso a los datos suficientes ni puede crear una infraestructura para generar los suyos propios.

Bibliografía y referencias

[1] itch.io. 2020. *Games By ANGELINA - Itch.Io*. [online] Available at: <https://gamesbyangelina.itch.io/>
[Accessed 18 May 2020].

IAT. n.d. *Inteligencia Artificial En Videojuegos: Origen Y Evolución*. [online] Available at: <https://iat.es/tecnologias/inteligencia-artificial/videojuegos/>
[Accessed 18 May 2020].

Fernández, I., 2016. *Procedurally Generated Content: La Revolución De Los Videojuegos Es Ahora (Aunque Llevamos 40 Años Creándola)*. [online] Xataka.com. Available at: <https://www.xataka.com/videojuegos/procedurally-generated-content-la-revolucion-de-los-videojuegos-es-ahora-aunque-llevamos-40-anos-creandola>
[Accessed 18 May 2020].

Bobriakov, I., 2019. *Top 8 Data Science Use Cases In Gaming*. [online] Medium. Available at: <https://medium.com/activewizards-machine-learning-company/top-8-data-science-use-cases-in-gaming-de1f429ae651>
[Accessed 18 May 2020].