IA

Nombre Asignatura: Prog. de inteligencia artificial en videojuegos
Profesor: Gustavo Garanda
Curso: 2020/2021
Autor/es: Mikel Mondragon, Alex Molina

ÍNDICE

TÉCNICAS BOTTOM-UP

Pathfinding

Primero de todo es necesario saber en qué consiste una técnica de pathfinding y ya después podremos profundizar mucho más en nuestra técnica más específicamente.

¿Qué es el pathfinding?

Es una técnica de inteligencia artificial donde el personaje o cualquier objeto movible dentro de un juego es capaz de encontrar el camino más corto para llegar a su objetivo.

Usualmente se ve mucho en juegos de tipo estrategia o donde una cuadricula está más marcada en el estilo del juego.

Es una de las técnicas más importantes durante todo el proceso de un juego que la contenga, ya que si no llega a estar bien implementada podrían bloquearse los objetos movibles que la implementen, produciendo una muy mala sensación de cara al usuario final.

Hay varios tipos de algoritmos conocidos para implementar esta técnica:

- Algoritmo de Dijkstra's
- Greedy Best-First-Search
- D* Algoritmo
- A* Algoritmo

En nuestro caso el elegido es el A*. A continuación, haremos una mayor explicación sobre este entrando en más detalles técnicos.

A* Pathfinding

A* es la opción más popular para la búsqueda de caminos, porque es bastante flexible y puede utilizarse en una amplia gama de contextos y tipos de juego.

A* es como el Algoritmo de Dijkstra en el sentido de que puede utilizarse para encontrar el camino más corto. A* se parece al Greedy Best-First-Search en que puede utilizar una heurística para guiarse.

El secreto de su éxito es que combina la información que utiliza el Algoritmo de Dijkstra (favoreciendo los vértices que están cerca del punto de partida) y la información que utiliza el Greedy Best-First-Search (favoreciendo los vértices que están cerca de la meta).

En la terminología estándar utilizada al hablar de A*, g(n) representa el coste exacto del camino desde el punto de partida hasta cualquier vértice n, y h(n) representa el coste heurístico estimado desde el vértice n hasta la meta y f(n) representa la suma de los dos anteriores.

Cada vez que pasa por el bucle principal, examina el vértice n que tiene la menor f(n) = g(n) + h(n), y una vez lo tiene este será el siguiente punto por el que seguirá investigando.

Heurística

Antes se ha mencionado este término en cuanto al valor que hace referencia a la h(n), por eso ahora explicaremos con más en profundidad que es este término.

Como el tema me parece algo complejo empezaré con la definición que nos proporciona Wikipedia como introducción al tema:

"*En informática, inteligencia artificial y optimización matemática, una heurística (del griego εὑρίσκω "encuentro, descubro") es una técnica diseñada para resolver un problema más rápidamente cuando los métodos clásicos son demasiado lentos, o para encontrar una solución aproximada cuando los métodos clásicos no logran encontrar ninguna solución exacta. Esto se consigue cambiando la optimización, la exhaustividad, la exactitud o la precisión por la rapidez. En cierto modo, puede considerarse un atajo.*"

Una vez visto esto podemos decir que la función heurística h(n) indica a A* una estimación del coste mínimo desde cualquier vértice n hasta la meta. Es importante elegir una buena función heurística que permita la rapidez deseada.

Reglas seguidas por el algoritmo a*:

- En un extremo, si h(n) es 0, entonces sólo g(n) juega un papel, y A* se convierte en el Algoritmo de Dijkstra, que está garantizado para encontrar un camino más corto.
- Si h(n) es siempre inferior (o igual) al coste de ir de n a la meta, se garantiza que A* encuentra el camino más corto. Cuanto menor sea h(n), más se expande el nodo A*, lo que lo hace más lento.
- Si h(n) es exactamente igual al coste de ir de n a la meta, entonces A* sólo seguirá el mejor camino y nunca hará nada de más, haciéndolo muy rápido. Aunque no se puede hacer que esto ocurra en todos los casos, se puede hacer que sea exacto en algunos casos

especiales. Es bueno saber que, dada la información perfecta, A* se comportará perfectamente.
- Si h(n) es a veces mayor que el coste de ir de n a la meta, entonces no se garantiza que A* encuentre un camino más corto, pero puede correr más rápido.
- En el otro extremo, si h(n) es muy alta en relación con g(n), entonces sólo h(n) juega un papel, y A* se convierte en Greedy Best-First-Search.

Ahora ya sabemos todo lo necesario, solo falta encontrar el algoritmo que nos proporcione esta información. En nuestro caso h(n) se buscará con la distancia de Manhattan que consiste en la distancia más corta en un plano entre dos puntos obviando cualquier tipo de obstáculo, es decir, nos dará el número de casillas que hay desde la posición analizada hasta el objetivo obviando las paredes u obstáculos del juego.

Implementación de A*

Dentro del bucle que hará este algoritmo seguiremos varias reglas que harán que dentro de una lista final tenga el camino a seguir y el objeto movible deseado podrá seguirlo o podrá comprobar si es posible llegar hasta él ya que puede estar en una habitación aparte, las reglas son las siguiente: Tendremos dos listas, a una la llamaremos la abierta que serán posiciones a analizar y la cerrada que ya tendrá la información de las ya analizadas y en la que nos quedará el camino final.

1. Pondremos la casilla inicial donde este situado el objeto en la lista abierta.
2. Haremos un bucle hasta que la lista abierta este vacía o hayamos encontrado el camino.
    a. Recorreremos la lista abierta y elegiremos la casilla con la menos puntuación. Para entendernos la llamaremos ‘S’.
    b. Borraremos esta casilla de la lista abierta y la introduciremos en la cerrada.
    c. Para cada casilla posible ‘T’ de S (norte, sud, este, oeste), haremos lo siguiente:
        i. Si T está ya en la lista cerrada lo ignoraremos.
        ii. Si T no está en la lista abierta calcularemos su valor y lo añadiremos a la lista.
        iii. Si T está en la lista abierta, miraremos su puntuación general de cuando la calculemos antes. Si la nueva puntuación es menor actualizaremos toda la información dentro de la lista abierta de esa casilla.

Una vez siguiente este proceso tendremos en la lista cerrada, las casillas de principio a fin en el caso de que el bucle haya salido porque ha encontrado el camino.

Teniendo en cuanta que cada casilla contiene la información de su padre, es decir, la casilla de la que proviene encontrar el camino final será muy simple. Lo único a hacer es recorrer desde el final de la lista, que será la casilla del objetivo final, e ir analizando los padres y así veremos porque camino a llegado a ese punto más rápidamente. En estos dos dibujos veremos un poco el proceso de los dos primeros pasos hasta llegar al destino final.

*Ilustración 1. Ilustraciones A**

"https://www.raywenderlich.com/3016-introduction-to-a-pathfinding"

El funcionamiento del algoritmo es el que buscábamos, pero críticamente con nosotros mismos sabemos que hay margen de mejora. Ahora mismo nuestro algoritmo funciona en las 4 direcciones cardinales sin tener nada más en cuenta. En un futuro nos gustaría mejorar esto incluyendo que las posiciones a tener en cuenta sean sus adyacentes, es decir, que el objeto que aplique la función de pathfinding tenga en cuenta las direcciones en diagonal, para así una mayor resolución del camino sumándole más rapidez.

Algoritmo generación de cuevas

Este algoritmo devuelve un gran array bidimensional de bloques, cada uno de los cuales es muro o suelo. Este algoritmo se puede usar para todo tipo de juegos: niveles aleatorios para juegos de estrategia, cuadro de mapas para juegos de plataformas, tal vez incluso como arenas para un tirador multijugador. Si se voltean los bloques de muro y suelo se puede hacer un generador de isla. Todo utiliza el mismo código y salida, lo que hace que esta sea una herramienta muy flexible.

Para este algoritmo se utilizan autómatas celulares, los cuales se rigen por ciertas reglas, al formar una cuadrícula con estos autómatas obtenemos el generador de cuevas.

Para este caso hemos utilizado las siguientes reglas:

Si una célula “muro” tiene X o más vecinos muros permanece “muro”
Si una célula “muro” tiene X-1 vecinos “muro” se transforma en “suelo”
Si una célula “suelo” tiene X-1 o más vecinos muros se transforma en “muro”
Si una célula “suelo” tiene X-2 vecinos “muro” permanece “suelo”

X es una variable que se puede cambiar dando diferentes resultados. Este algoritmo se ejecuta más de una vez para darle forma, en nuestro caso utilizamos una variable llamada iteraciones.

Para realizar este algoritmo se deben seguir diferentes pasos:

En primer lugar, se crea una nueva cuadrícula en la que podamos poner nuestros valores de celda actualizados. necesitamos mirar a sus ocho vecinos:

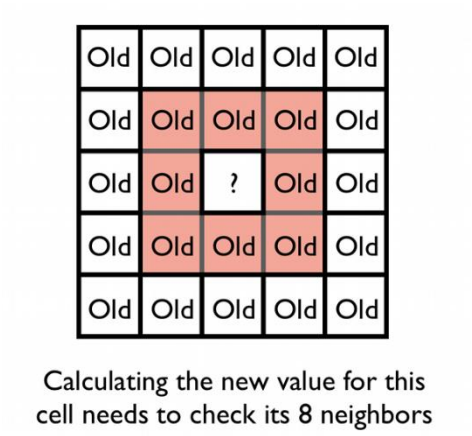
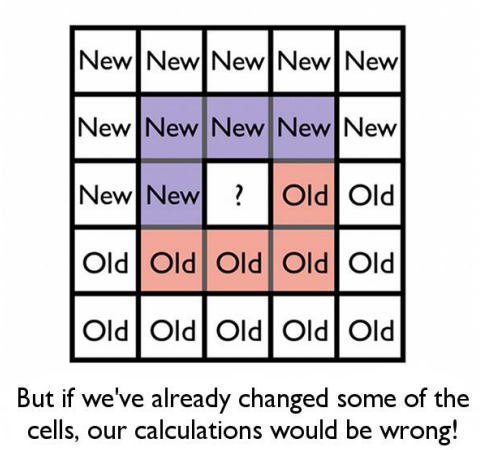


Ilustración 2. Gráfico cuevas 1

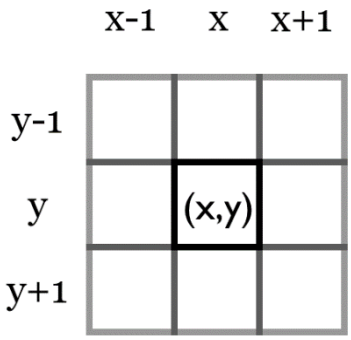
Si ya hemos calculado el nuevo valor de algunas de las celdas y las hemos vuelto a colocar en la cuadrícula, entonces nuestro cálculo será una mezcla de datos nuevos y antiguos, como esto:



Cada vez que se calcula un nuevo valor de celda, en lugar de volver a colocarlo en el mapa antiguo, se creará uno nuevo para no destruir los datos antiguos, es decir necesitaremos dos cuadrículas.

Ilustración 3. Gráfico cuevas 2

En la imagen de abajo se puede ver cómo los índices que queremos. Con un bucle for recorremos la cuadrícula teniendo cuidado en los casos en los que se pueda salir del array, los primeros elementos etc.



Finalmente, solamente hay que sustituir la nueva cuadrícula por la antigua y en caso de desearlo se puede ejecutar el algoritmo tantas veces como se quiera para tener un resultado más sólido.

Ilustración 4. Gráfico cuevas 3

La efectividad del algoritmo es la deseada, está aplicada en su versión más simple, se podría aplicar el algoritmo de inundación para realizar comprobaciones de calidad. El relleno de inundación es un algoritmo muy simple que puede utilizar para encontrar todos los espacios en una matriz que se conectan a un punto en particular. Al igual que el nombre sugiere, el algoritmo funciona un poco cómo verter un cubo de agua en su nivel - se extiende desde el punto de partida y se llena en todas las esquinas. Algunos mapas generados están formados por una gran cueva, mientras que otros tienen unas cuantas cuevas más pequeñas que están separadas entre sí. El relleno de inundación puede detectar la gran cueva que hay y luego regenerar el nivel si es demasiado pequeño o decidir dónde quieres que empiece el jugador si crees que es lo suficientemente grande.

TÉCNICAS TOP-DOWN

Árboles

Son árboles jerárquicos que deciden el flujo de las decisiones de la inteligencia artificial de un agente (un personaje). Permiten "iteraciones más rápidas que otros sistemas" y se pueden extender nuevos nodos al árbol. Cada parte del árbol permite encapsular acciones que se pueden utilizar en cualquier IA o subrama de una IA; por ejemplo, se puede disparar corriendo o desde una cobertura.

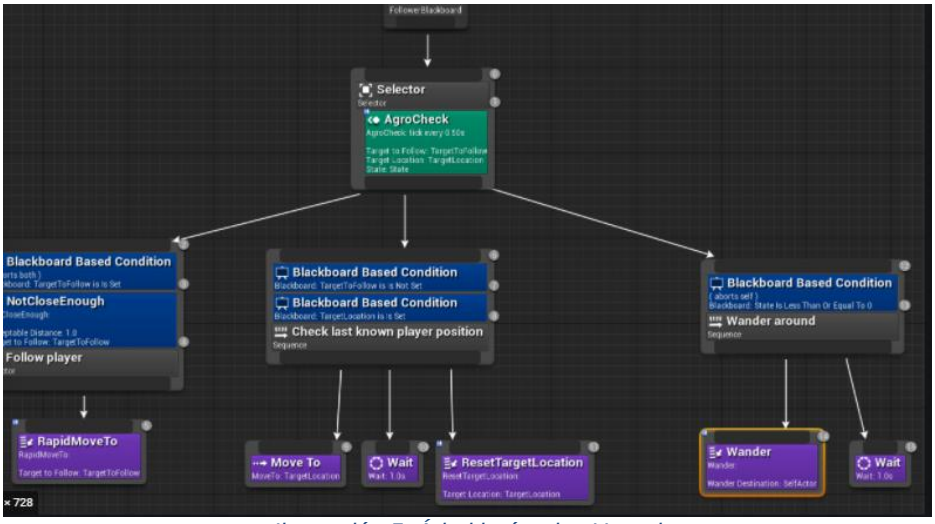


Ilustración 5. Árbol jerárquico Unreal

En Unreal los árboles se ejecutan de arriba abajo, y muy importante de izquierda a derecha, en primer lugar, cada vez que se avanza en un nodo se verifica si cumple los requisitos de los decoradores (azul), en caso de cumplirlo se ejecuta el siguiente nodo el cual lo marca con un número en la parte superior derecha. Las tareas (morado) realizan las acciones del personaje que contiene el árbol de comportamiento.

Para aplicarlo en nuestro juego en primer lugar hemos creado un blackboard (el cual contiene las variables de nuestro árbol) y un behaviour tree, una vez creados estos hemos creado una IA y lo hemos configurado de manera que utilice estos componentes, una vez terminada la base hemos creado diferentes tareas por código cómo sería calcular el pathfinding, atacar, etc. Finalmente, al tener todo lo necesario se ha creado el árbol de comportamiento con sus correspondientes decoradores y tareas.

El árbol de comportamiento aplicado es simple pero correcto, al ser una jugabilidad simple no es necesario crear un árbol de una gran complejidad, en caso de no aumentar la complejidad del combate podríamos añadirle diferentes patrones de movimientos dependiendo del estado de los enemigos, por ejemplo que calcule la distancia que hay hasta el jugador y en caso de que sea mayor que X realice un patrón de movimiento de patrulla o aleatorio cada vez que sea su turno, también podría añadirse un patrón de movimiento de huida cuando la salud actual del enemigo fuese menor que X porcentaje. En caso de que el combate fuese más complejo y existieran más de un combate se podría realizar una comprobación para ver cual es el movimiento más óptimo y realizar dicho movimiento, o también podría añadirse cierta aleatoriedad.

TÉCNICAS EMERGENTES

Dentro de este apartado hablaremos de una tecnología emergente que está cada vez más presente en el mundo de la inteligencia artificial, aparte le daremos un enfoque personal agregando un apartado de como incluirla en el mundo de los videojuegos.

Machine learning

Los algoritmos de machine learning utilizan la estadística para encontrar patrones en cantidades masivas de datos (big data). Y los datos, en este caso, abarcan muchas cosas: números, palabras, imágenes, clics, etc. Si pueden almacenarse digitalmente, pueden introducirse en un algoritmo de aprendizaje automático.

El machine learning es el proceso que impulsa muchos de los servicios que utilizamos hoy en día: sistemas de recomendación como los de Netflix, YouTube y Spotify, motores de búsqueda como Google y Baidu, fuentes de medios sociales como Facebook y Twitter, asistentes de voz como Siri, Alexa y muchos más.

Los algoritmos suelen clasificarse como supervisados o no supervisados.

- **Algoritmos supervisados**: pueden aplicar lo aprendido en el pasado a nuevos datos utilizando ejemplos etiquetados para predecir eventos futuros. Partiendo del análisis de un conjunto de datos de entrenamiento conocido, el algoritmo de aprendizaje produce una función inferida para hacer predicciones sobre los valores de salida. El sistema es capaz de proporcionar objetivos para cualquier entrada nueva después de un entrenamiento suficiente. El algoritmo de aprendizaje también puede comparar su salida con la correcta, la prevista, y encontrar errores para modificar el modelo en consecuencia.
- **Algoritmos no supervisados**: se utilizan cuando la información utilizada para el entrenamiento no está clasificada ni etiquetada. El aprendizaje no supervisado estudia cómo los sistemas pueden inferir una función para describir una estructura oculta a partir de datos no etiquetados. El sistema no averigua la salida correcta, sino que explora los datos y puede extraer inferencias de los conjuntos de datos para describir estructuras ocultas a partir de datos no etiquetados.
- **Los algoritmos semisupervisados**: se sitúan en un punto intermedio entre el aprendizaje supervisado y el no supervisado, ya que utilizan

tanto datos etiquetados como no etiquetados para el entrenamiento, normalmente una pequeña cantidad de datos etiquetados y una gran cantidad de datos no etiquetados. Los sistemas que utilizan este método son capaces de mejorar considerablemente la precisión del aprendizaje.

- **Algoritmos por refuerzo**: son un método de aprendizaje que interactúa con su entorno produciendo acciones y descubre errores o recompensas Este método permite a las máquinas y agentes de software determinar automáticamente el comportamiento ideal dentro de un contexto específico para maximizar su rendimiento. Para que el agente aprenda cuál es la mejor acción se requiere una simple retroalimentación de recompensa, lo que se conoce como señal de refuerzo.

A continuación, y ver una pequeña relación con el mundo de los videojuegos mostraré una pequeña lista de cosas que el machine learning podría proporcionar a nuestra industria.

1. **Algoritmos que juegan como NPC**: En la actualidad, tus oponentes en un videojuego son NPC (personajes no jugables) preestablecidos, pero un NPC basado en el aprendizaje automático podría permitirte jugar contra enemigos menos predecibles. Estos enemigos también podrían ajustar su nivel de dificultad. A medida que el jugador aprende a jugar, sus enemigos podrían ser más inteligentes y responder de forma única en función de sus acciones dentro del juego.

2. **Modelado de sistemas complejos**: El punto fuerte de un algoritmo de machine learning es su capacidad para modelar sistemas complejos. Los desarrolladores de videojuegos intentan constantemente que los juegos sean más inmersivos y realistas. Por supuesto, modelar el mundo real es increíblemente difícil, pero un algoritmo de aprendizaje automático podría ayudar a predecir los efectos posteriores de las acciones de un jugador o incluso a modelar cosas que el jugador no puede controlar, como el clima.

3. **Hacer los juegos más bonitos**: Microsoft está trabajando con Nvidia en este problema. Están utilizando el machine learning para mejorar las imágenes y las representaciones de forma dinámica. En la vida real, cuando estás lejos de un objeto los detalles no son claros, pero a medida que te acercas puedes notar detalles más finos. Este renderizado dinámico de detalles más finos es un reto en el que los algoritmos de visión por ordenador pueden ayudar.

4. **Interacciones más realistas**: El uso del procesamiento del lenguaje natural podría permitirte hablar en voz alta con los personajes del juego y obtener respuestas reales, de forma parecida a cuando hablas con Siri, Alexa o Google Assistant. Además, los juegos que incorporan hápticos de RV o imágenes del jugador podrían permitir que los algoritmos de visión por ordenador detecten el lenguaje corporal y las intenciones, mejorando aún más la experiencia de interactuar con los NPC.

5. **Creación de universos sobre la marcha**: Hasta la fecha, algunos de los videojuegos más populares son juegos de mapa abierto que permiten explorar un paisaje masivo. Esos juegos requieren miles de horas de tiempo de desarrolladores y artistas para su renderización. Sin embargo, el machine learning podría ayudar en la búsqueda de caminos y la creación de mundos. Un ejemplo es un juego como No Man's Sky con un número infinito de nuevos mundos para descubrir, todos generados en el momento a medida que exploras.

Una vez vista la lista es fácil asimilar que el machine learning puede estar presente en cualquier ámbito de los videojuegos como: Diseño, IA, Mobile, Gráfica, etc.

Para finalizar mostraré críticamente la efectividad del machine learning con una pequeña tabla de ventajas y desventajas, para una mejor comprensión final:

VENTAJAS	DESVENTAJAS
1. Identifica fácilmente tendencias y patrones	1. Adquisición de datos: Necesita una gran cantidad de datos para ser útil.
2. No es necesaria la intervención humana (automatización)	2. Tiempo y recursos
3. Mejora continua	3. Interpretación de los resultados
4. Manejo de datos multidimensionales y multivariados	4. Alta susceptibilidad a los errores

TECNOLOGIAS FUTURAS

Como parte final del documento, mencionaremos tres tecnologías de IA emergentes en un futuro próximo y lo que traerán con ellas.

Compresión de redes neuronales

La IA ha progresado rápidamente en el análisis de big data aprovechando las redes neuronales profundas (DNN). Sin embargo, la principal desventaja de cualquier red neuronal es que es intensiva en términos de computación y memoria, lo que dificulta su implementación en sistemas integrados con recursos de hardware limitados. Además, con el aumento del tamaño de las DNN para realizar cálculos complejos, también aumentan las necesidades de almacenamiento. Para resolver estos problemas, los investigadores han ideado una técnica de IA denominada compresión de redes neuronales.

Por lo general, una red neuronal contiene muchos más pesos, representados con mayor precisión de la que se requiere para la tarea específica para la que están entrenados. Si queremos aportar inteligencia en tiempo real o potenciar las aplicaciones de vanguardia, los modelos de las redes neuronales deben ser más pequeños. Para comprimir los modelos, los investigadores recurren a los siguientes métodos: poda y compartición de parámetros, cuantificación, factorización de bajo rango, filtros convolucionales transferidos o compactos y destilación de conocimientos.

- La poda identifica y elimina los pesos o conexiones o parámetros innecesarios, dejando la red con los importantes.
- La cuantificación comprime el modelo reduciendo el número de bits que representan cada conexión. La factorización de bajo rango aprovecha la descomposición de matrices para estimar los parámetros informativos de las DNN.
- Los filtros de convolución compactos ayudan a filtrar el espacio de pesos o parámetros innecesarios y a retener los importantes que se necesitan para llevar a cabo la convolución, con lo que se ahorra espacio de almacenamiento.
- Y la destilación de conocimientos ayuda a entrenar una red neuronal más compacta para imitar la salida de una red más grande.

Aprendizaje federado

El aprendizaje federado se define como una técnica de aprendizaje que permite a los usuarios cosechar colectivamente los beneficios de los modelos compartidos entrenados a partir de datos, sin necesidad de almacenarlos de forma centralizada. "Communication-Efficient Learning of Deep Networks from Decentralized Data"

El machine learning tradicional implica una canalización de datos que utiliza un servidor central que aloja el modelo entrenado para realizar predicciones. El inconveniente de esta arquitectura es que todos los datos recogidos por los dispositivos y sensores locales se envían al servidor central para su procesamiento y, posteriormente, se devuelven a los dispositivos. Este viaje de ida y vuelta limita la capacidad del modelo para aprender en tiempo real.

El aprendizaje federado (FL), por el contrario, es un enfoque que descarga el modelo actual y calcula un modelo actualizado en el propio dispositivo utilizando datos locales. Estos modelos entrenados localmente se envían desde los dispositivos al servidor central, donde se agregan, es decir, se promedian los pesos, y luego se envía a los dispositivos un único modelo global consolidado y mejorado.

Como ejemplos del aprendizaje federado y donde se utiliza mencionaremos a:

- Mejorar las capacidades de reconocimiento de voz de Siri.
- Google había empleado inicialmente el aprendizaje federado para aumentar la recomendación de palabras en el teclado de Google para Android sin subir los datos de texto del usuario a la nube.

IA Generativa

Los recientes avances en IA han permitido a muchas empresas desarrollar algoritmos y herramientas para generar imágenes artificiales en 3D y 2D de forma automática. Estos algoritmos forman esencialmente la IA generativa, que permite a las máquinas utilizar elementos como texto, archivos de audio e imágenes para crear contenidos. La revista MIT Technology review describió la IA generativa como uno de los avances más prometedores en el mundo de la IA en la última década. Está preparada para la próxima generación de aplicaciones de programación de automóviles, desarrollo de contenidos, artes visuales y otras actividades creativas, de diseño e ingeniería. Por ejemplo, NVIDIA ha desarrollado un software capaz de generar nuevos rostros fotorrealistas a partir de unas pocas fotos de personas reales.

También puede utilizarse para ofrecer un mejor servicio al cliente, facilitar y agilizar los registros, permitir la supervisión del rendimiento, la conectividad sin fisuras y el control de calidad, y ayudar a encontrar nuevas oportunidades para establecer contactos.

La IA generativa también puede ayudar en el ámbito de la sanidad, puede permitir la identificación temprana de posibles enfermedades malignas para que los planes de tratamiento sean más eficaces. Por ejemplo, incluso IBM está utilizando esta tecnología para investigar sobre el péptido antimicrobiano (AMP) con el fin de encontrar medicamentos para el COVID-19.

Para finalizar mencionar en como la IA a cambiado el mundo en el que vivimos. Ya se han visto 3 ejemplos de tecnologías futuras y de lo que serán capaces de hacer, pero ya en la actualidad tenemos millones de sistemas controlados por inteligencia artificial que ayudan a la población en su día a día. Desde cosas tan simples como mejoras en el campo y ámbitos del sector principal de nuestra sociedad hasta cosas ya más innovadores pero útiles, como IAs capaces de controlar industrias o empresas al completo, cualquier tipo de gran empresa ahora mismo tendrá en menor o mayor medida IA implantada dentro, en el mundo del deporte constantemente se usa IA para mejorar el rendimiento de jugadores, hasta el ejemplos de ocio como la increíble mejora en IA dentro de videojuegos permitiendo a los usuarios sentirse más realizados o plasmados en como va mejorando y como NPC o otros factores de los juegos ya se asemejan a la vida real.

WEBGRAFIA

https://www.gamerdic.es/termino/pathfinding/

https://medium.com/swlh/pathfinding-algorithms-6c0d4febe8fd

https://es.wikipedia.org/wiki/Heur%C3%ADstica

https://www.raywenderlich.com/3016-introduction-to-a-pathfinding

http://theory.stanford.edu/

https://karlmatthes.medium.com/pathfinding-a-search-algorithm-d77400c89888

https://cleverdata.io/que-es-machine-learning-big-data/

https://www.expert.ai/blog/machine-learning-definition/

https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/

https://www.ibm.com/cloud/learn/machine-learning

https://www.logikk.com/articles/machine-learning-in-game-development/

https://www.analyticsinsight.net/top-3-emerging-technologies-in-artificial-intelligence-in-the-2020s/

https://content.techgig.com/3-emerging-ai-technologies-for-2021-and-beyond/articleshow/79511388.cms

https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

https://medium.com/@ODSC/what-is-federated-learning-99c7fc9bc4f5

https://gamedevelopment.tutsplus.com/es/tutorials/generate-random-cave-levels-using-cellular-automata--gamedev-9664

TABLA DE ILUSTRACIONES