NavAgent AI

Autor: Raúl Beltrán Gómez C312.
Repo: https://github.com/rb58853/NavAgent-AI

Introducción

Agentes de navegación y sistemas multiagentes. Se entiende por agente una entidad capaz de percibir su entorno, procesar tales percepciones y responder a las mismas con una acción en su entorno, un agente de navegación es, a su vez, una entidad, además, capaz de navegar (moverse) en un espacio dado, el ejemplo más simple de un espacio de navegación sería una matriz de posiciones donde una entidad se mueve bajo determinadas reglas o movimientos validos; el agente debe ser capaz de moverse de una posición a otra de este espacio. En el presente proyecto definimos como espacio de navegación una malla de polígonos transitables que colindan a través de aristas, específicamente una malla de triángulos. Un sistema multiagentes es un sistema compuesto por múltiples agentes que interactúan entre sí, y tienen la capacidad de comunicarse entre ellos. Entre las características principales de un agente cabe destacar su comportamiento autónomo, su visión local del sistema y su descentralización, es decir que no existe una entidad que controle el comportamiento global de todos los agentes.

Estos sistemas multiagentes se llevan utilizando en la computación desde hace años para la simulación de multitudes, algunas de sus utilidades son la simulación de personas en ciertas circunstancias como en una evacuación, una terminal de ómnibus, etc. En los últimos años se han utilizado sistemas multiagentes en simulación de multitudes cinematográficas, como es el caso de El Señor de los Anillos. Específicamente en nuestro proyecto trataremos este sistema multiagentes y sistema de navegación de los mismos orientado al mundo de los videojuegos.

Objetivos del proyecto

Decidir y simular los movimientos óptimos de agentes de navegación que interactúan entre ellos y donde influyen condiciones del terreno, compatibilidad con los elementos de cada agente e interacción entre los mismos.

El objetivo es diseñar un sistema de navegación optimo que pueda ser utilizado como un Asset para posterior uso en vista al desarrollo de videojuegos, así como un sistema de multiagentes que interactúen de forma independiente al resto y sean capaces de navegar en situaciones grupales o individuales. Para el desarrollo del mismo se utiliza el motor gráfico Unity y lenguaje de programación C#, en el mismo que en un futuro se crearía un videojuego utilizando esta herramienta de navegación.

Movimiento

Para el movimiento de los agentes necesitamos de un espacio de navegación y ciertas reglas para moverse de una posición a otra. El espacio que usamos es una malla de navegación que nos proporciona el propio motor de Unity **[fig1]**, la malla de navegación de su componente NavMesh. El motor nos brinda como información de la misma un arreglo de flotantes y un arreglo de enteros que nos definen los vértices de cada triangulo. Esta malla a su vez la trasformamos en un grafo de triángulos creando un script para ello, de esta forma nuestro

espacio de navegación es ahora un grafo de triángulos donde se define que se transita de un triángulo a otro si y solo si estos triángulos son adyacentes.

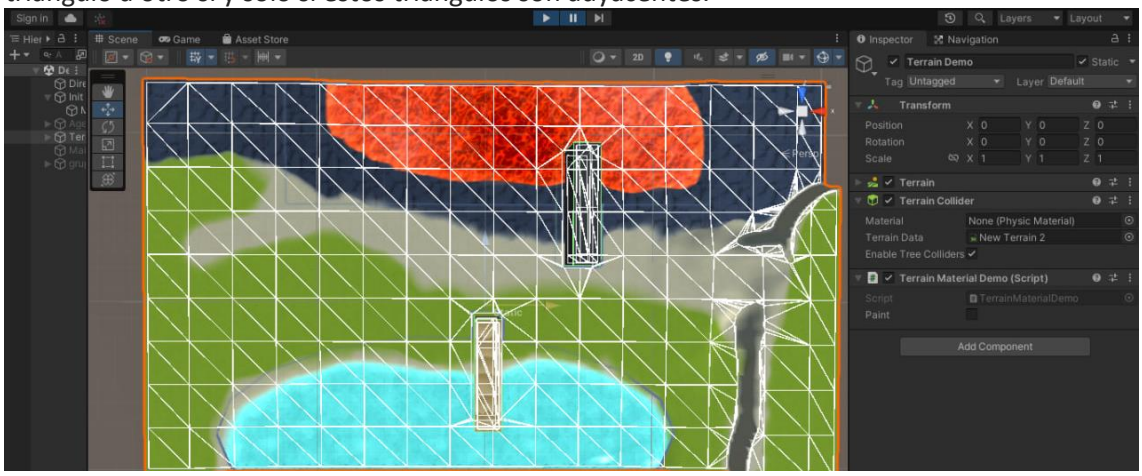

fig 1: Malla de triangulación de Unity NavMesh

Grafo de triángulos

Nuestro grafo de triángulos se define como una colección de triángulos que se relacionan entre si mediante adyacencia. La adyacencia de un triángulo a otro está dada a través de una arista de paso, es decir el lado que comparten ambos triángulos es la arista que define la adyacencia entre ellos. Algunas definiciones importantes sobre los nodos del grafo son las siguientes:

- Dictionary< MapNode, Arist > adyacents: los adyacentes de un nodo donde las llaves representan los nodos(triángulos) colindantes al mismo y los valores representan la arista por la cual colindan.
- Material material: Define el material que posee este polígono, este es de gran importancia para definir los caminos de los distintos tipos de agentes.

El recorrido, costo o distancia de un triángulo a otro está definido por la distancia euclidiana desde el punto más cercano de la arista de adyacencia a ambos baricentros hasta el baricentro de un triángulo multiplicado por el costo del material (depende del agente) más lo análogo con el otro triangulo. La distancia no es la exactamente la mejor, es un aproximado, aun mas teniendo en cuenta que los triángulos pueden tener cualquier tamaño y forma, todo depende de cuan compleja sea la malla de navegación.

Para crear el movimiento final del personaje se podría ir de un triángulo a otro pasando por el punto más cerca entre sus baricentros o por la proyección ortogonal de la posición actual a la próxima arista, pero estas soluciones dejan mucho que desear dado que nuestro objetivo es crear un sistema de navegación preciso y a su vez para múltiples agentes. La solución que se le dio a esto fue crear puntos de paso entre los triángulos subdividiendo las aristas en puntos, así moverse de un triángulo a otro sería equivalente a moverse por los puntos de sus aristas. Las ventajas de la misma son que se pueden subdividir tanto como queramos y así lograr un movimiento prácticamente en línea recta y a su vez la subdivisión puede ser suficientemente baja como para dejar una cantidad de puntos de paso pequeña lo cual implica menos cálculos de caminos.

En un primer momento se implementó búsqueda de camino simple, pero esto tenía como inconveniente que para recalcular un camino en caso de colisión habría que aplicar

nuevamente el algoritmo de búsqueda. Dado este problema se decidió calcular un camino de puntos donde cada punto posee el valor del costo de llegar hasta el nodo final, así en cada movimiento de un triángulo a otro el agente podrá seleccionar el próximo punto alcanzable al cual moverse. Utilizando un heap seleccionar dicho camino seria en el peor de los casos O (n Log n) donde n es la cantidad de adyacentes a la posición del agente, el mismo, como ya se menciona antes, estará condicionado por la cantidad de subdivisiones de las aristas. El algoritmo de búsqueda de caminos de divide en dos fases:

1. Calcular el camino de triángulos desde la posición final hasta la posición de inicio. Para ello se utiliza Dijkstra + A*, donde nuestra heurística es la distancia Euclidiana multiplicada por la forma más rápida de moverse que tiene un agente desde el baricentro de un triángulo hasta el punto objetivo (como vamos desde el final hasta el inicio entonces el objetivo es la posición). La heurística es admisible ya que nunca sobreestima el costo del camino **[fig2]**. ¿Qué pasa si la malla es excesivamente grande? para ello solo se le aplican estos algoritmos a un área limitada donde se debe asegurar que esté contenido el triángulo objetivo y el triángulo actual, lo cual representa la porción de mapa que ocupa el agente en ese momento. Al camino de triángulos seleccionado se le aplica un algoritmo para dilatarlo y así modificar el ancho del camino que quiero tener en cuenta, por ejemplo, un camino más ancho implica que habrá más posibilidades de movimiento, pero determinar cada distancia de cada punto de este camino será más costoso computacionalmente, y un camino más estrecho puede saturar los puntos de paso, pero el cálculo del mismo es más rápido. El ancho del camino se debe decidir en dependencia al tipo de escenario que se quiere crear, si el movimiento es grupal es conveniente que el camino sea un poco más ancho, si el escenario no contempla que abundaran estos tipos de movimiento con múltiples agentes entonces será mejor un camino que se calcule más rápido.

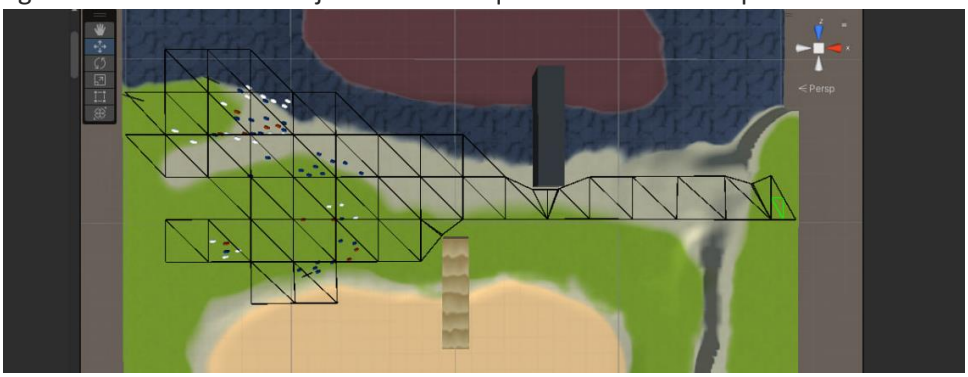

fig 2: Camino de triángulos seleccionado por un grupo de agentes hasta el mismo destino

2. Calcular la distancia exacta desde el nodo final a cada punto del camino de triángulos que se encontró en el paso anterior con Dijkstra, y usar esta distancia como heurística para el movimiento del agente, de esta forma tendríamos un movimiento que se basa por la heurística en su mayor parte y el costo de moverse hasta un punto adyacente que es un valor exacto, la misma es el costo exacto por lo cual tampoco sobreestima el costo del camino. Luego el agente se mueve según una heurística que no sobreestima el camino, que a su vez es completa ya que se

llega al destino, y es óptima ya que da el mejor camino de un punto a otro en el espacio analizado.

En cada punto de paso el agente selecciona cuál será su próximo punto de paso, si el mejor punto de paso próximo no es alcanzable entonces preguntará por el próximo. Si el agente no tiene puntos alcanzables se detiene y vuelve a preguntar por qué camino tomar luego de una pausa, a su vez se puede definir una tolerancia al agente para no seleccionar un camino que sea mucho más largo que el mejor camino posible, y en caso de que todos los puntos alcanzables no cumplan con esa tolerancia entonces detenerse. El Agente es también capaz de crear puntos dinámicos de paso en el terreno, estos puntos se crean cuando ocurre una colisión y el agente decide crear puntos para bordear el obstáculo, dicho sistema de creación de bordes no es adecuado para el movimiento grupal dado que en un área saturada de agentes estará creando estos puntos una y otra vez y el costo de hacer esto para dicho caso puede resultar inconveniente, pero para la navegación simple puede resultar muy interesante y crea opciones de camino mejores que las alcanzables en el momento.

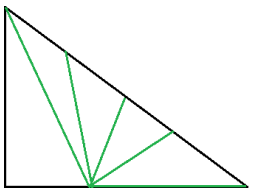

fig 3 Opciones de práximo punto de paso desde una arista a otra de un triángulo

Metaheurísticas

Usamos como metaheurística una idea de colonia de hormigas, pero en nuestro caso hay más de un tipo de hormigas, las feromonas que deja una no atrae a todas las demás, sino a las que deben estar interesadas por las mismas. En palabras más claras, para el movimiento grupal seleccionar un camino por cada agente puede resultar engorroso y muy costoso, este problema se resuelve utilizando metaheurísticas de forma tal que un agente selecciona un camino y los demás, si están dentro de ese camino lo siguen, en caso contrario crean otro camino hacia el punto final y lo unen (merge) con el camino que ya se obtuvo, así todos los agentes pueden utilizar el mismo camino y no es necesario recalcular el mismo por cada uno de estos. Pero como mencione anteriormente, nuestros agentes no todos poseen igual compatibilidad con los materiales del suelo, por lo tanto, que un agente seleccione un camino hacia un punto no implica que otro agente cualquiera decida que ese será un buen camino para él, para ello la metaheurística consiste en crear caminos para cada tipo de agente, y hacer merge solo con los caminos que son del tipo del mismo .Así por ejemplo: un agente tipo acuático selecciona un buen camino hasta cierto punto y otro acuático selecciona camino distinto entonces el camino de metaheurística ahora será la unión de los dos anteriores. Se puede definir a cada agente que con una variable aleatoria decida si crear un nuevo camino o tomar uno ya existente en caso de poder hacerlo, pero en la configuración default del proyecto esto no viene incluido. Ahora si un agente de tipo fuego selecciona un camino hacia el mismo lugar que un agente de tipo agua, los caminos de estos agentes de distinto tipo no se unen ni será un camino utilizable por un agente distinto. En la imagen **[fig4]**, se puede apreciar como el grupo de agentes posee tres tipos de agentes y los mismos toman caminos según su

compatibilidad con el terreno y a su vez cada tipo de agente tiene su camino de metaheurísticas definido.

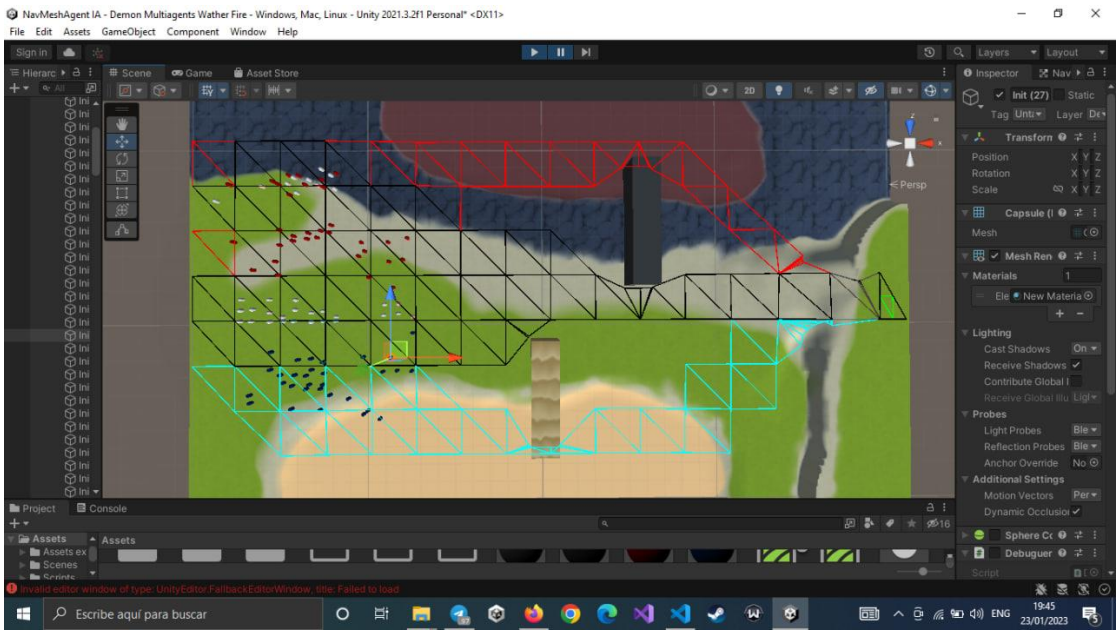

Fig 4 Seleccion de caminos por agentes de distinto tipo

¿Como funciona el Merge? La unión de dos caminos está dada por mantener el primer metacamino seleccionado y unirle el otro, actualizando la distancia de cada punto que coincida con él la menor de las distancias. Se unen creando adyacencias desde aristas nuevas a aristas locales (que ya existen en el camino meta), eliminando adyacencias a aristas que quedaran obsoletas, y agregando y eliminando adyacencias sobre triángulos nuevos, locales y obsoletos. De esta forma el camino crece, crecen las opciones de llegar al punto final, aumenta la posibilidad de encontrarse sobre un triángulo que pertenece al camino para así no tener que calcularlo, crece las opciones de recorrer el camino, y el costo computacional siempre será el mismo, que es seleccionar el mejor próximo punto de paso.

Movimiento Grupal

El movimiento grupal **[fig 5]** se basa en los aspectos antes mencionados, cada agente se mueve de forma independiente y tomando sus propias decisiones, pero al moverse en grupo esto trae consigo que los agentes de su propio grupo le calculen, si está lo suficientemente cerca, un camino para el también. De esta forma un agente de grupo busca camino desde el destino final hasta los triángulos donde se encuentran sus compañeros, si estos compañeros están muy lejos no crea este camino por lo de antes mencionado de área reducida, un agente busca camino en área reducida. No obstante, esta área es maleable desde las configuraciones del Environment y se puede hacer lo suficientemente grande como para que alcance un número considerable de agentes del grupo, incluso infinita (int.maxValue) para alcanzar toda la malla. En caso de no alcanzar a todos los agentes otro agente repetirá el proceso y así hasta que todos estén ubicados en un camino hacia el punto final. Este movimiento grupal depende directamente de la metaheurística para que los agentes seleccionen un camino ya existente.

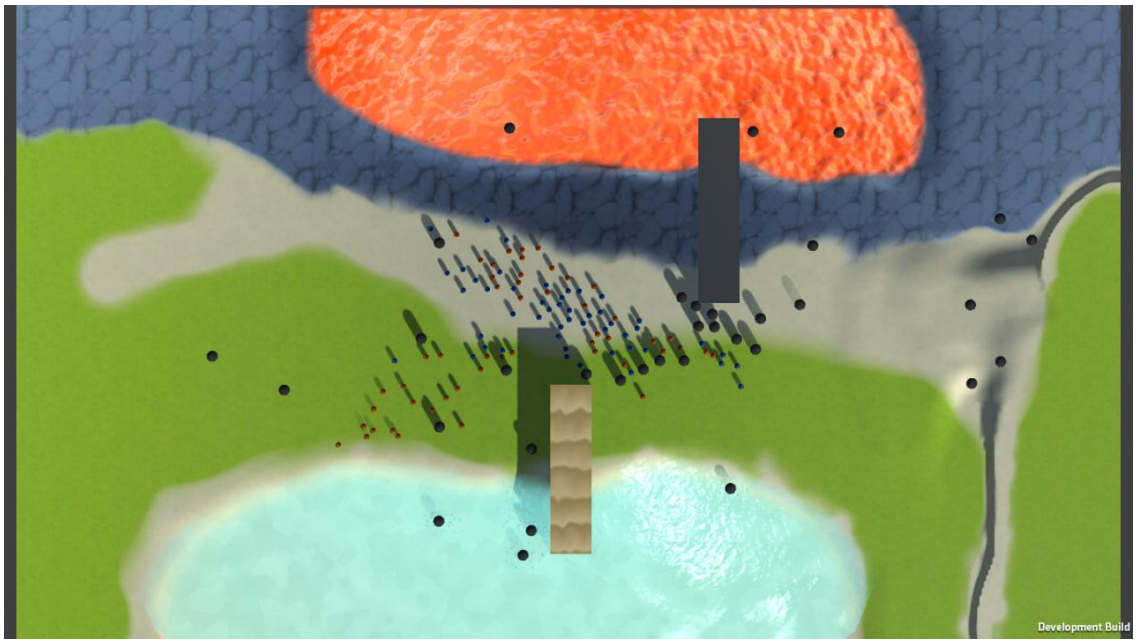

fig 5 Ejemplo de movimiento grupal

Simulación

La simulación está basada en agentes inteligentes capaces de tomar decisiones al percibir su entorno y realizar acciones autónomas en base a ello.

Comportamiento reactivo. Perciben una vecindad del terreno y sus cambios y deciden cómo moverse en consecuencia, en dependencia además de las características específicas de cada tipo de agente. Los agentes se dividen en tipos, cada tipo de agente tiene cierta compatibilidad con materiales del terreno, por ejemplo, los agentes acuáticos tienen buena compatibilidad con el agua y mala con el fuego. En dependencia a estos materiales del ambiente son capaces de seleccionar sus movimientos.

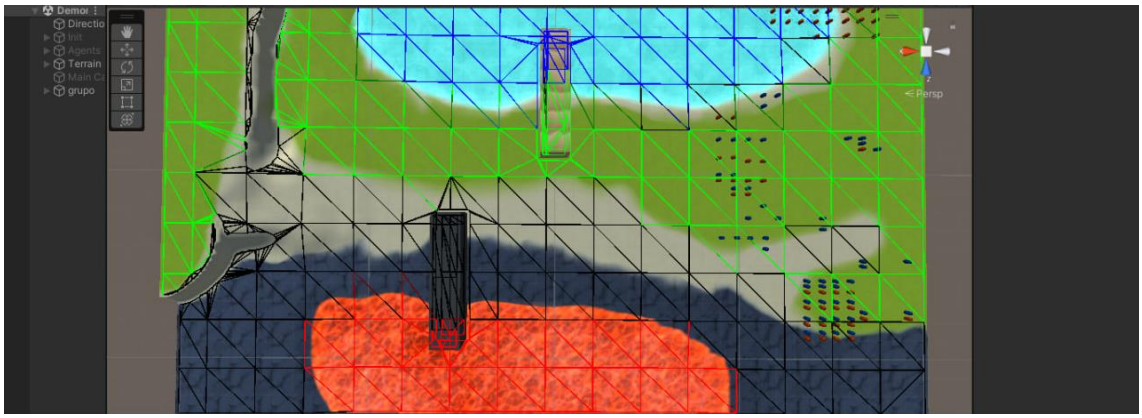

fig 6 Materiales del terreno

Detección de colisiones. Para la detección de colisiones los agentes tienen definido una distancia de vista**[fig7]** y una distancia en la cual analizara si hay colisión (radio de análisis). Si un agente detecta una colisión en su camino cambiará su próximo punto objetivo. Revisan colisiones cada cierta cantidad de movimientos, esto es configurable desde el Environment, el valor default es, grosso modo, entre 10-20 revisiones por segundo. Poseen cualquier tamaño

de radio y son capaces de detectar colisiones independientemente del radio que tenga el agente con el que está próximo a colisionar. Si dos agentes colisionan, lo cual no suele suceder, pero en dependencia de las revisiones por paso puede darse el caso, estos agentes chocan y se separan uno del otro hasta quedar suficientemente separados, simulan un choque.

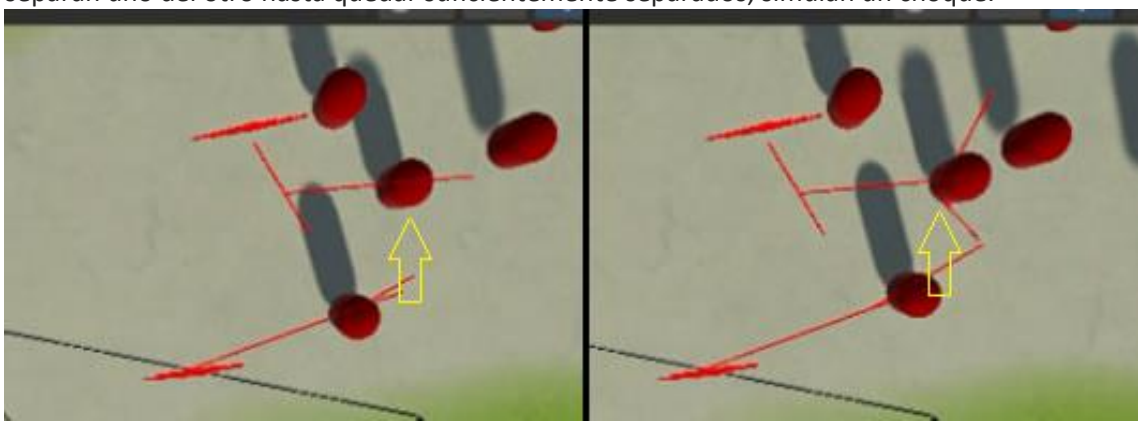

fig 7 las líneas rojas son la dirección en la que está mirando cada agente, se pintan en cada iteración y no se borran.

Son capaces de crear caminos dinámicos, puntos de paso de forma independiente, de tal forma que si perciben una colisión crean un camino para desviarse sin que ninguna entidad les proporcione este. Se puede decir que los agentes son capaces de crear sus propios caminos independientemente de los puntos predefinidos en la búsqueda de caminos, lo cual implica que hay infinitas posibilidades de llegar de un punto a otro creando puntos dinámicos en un espacio continuo.

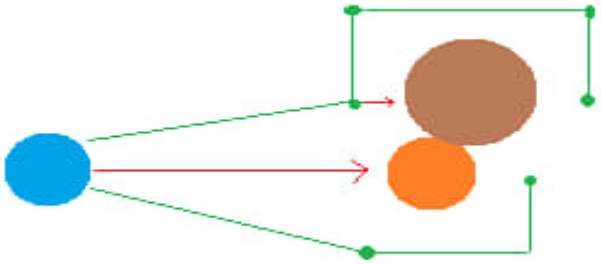

Figure 8 Al encontrar colisión crea puntos dinámicamente que bordean la misma.

Comportamiento proactivo. Se mueven con comportamiento dirigido a llegar a la posición especificada de forma óptima **[fig4]**. Toman decisiones dado el camino precalculado, el cual depende directamente del agente, o más bien del tipo de agente que crea el camino. De esta forma un agente de tipo fuego tomará un camino distinto a un agente de tipo agua desde y hasta iguales posición y destino.

Poseen un factor de tolerancia que usan para decidir qué tan dispuestos están a sacrificar camino corto para salir de un atasco o posición de parada (Stop), mientras más tiempo pasen atascados más tolerantes se vuelven a tomar un camino malo **[fig 9]**.

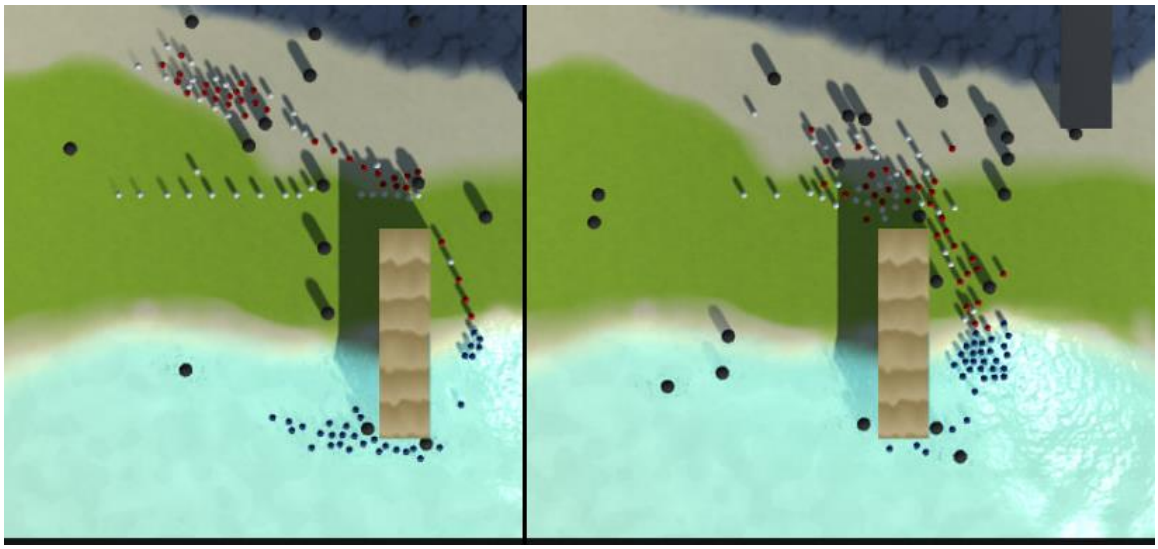

fig 9 izquierda tolerancia baja, derecha tolerancia alta

Futuros Pasos

El proyecto es extensible a muchas ideas, nuevas funcionalidades y mejoras que no fueron implementadas por falta de tiempo. A continuación, se presentan algunas de estas:

- Crear mundos de forma procedural con variables aleatorias, semillas, así como simular cambios estacionales, eventos climatológicos y con ello alterar los materiales de los triángulos: hojas en el suelo, la hierba crece, el agua se seca, etc.

- Crear un sistema de batalla con utilizando Min Max o Machine Learning. O más bien utilizar la herramienta creada para crear un juego de batallas.

- Crear mapas utilizando procesamiento de lenguaje natural para que así usuarios comunes creen sus mundos. Por ejemplo, definir que haya nieve, montanas, un rio, arboles altos, etc.

- Utilizar ejecución en paralelo para ejecutar operaciones en distintos hilos, de esta manera la CPU pasará a la tarea de procesar y el rendimiento será mucho más optimo.

- Mejorar el sistema de traslación visual en Unity Engine suavizándolo y evitando cierto desfase.

- Mejorar el sistema de Bordeo de colisiones a nivel de triángulos externos al actual sin recalculando nueva posición. Dado que si bordeando cambias de un triangulo a otro los adyacentes de los puntos dinámicos no cambian y son los mismos que tenia en su triangulo original. Por ello se evita utilizar el bordeo fuera del triángulo.