AI Jugando Las Traes

Roberto André Mossi Milla
Estudiante de Ciencias de la Computación, Universidad de Gannon
Dr. Ramakrishnan Sundaram
Profesor de Ingeniería Eléctrica y Cibernética, Universidad de Gannon

Resumen

Este artículo desarrolla un AI que juega a las traes a través de la competencia entre diferentes sujetos y el aspecto de supervivencia de un juego de *las traes*. Mediante el uso de genomas, se crean entidades sofisticadas que se coordinan entre sí para el beneficio de su supervivencia. A través de múltiples pruebas utilizando aprendizaje reforzado con genomas como base, se buscan los genes de supervivencia más aptos y estrategias de auto-desarrollo. Posteriormente, se introducen sus capacidades de razonamiento mediante el uso de entornos más complejos generados aleatoriamente, lo que conduce a habilidades más similares a las humanas. Esta investigación ya ha sido realizada previamente en otros juegos como Open-AI hide & seek, pero para este estudio se añadieron comportamientos más humanos y se utilizaron genomas para observar su ingenio desde una perspectiva biológica. Esta investigación apoya el desarrollo del currículo al introducir a estudiantes del Departamento de Ciencias de la Computación (CS) al desarrollo de inteligencia artificial, y cómo otras áreas como la biología pueden aportar un desarrollo adicional a la tecnología que desarrollan. Se compara cómo esto se relaciona con la estructura biológica de los organismos en su toma de decisiones basada en tareas, inicialización aleatoria y restricciones. Con base en su comportamiento, se propondrán métodos de ajuste fino para mejorar la investigación hacia una replicación más cercana del comportamiento humano mediante el uso de genomas.

[1] Publicación de ASEE disponible en `https://peer.asee.org/54692`

[2] Presentado en la Exposición de Investigación Sigma Xi de la Universidad Estatal de Pensilvania (`https://sites.psu.edu/behrendsigmaxi2025/2025/03/11/tag-ai-sandbox/`)

Introducción

Utilizando genomas, el comportamiento de los sujetos será determinado a través de sus genes. Los genes se determinan al inicio mediante un proceso aleatorio, con el fin de otorgar a las entidades un punto de partida para aprender y adaptarse a las tareas de supervivencia que deben completar. A partir de ahí, se les asigna una tarea; en este caso, se establecen aleatoriamente dos grupos, los cuales se dividen en corredores y buscadores. La tarea de estos grupos consiste en que los buscadores deben atrapar a los corredores, y los corredores deben huir de ellos en un número limitado de intentos. Los buscadores que logran atrapar a su objetivo sobreviven, mientras que aquellos que no lo hacen mueren inmediatamente. El equipo contrario aplica el mismo principio, pero con el objetivo de evitar ser atrapado.

Para el desarrollo completo de este proyecto, se utilizó Unreal Engine. La razón detrás de usar Unreal Engine es que las interacciones de comportamiento similares a las humanas requieren física. Existen muchas opciones, pero Unreal Engine es uno de los motores más populares del mercado, aunque también es conocido por su mayor complejidad debido a que su lenguaje principal es C++. Se tomó esto como un desafío con la esperanza de obtener el mayor rendimiento posible. Para lograrlo, se cuenta con la asistencia de varias librerías. Una de ellas es PyTorch, utilizada para el desarrollo de inteligencia artificial. La otra es CUDA, cuya funcionalidad consiste en redirigir el procesamiento gráfico del proyecto a la tarjeta gráfica, permitiendo que el resto del sistema se encargue del procesamiento del comportamiento.

Trabajos Relacionados

Durante un tiempo, han existido múltiples intentos de crear inteligencia artificial enfocada en el estudio de organismos vivos. Para este caso específico, existe inspiración en dos proyectos concretos. El primer proyecto es *Multi-Agent Hide and Seek* [2], desarrollado por Open-AI. Este proyecto brindó la inspiración para crear un entorno basado en un juego con el fin de estudiar el ingenio de la IA en situaciones de resolución de problemas. El segundo proyecto es el video *I programmed some creatures* [3]. Este proyecto tiene como idea principal la gestión del comportamiento de los organismos mediante el uso de genomas bajo las restricciones del entorno. Esto establece una conexión entre el mundo de la informática y la biología.

Requisitos

Tipo	Versión	Nombre	Descripción	Razón
Lenguaje	C++20	C++	Lenguaje OOP de bajo nivel usado para Unreal Engine	Es ampliamente utilizado en programación de IA y por su velocidad.
Lenguaje	3.31.6	CMake	Lenguaje utilizado para automatización entre sistemas operativos	Al usar múltiples repositorios, se emplea CMake para automatizar la configuración y asegurar compatibilidad multiplataforma
Librería	12.0.0	Cuda	Modelo de programación para GPU	Para procesamiento gráfico intensivo, se usa esta librería para acelerar el cálculo por GPU
Gráficos	5.4.4	Unreal Engine	Creación gráfica y visualización de entidades	Capaz de manejar computación compleja y alto rendimiento

Generación del Mundo

En este caso era evidente que las entidades requerían un mundo en el cual pudieran desplazarse. Inicialmente se consideró crear un mundo estático genérico donde pudieran moverse libremente y aprender el juego por sí mismas. Sin embargo, surgió la posibilidad de que eventualmente se volvieran muy buenas jugando, pero sin espacio adicional para crecer al estar limitadas al mismo entorno. Para resolver esto, se determinó que sería necesario generar nuevos mundos para que pudieran aplicar lo aprendido en nuevos escenarios y así mejorar su toma de decisiones. Hacer esto manualmente después de cada ciclo implicaría un costo de tiempo significativo, por lo que se decidió implementar un generador de mundos[1]. Esta decisión ya había sido aplicada en proyectos a largo plazo como Hide & Seek de Open-AI, aunque en proyectos pequeños suele modelarse el mundo manualmente. Para evitar redundancia, un generador aleatorio resulta beneficioso.

Para crear este mundo, primero se planificó el algoritmo a utilizar. Se revisaron diversos recursos sobre generadores en Unreal Engine y fuera de él. Muchos utilizaban el algoritmo A-Star, basado en el algoritmo de Dijkstra, el cual considera vecinos, rutas más cortas y distancia al nodo final [4]. Esto es útil cuando existe un punto inicial y final definido [5].

En su lugar, se optó por un enfoque similar a un algoritmo de búsqueda en anchura [6], sin un punto final definido, sino con un número total de salas a generar. Dado que el objetivo del juego no es llegar a un destino específico, sino huir o perseguir, se eligió una solución más aleatoria.

El proceso funciona de la siguiente manera: se define un punto inicial llamado origen, el cual genera una sala aleatoria. Las salas contienen flechas de salida que indican posibles conexiones. El generador elige aleatoriamente una salida y genera la siguiente sala conectada, repitiendo el proceso hasta alcanzar el número deseado de salas [7].

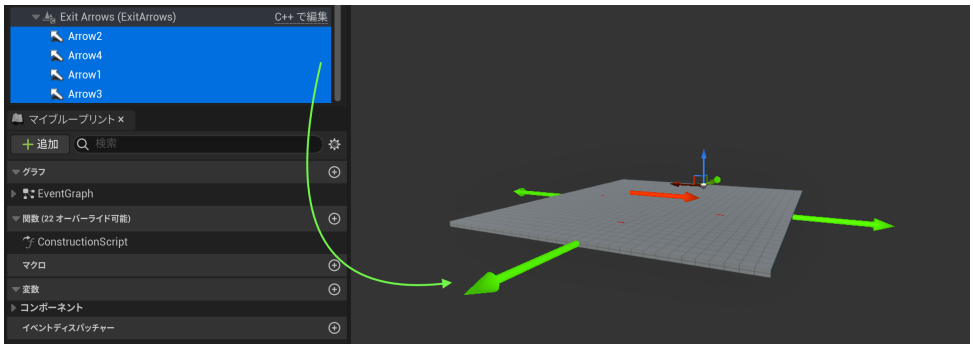

Figura 1: Configuración de Salas

Detección de Colisiones

¿Pero qué ocurrirá si ya no hay más salidas disponibles? Teniendo esa pregunta en consideración, creé dos funciones adicionales: reboot y delete room. La función reboot lo que hace es ejecutar la eliminación de cada habitación que haya sido creada; una vez que todas han sido eliminadas, ejecuta una función de borrado para el generador actual, la cual espera a que se ejecuten todas las instrucciones restantes antes de eliminarse a sí misma. Después de eso, se ejecuta la última instrucción, que genera un nuevo generador de habitaciones e inicia todo el proceso nuevamente, generando aleatoriamente un nuevo mundo con los requisitos dados.

Pero, ¿qué sucede con la colisión entre diferentes habitaciones?, ¿se superpondrán? Para esta situación creé varias funciones que cubren colisiones en distintos escenarios. Estas situaciones pueden separarse en dos categorías diferentes.

[1] El código puede encontrarse en https://github.com/AndreM222/Procedural-Generator

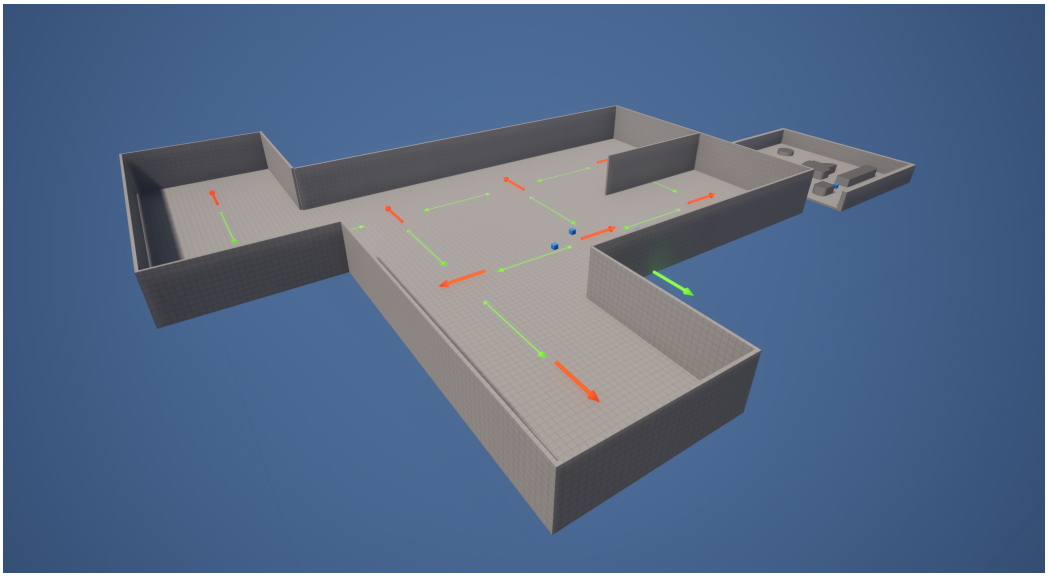

Figura 2: Vista Previa del Generador

- **Colisión General:** Para la colisión general, todo lo que se requiere es obtener el espacio donde se intenta generar una habitación y comprobar si existe algo en ese punto. Para ello, la función genera una caja de trazado invisible que, si colisiona con cualquier objeto, indica que se debe buscar una nueva ubicación.

Comprobación de barrido de colisión — cpp

```cpp
// Setup Box Dimensions for trace
FVector BoxDimensions = FVector(50, 50, 50);

FVector traceLocation = exitLocation.GetLocation();

// Trace box to check if collision available
FCollisionShape box = FCollisionShape::MakeBox(BoxDimensions);
exitsList[spaceIndex]->GetWorld()->SweepSingleByChannel(
    Hit,
    traceLocation,
    traceLocation,
    FQuat::Identity,
    ECC_Visibility,
    box,
    QueryParams
);

// Return true if room type has been found connected
if (Cast<AMaster_Room>(Hit.GetActor())) return true;
```

- **Generación por Dimensiones:** En base a las dimensiones de la habitación que se intenta generar, es necesario verificar si existe espacio suficiente para que dicho punto pueda ser ocupado. Sin embargo, obtener las dimensiones de una habitación de antemano no es posible con la versión actual de Unreal Engine. Para resolver esto, implementé una solución improvisada: generar brevemente la habitación deseada para obtener sus dimensiones, almacenarlas en una variable de transformación y luego destruir la habitación. Todo este proceso ocurre tan rápido que la habitación generada no llega a ser visible. A partir de ahí, junto con la colisión general, se pasan los nuevos valores calculados y se reemplazan las dimensiones de la caja de trazado para realizar una prueba de colisión más precisa.

Generador de dimensiones cpp

```cpp
// Make temporary room
AActor* roomModel = GetWorld()->SpawnActor<AMaster_Room>(
    RoomToSpawn[roomIndex],
    exitLocation.GetLocation(),
    FRotator::ZeroRotator,
    SpawnParams
);

// Setup Box Dimensions for trace
FVector BoxDimensions = FVector(roomModel->GetComponentsBoundingBox().GetExtent() - 4);
BoxDimensions.Z += 3;
roomModel->Destroy(); // Destroy tmp room after setup

FVector traceLocation = exitLocation.GetLocation() + (
    exitLocation.GetLocation() - actorLocation.GetLocation());
traceLocation.Z += BoxDimensions.Z; // Center Box Trace

// Trace box to check if collision available
FCollisionShape box = FCollisionShape::MakeBox(BoxDimensions);
```

Encapsulación del Mundo

En el mundo que estamos creando, es necesario agregar ciertas restricciones para las entidades. Estas restricciones ayudan a evitar que las entidades salgan de los límites del mundo y caigan al vacío. El enfoque principal de este proyecto no es que aprendan dónde pueden caerse, sino cómo reaccionar ante la interacción entre diversas entidades. Para estas prevenciones, decidí encapsular con paredes las salidas que quedaron vacías al final de la generación.

Al cerrar las paredes, es importante evitar que se generen en lugares que conectan con otras habitaciones. Si no se tomara esta precaución, no habría diferencia entre un mundo y un laberinto. Para aclarar esta distinción: un laberinto es generalmente restrictivo; el movimiento está limitado por caminos delimitados por paredes, comúnmente definidos como corredores. En cambio, un mundo, en este contexto, es menos restrictivo y está compuesto tanto por corredores como por espacios abiertos.

Para evitar que las paredes se generen en todos los lugares posibles, se crean pequeñas cajas de trazado. Estas cajas se utilizan únicamente para detectar si existe alguna habitación conectada al punto donde se desea colocar la pared. Si se detecta una habitación conectada, dicha salida se elimina de la lista de salidas abiertas y se continúa con la siguiente salida a cerrar. Si la habitación deseada no puede colocarse en ninguna de las salidas disponibles, se elimina de la lista de habitaciones posibles y se continúa con la siguiente hasta que no queden más salidas ni habitaciones disponibles.

Semilla

En cierto punto me di cuenta de que podría ser útil mantener a las entidades en una habitación específica durante un período de tiempo. Para ello decidí agregar una opción de semilla[2]. Cuando se utiliza una semilla, dependiendo del valor, se generará una habitación que permanecerá igual sin importar cuántas veces se recargue. Sin embargo, el uso de una semilla introduce la posibilidad de que no se logre generar el número total de habitaciones deseado. Este fenómeno puede ocurrir cuando las habitaciones se generan siguiendo un camino específico que eventualmente termina cerrándose sobre sí mismo, sin dejar salidas disponibles. Para este caso, indiqué al generador que omitiera el ciclo de creación de nuevos generadores hasta que uno cumpliera con los requisitos establecidos. Esto se hizo para evitar la ocurrencia de un bucle infinito.

[2]Más información disponible en https://github.com/AndreM222/Procedural-Generator/blob/main/docs/setup.md

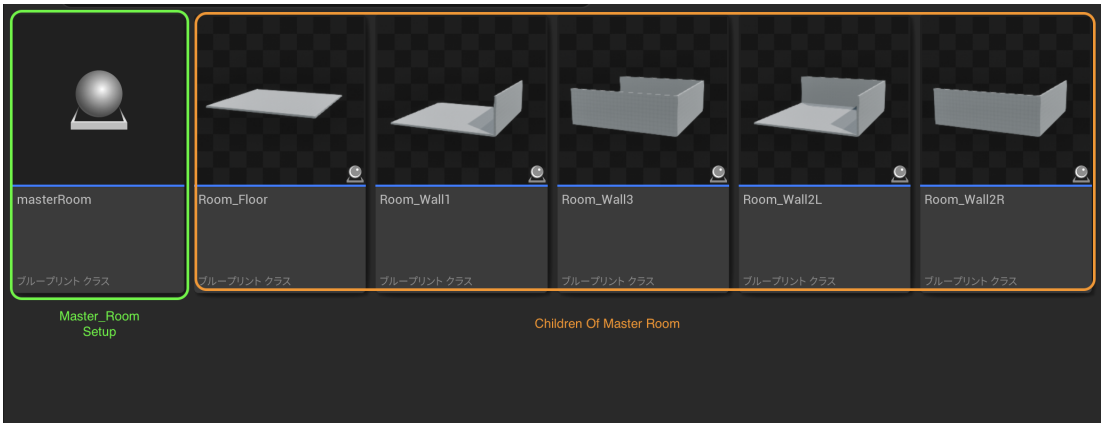

Figura 3: Creación de Habitaciones

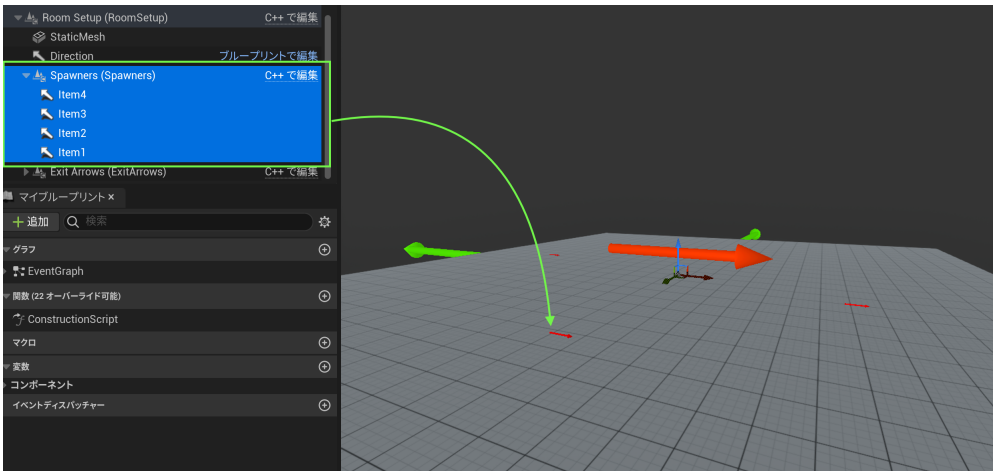

Figura 4: Generador de Props

Figura 5: Configuración de Props

Props y Generadores

Para añadir generadores de entidades y obstáculos, implementé una función de generación que crea actores de manera aleatoria con capacidades de filtrado, distribuyéndolos a lo largo de las habitaciones. Esta función busca las flechas de generación dentro de la carpeta de generadores.

Configuración del Personaje

Para el modelo 3D y las animaciones estoy utilizando los recursos de un paquete llamado *Advance Locomotion System V4* [8]. Este paquete contiene una gran cantidad de animaciones útiles y algunos modelos simples de los cuales pude beneficiarme para enfocarme en el desarrollo de las acciones y el razonamiento de los personajes.

Este paquete también incluye una configuración para el movimiento, pero fue realizada completamente en blueprints, lo cual no es fácilmente accesible desde C++. Para esta situación desarrollé mi propia lógica con respecto a la interacción que las entidades[3] realizan en el espacio 3D. Los comportamientos incluidos en el proyecto fueron realizados completamente utilizando blueprints. Esta es una de las ventajas de Unreal Engine, pero surge un problema. Actualmente no es posible interactuar desde C++ hacia los blueprints, aunque sí puede hacerse de forma inversa. Por lo tanto, desarrollé mi propia versión de la locomoción para lograr un movimiento realista, utilizando las animaciones del paquete.

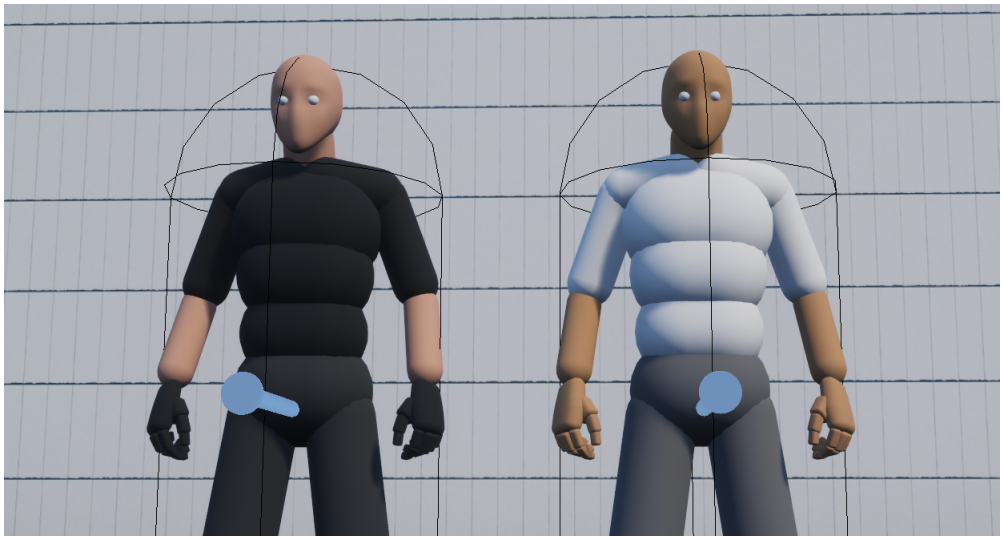

Figura 6: Configuración del Personaje

Estados

Los estados son las diferentes posturas en las que las entidades pueden cambiar dependiendo de su interacción, con el objetivo de aplicar interacciones más humanas en el mundo real. Las interacciones podrían haberse realizado sin animaciones, pero si se desean considerar los retrasos en la reacción y una mejor facilidad de visualización, decidí utilizar las animaciones del paquete de recursos para hacerlo posible. A continuación se muestran los estados disponibles creados.

[3] El código puede encontrarse en https://github.com/AndreM222/AI-Entities

- **Usado:** Actualmente se utiliza para algún tipo de interacción.
- **Planificado:** Está listo para ser utilizado, pero aún no se ha combinado con la lógica para activarlo.

Postura	Descripción	Estado
Default	Es una postura que invoca neutralidad y paz	Planificado
Masculine	Está listo para combate o para mostrar poder	Usado
Feminine	Es una postura más elegante y noble	Planificado
Injured	Muestra la baja cantidad de salud de las entidades	Usado
HandsTied	Muestra cuando existe una restricción en las entidades	Planificado
Rifle	Sosteniendo un rifle	Planificado
Pistol 1H	Sosteniendo una pistola con una mano	Planificado
Pistol 2H	Sosteniendo una pistola con ambas manos	Planificado
Bow	Sosteniendo un arco	Planificado
Torch	Sosteniendo una antorcha para iluminación	Planificado
Binoculars	Sosteniendo binoculares para ver a larga distancia	Planificado
Box	Cargando cajas	Planificado
Barrel	Cargando un barril	Planificado
Reach	Postura para alcanzar y tocar el objetivo a corta distancia	Planificado

Acciones

Las acciones son las diferentes capacidades que las entidades tienen para interactuar con el mundo. Esto añade mayor complejidad y variedad al razonamiento de las entidades.

Acción	Descripción	Estado
Run	Movimiento a alta velocidad	Usado
Sprint	Movimiento a velocidad media	Usado
Walk	Movimiento a baja velocidad	Usado
Crouch	Agacharse en espacios pequeños	Usado
Climb	Escalar paredes	Usado
Grab	Agarrar un objeto	Planificado
Drop	Soltar el objeto sostenido	Planificado
Jump	Saltar	Usado
Touch	Tocar el objetivo	Usado

Toma de Decisiones a Través de Genomas

He comenzado el desarrollo de la inteligencia de los personajes utilizando genomas, pero aún faltan una variedad de funciones para que puedan tomar decisiones racionales a través de generaciones de datos proporcionados por los

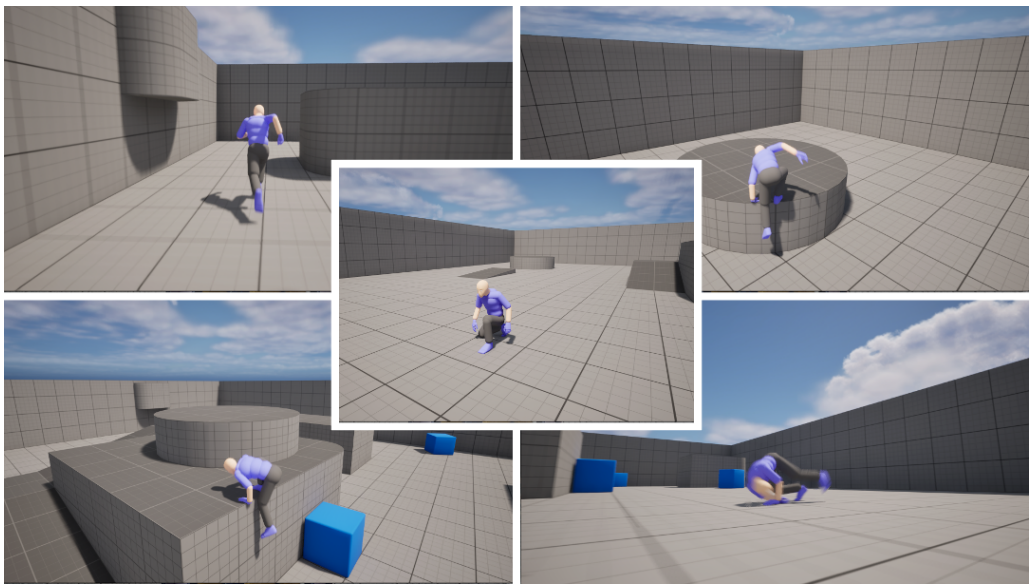

Figura 7: Vista Previa de Acciones

padres.

Esta aplicación tuvo éxito en tareas simples, como que las entidades tengan la necesidad de dirigirse a áreas específicas del mapa. Sin embargo, el juego de atrapar (las traes) es una tarea más compleja que requerirá más datos y capacidades por parte de las entidades.

Algoritmo Genético

Para más detalles sobre cómo funciona esto, se utiliza un algoritmo genético [9], lo cual implica el uso de la evolución. En cuanto a la toma de decisiones, podríamos permitir que una entidad intente cablear su propio cerebro mediante algo llamado mutación. Sin embargo, habrá un límite en cuanto a cuánto puede crecer o encontrar mejores formas de resolver un problema. Como todos sabemos, cuando se trata del desarrollo personal existen factores internos y externos que nos ayudan a crecer. Los factores internos se entienden como las experiencias que hemos vivido y que utilizamos para tomar decisiones, mientras que los factores externos son aquellos que nos enseñan a partir de su propia experiencia. Este factor no solo nos hace crecer, como lo hace el interno, sino que nos permite crecer de la manera más eficiente.

Algoritmo NEAT

El algoritmo NEAT [10] utiliza el algoritmo genético, pero pone en práctica la idea de los genomas para desarrollar sus cerebros. Al contar con sentidos, acciones y neuronas internas, estas neuronas se conectan como topologías aumentadas, lo que significa que su cerebro evoluciona según sus necesidades. Sin embargo, para mantener cierto control, se establece un límite máximo de neuronas.

El algoritmo toma dos padres y los combina utilizando feromonas determinadas por su similitud, la cual en este caso se define como qué tan bien resuelven su tarea. Todas estas neuronas pueden combinarse entre sí y consigo mismas. Para realizar una limpieza, se implementó una función que detecta si alguna conexión no cumple ninguna función y la elimina.

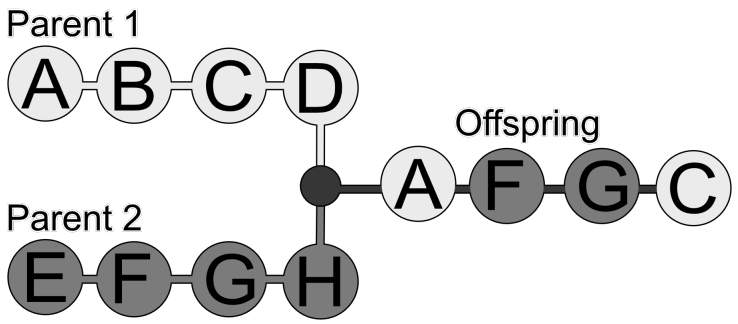

Figura 8: Algoritmo Genético

Si una entidad tiene demasiados sentidos, puede verse abrumada y no ejecutar correctamente sus acciones, pero si tiene muy pocos, no encontrará la mejor forma de resolver un problema.

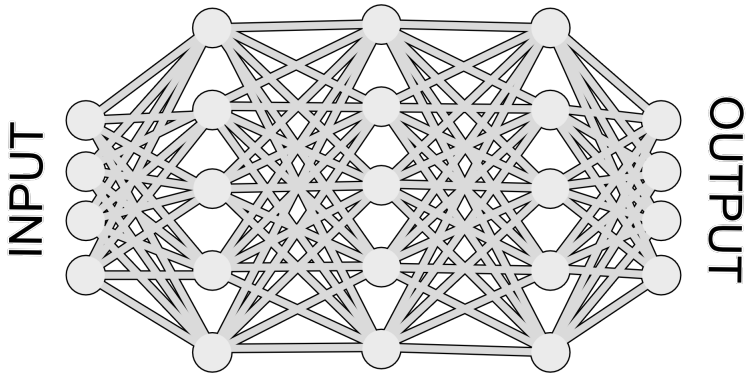

Figura 9: Algoritmo NEAT

Senses

Para estas entidades, comencé a desarrollar sentidos similares a los de los humanos. Por sentidos me refiero al sentido del oído, el sentido de la vista, etc. Sin embargo, para este proyecto limitaremos estos sentidos a los siguientes.

Tipo de Sentido	Descripción	Estado
Espacio	Este sentido permitirá a las entidades saber dónde se encuentran.	En Uso
Tacto	Se utilizará para obtener datos de sus interacciones y decidir si les agrada o desagrada.	En Desarrollo
Vista	Permite a las entidades saber qué tienen frente a ellas y decidir qué hacer.	En Uso

Pero cada entrada tiene subdivisiones que crean complejidad adicional. Por ejemplo, para obtener la vista es necesario calcular la distancia a los actores, la distancia a las paredes, la cantidad de población, entre otros. Para ello, se utilizan diversos trazados con el fin de calcular distancias, cantidades, qué hay al frente, etc.

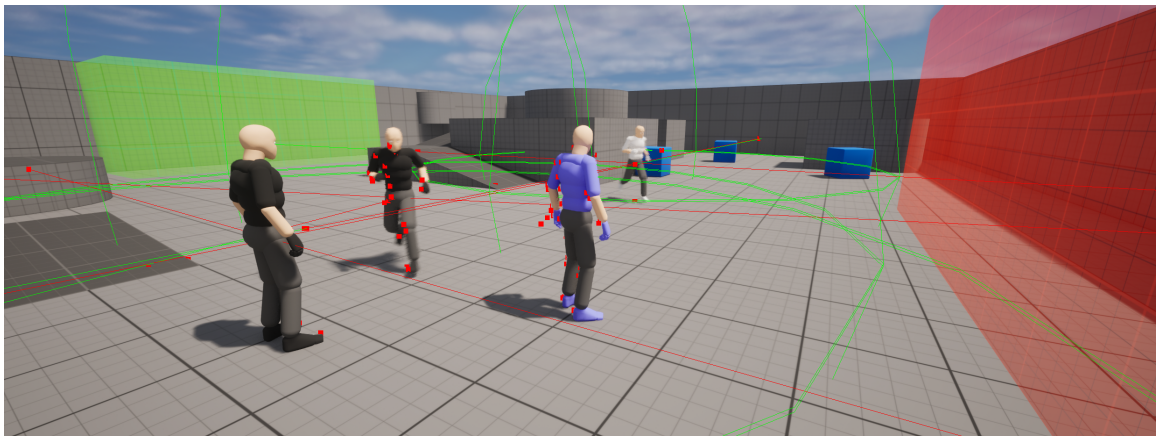

Figura 10: Vista Previa de los Sentidos

El comportamiento inicial solo contaba con correr, saltar y escalar como acciones. En cuanto a los sentidos, se detectaba la densidad de población y la distancia, así como una configuración mínima de la distancia a los límites. Los personajes comenzaron únicamente a moverse en círculos y no realizaban nada en particular. Por ello, se añadieron más sentidos y acciones como la detección de bordes, la ubicación actual basada en los límites y valores normalizados. Con esto se incrementó su proceso de razonamiento y comenzaron a navegar más; algunos no hacían nada, y otros saltaban si se detectaba una pared cercana o, por el contrario, si no la había. Resultó fascinante observar cómo comenzaron a desarrollarse más a medida que podían percibir y realizar más acciones.

Todo en Uno

Dado que la escala de este proyecto es bastante grande y que además muchos recursos pueden utilizarse para pruebas que no serán necesarios en el producto final, separé todos los componentes en distintos repositorios. Sin embargo, esto implica que en algún punto debe realizarse una fusión. Unreal Engine no es tan sencillo de manejar cuando se trata de fusionar distintos proyectos en uno solo, a diferencia de otras herramientas, por lo que fue necesario crear un sistema de fusión automatizado utilizando un archivo de cmake [4].

Este proyecto se encargará de la interfaz de usuario y de la comunicación entre todos los componentes creados sin afectar los scripts originales. Esto permite mantener la seguridad del código al aislarlo en diferentes repositorios, evitando así posibles rupturas.

Por qué CMake

Inicialmente pensé en crear un archivo bash para ejecutar la fusión entre los proyectos, pero esto habría limitado la configuración a sistemas operativos específicos. CMake, aunque requiere una instalación, permite que el proceso de ejecución sea multiplataforma, lo que lo convierte en una opción ideal.

[4] Para el proyecto fusionado, véase https://github.com/AndreM222/Tag-AI-Sandbox

Cómo Añadir

Para facilitar la configuración, decidí utilizar un archivo json para listar los proyectos que se desean fusionar, así como una opción de filtrado. Este sistema de fusión se encarga de eliminar archivos no importantes, como los de compilación y el script principal de los proyectos, y reemplaza todas las APIs por las del proyecto actual.

Evaluación

Trayectoria

Las entidades parecen terminar siguiendo un patrón de desplazarse hacia direcciones específicas hasta que logran tocar su objetivo o evitarlo. Sin embargo, se observa una mejora constante con el tiempo en su comportamiento y adaptación a diferentes entornos.

Intenté ejecutarlo bajo los mismos requisitos que *I programmed some creatures* [3] para comprobar qué ocurriría con una misión más básica, como dirigirse a áreas específicas del mapa. Al ejecutarlo, terminaron reaccionando de la misma manera. Esto puede deberse a que un juego de atrapar requiere sentidos como la vista, el tacto y el sentido del espacio para tomar decisiones más racionales.

Obstáculos

Dado que aún no se han realizado muchas pruebas, por ahora las entidades parecen caminar hacia las paredes en ciertos ángulos hasta que logran cruzar al otro lado si es necesario.

Se requerirán más pruebas para observar cómo mejora su comportamiento con el tiempo.

Qué Sigue

Continuaré desarrollando la estructura para la toma de decisiones y trasladaré el proyecto a la supercomputadora para comenzar a obtener datos mientras continúo implementando nuevas funcionalidades. También necesito desarrollar una interfaz de usuario que me permita obtener datos de los eventos, así como tomar capturas de pantalla en cada generación. Mientras las entidades se ejecutan en segundo plano, estaré trabajando en los comportamientos adicionales para implementarlos.

Para mejorar la toma de decisiones de las entidades, comenzaré a incorporar más sentidos. Esto mejorará su capacidad de decisión al poder elegir ubicaciones basándose en dónde estuvieron, lo que tocaron o lo que vieron, así como en lo que consideran interesante, peligroso o irrelevante.

El comportamiento de las entidades aún es primitivo y requiere mayor desarrollo para observar mejoras más significativas. Se necesitan iteraciones adicionales para analizar cómo interactúan con sus entornos y si logran aprender exitosamente a atrapar o escapar del objetivo.

Referencias

[1] R. A. Mossi and R. Sundaram, "Tag ai-sandbox," in *2025 ASEE North Central Section (NCS) Annual Conference*, Marshall University, Huntington, West Virginia: ASEE Conferences, Mar. 2025. [Online]. Available: https://peer.asee.org/54692

[2] OpenAI. "Multi-agent hide and seek," Accessed: Feb. 14, 2025. [Online]. Available: https://www.youtube.com/watch?v=kopoLzvh5jY

[3] D. R. Miller. "I programmed some creatures," Accessed: Feb. 14, 2025. [Online]. Available: https://www.youtube.com/watch?v=N3tRFayqVtk&t=2368s

[4] GeeksForGeeks. "A* search algorithm," Accessed: Feb. 14, 2025. [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/

[5] Jacob. "I rewrote my dungeon generator!" Accessed: Feb. 14, 2025. [Online]. Available: https://www.youtube.com/watch?v=NpS5v_Tg4Bw&t=200s

[6] GeeksForGeeks. "Breadth first search or bfs for a graph," Accessed: Feb. 5, 2025. [Online]. Available: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[7] R. Ree. "How to create epic procedural dungeons," Accessed: Feb. 14, 2025. [Online]. Available: https://www.youtube.com/watch?v=4ddbnQIuwAM

[8] Longmire. "Advance locomotion system v4," Accessed: Feb. 14, 2025. [Online]. Available: https://www.fab.com/listings/ef9651a4-fb55-4866-a2d9-1b38b028f9c7

[9] GeeksforGeeks. "Genetic algorithms," Accessed: Mar. 30, 2025. [Online]. Available: https://www.geeksforgeeks.org/genetic-algorithms/

[10] R. MacWha. "Evolving ais using a neat algorithm," Accessed: Mar. 30, 2025. [Online]. Available: https://www.geeksforgeeks.org/a-search-algorithm/