BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

UNA PROPUESTA DE UN MOTOR DE COMPORTAMIENTOS GRUPALES PARA EL DESARROLLO DE VIDEO JUEGOS

TESIS QUE PARA OBTENER EL TÍTULO DE:

LICENCIADA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA: GLORIA IVONNE MONARCA PINTE

ASESOR: DR. ABRAHAM SÁNCHEZ LÓPEZ.

MARZO 2016

RESUMEN

La habilidad para simular el movimiento coordinado de un grupo de criaturas geométricas (robots, personajes virtuales,…) juega un papel importante en la vida artificial y en los video juegos. La representación artificial de tales criaturas requiere técnicas para generar el movimiento de entidades individuales dentro del grupo y técnicas para dirigir el movimiento global del grupo.

Desafortunadamente los métodos existentes no se adecuan correctamente, si se requiere de una navegación compleja. Dada la posición inicial y final de un carácter en un ambiente virtual, y la secuencia de movimientos para mover al carácter virtual en dicho ambiente, nuestro objetivo es encontrar una secuencia de movimientos para un grupo de caracteres, es decir, realizar una animación de comportamientos grupales. Esto nos lleva a considerar fuertemente la necesidad de tener dentro del trabajo, una funcionalidad especializada de planificación de movimientos para caracteres virtuales.

La planificación de movimientos de objetos tiene múltiples aplicaciones en muchas áreas tales como la robótica, sistemas de realidad virtual, diseño asistido por computadora, etc. Aunque se han propuesto muchos métodos de planificación de movimientos, la mayoría no son usados en la práctica, dado que no son factibles computacionalmente hablando (algunos de ellos sólo han servido para estudios teóricos).

Para realizar esto, investigaremos como la inclusión de la información global en forma de un roadmap del ambiente nos permitirá obtener comportamientos de agrupamientos más sofisticados y esto nos ofrecerá una mejor navegación global y planificación.

La propuesta del presente trabajo será desarrollar un motor de comportamientos grupales que contenga funcionalidades propias a los expertos en el desarrollo de video juegos, la idea es utilizar las funcionalidades que nos proporciona una herramienta muy utilizada para el desarrollo de video juegos, como lo es Unity, y la posibilidad de usar scripts. Se pretende dotar de algunos de los comportamiento grupales básicos al motor.

CONTENIDO

ÍNDICE DE FIGURAS

Capítulo 1 INTELIGENCIA ARTIFICIAL EN LOS VIDEOJUEGOS

La inteligencia artificial (IA) es un área multidisciplinaria, que a través de distintas áreas como las ciencias de la computación, la matemática, la lógica y la filosofía, estudia la creación y diseño de sistemas capaces de resolver problemas cotidianos por sí mismas utilizando como paradigma la inteligencia humana.

En 1956, John McCarthy define la Inteligencia Artificial como "La ciencia e ingenio de construir maquinas inteligentes" [1]

Algunas áreas de investigación típicas de Inteligencia Artificial son: aprendizaje, representación de conocimiento y razonamiento, sistemas multi agente, procesamiento de lenguaje natural, gestación de planes y agendas, robótica, búsqueda, reconocimiento de imágenes, optimización, etc.

En el último siglo la comunidad científica de Inteligencia Artificial (IA) se ha interesado por el desarrollo de juegos de entretenimiento interactivo digital. Prueba de ello es la creación de varios congresos sobre IA en videojuegos, [2] [3], libros sobre el tema [4] [5] [6] y grupos de investigación que se encuentran en algunas universidades de países como Canadá y Holanda.[7][8]

La comunidad de IA ve a los videojuegos como campo de exploración y la comunidad de videojuegos demanda mayor innovación en IA.

En videojuegos, la IA se refiere a las técnicas utilizadas para producir la ilusión de inteligencia en el comportamiento de los personajes no jugadores (NPCs o Non-player character). Los primeros intentos en crear comportamiento inteligente se basaron en juegos como el ajedrez.

Actualmente existe una gran variedad de juegos, acción, RPG, aventuras, juegos estratégicos, deportes en equipo, deportes individuales y. Estos juegos necesitan cierta inteligencia para ser más realistas y mejorarla experiencia del usuario.

La IA se usa principalmente en los enemigos, para hacerlos navegar por el mama y/o terreno, ocultarse y percibir la cercanía del jugador, saber cuándo y cómo perseguir al jugador, adaptar estrategias dinámicas. La IA también se usa en las unidades para hacerlas mover en formaciones inteligentes y reaccionar adecuadamente ante ataques inesperados.

1.1 Breve historia de la IA en los videojuegos.

Aunque no lo parezca, la Inteligencia Artificial tiene una relación antigua con los juegos. En los años 50 se aplicaron los primeros sistemas de IA en juegos. En 1950 Claude Shannon presento junto al economista y matemático D. G. Champernowne un artículo llamado "Programming a Computer for Playing Chess"[9], en donde especificaban las primeras técnicas y algoritmos necesarios para crear un programa de ajedrez. En 1959 Arthur Samuel programo un juego de damas en el que se implementaba por primera vez un tipo de IA basado en el autoaprendizaje. [10]

Estos juegos fueron el punto de partida de la IA en los videojuegos, permitiendo en años posteriores aparecieran juegos basados en la lógica, como fue el caso del Pong (Figura 1.1) o Space war(Figura 1.2), en los años 60.

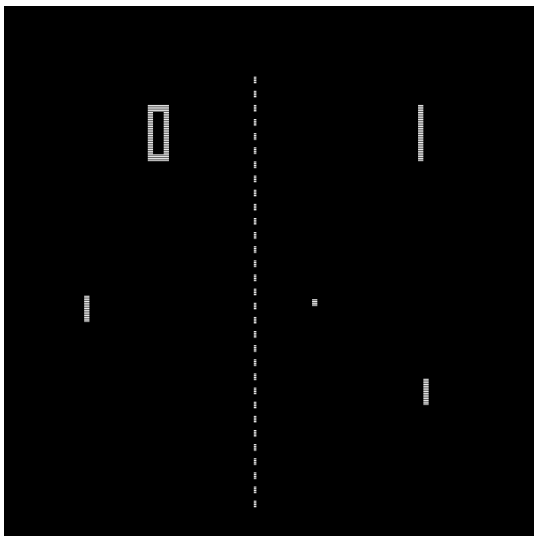

Figura 1.2 Pong

Figura 1.1 Space War

Con los avances tecnológicos a finales de los años 70 surgieron juegos de 1 jugador contra enemigos que se movían aleatoriamente mediante patrones almacenados.Dos juegos destacan de estos años: Space Invaders(1978) y Galaxian(1979), (Figura 1.3 Juegos 70),que añadieron complejidad al juego introduciendo funciones hash que "aprendían" de los actos del jugador.

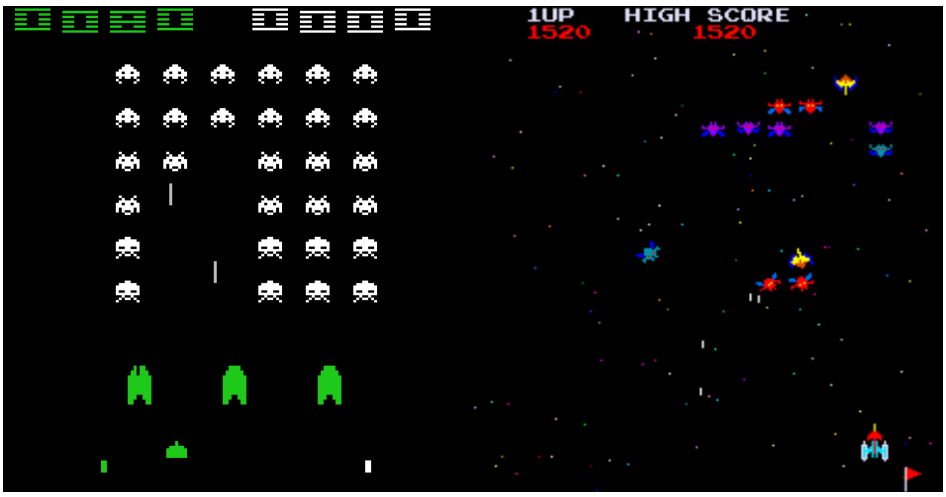

Figura 1.3 Juegos 70. En la izquierda se muestra el juego de Space Invaders y a la derecha Galaxiann

En los 80 surgió, Pac-Man, quien además de incorporar algoritmos de búsqueda en videojuegos, junto con Kárate Champ, introdujeron el concepto de personalidad en los enemigos mejorando la experiencia de juegos. Madden Football es otro juego destacable, pues intento repicar el comportamiento de los jugadores más importantes de la liga.

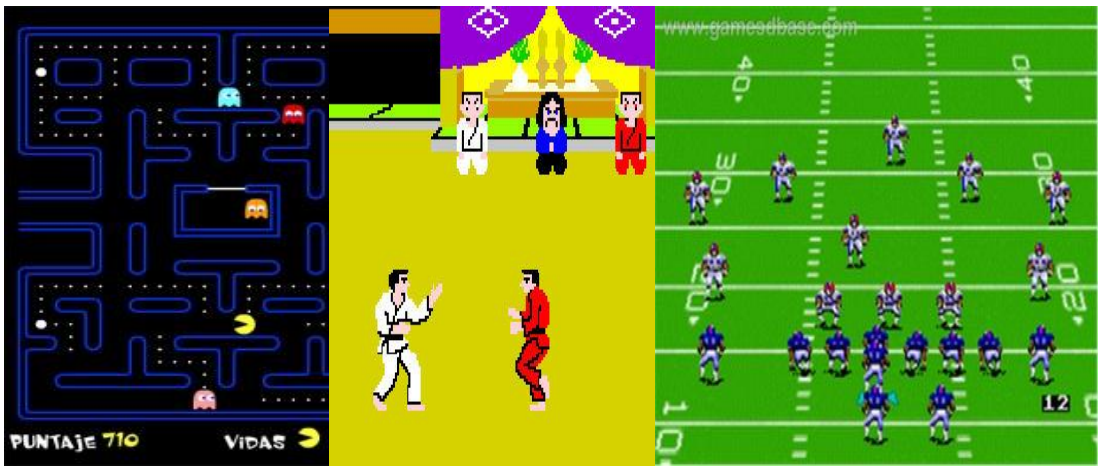

Figura 1.4 Juegos 80's. En la primera imagen se muestra Pacman, en medio Karate Champ y a la iaquiera Madden Football

Sin embargo fue hasta los años 90, que se dio el verdadero éxito. Gracias a la aparición de numerosas consolas de enorme calidad, se introdujeron las máquinas de estados, las decisiones en tiempo real, la toma de caminos, el aprendizaje, etc.

Se desarrolló el primer RPG, Dragon Warrior (1990) que permitía variar las rutinas de IA de los enemigos durante las batallas. GoldenEye 007 fue el primer FPS(first-person shooter), el cual introdujo una IA que reaccionaba en tiempo real a las acciones y movimientos de los jugadores. A finales de los 90 Half-life introdujo el concepto de cooperación entre enemigos para cubrirse, buscar al jugador, flanquear al adversario,etc.

Figura 1.5 Juegos 90's. Del lado derecho tenemos a Dragon Warrior y del izuiqero a GoldenEye

A principios del siglo XXI se lograron más avances de IA en los videojuegos. Age of Empires(AOE), serie juegos de estrategia en tiempo real, Halo (2001) que introdujo en los enemigos el reconocimiento de amenazas. Far Cry (2004) utilizo métodos de aprendijzaje para adivinar el comportamiento del jugador y emplear tácticas militares contra él. Fear(2005) incluyó por primera vez planificación en tiempo real usando GOAP (Goal-Oriented Action Planning) el cual le permitía adaptarse de forma dinámica a los sucesos del entorno.

Figura 1.6 Juegos del siglo XXI. En la primera imagen se puden observar algunas formaciones de Age of Empires.. En la imagen de en medio se muestra Far Cry. Y por último tenemos Halo uno de los juegos mas populares y con una gran IA desde entonces

Uno de los videojuegos que más avanzo en la IA al incluir elementos de vida artificial y estrategia en un juego donde el jugador actúa como un Dios sobre esa vida artificial inteligente, es Black and White.

Este juego tuvo grandes logros dentro de la IA como la introducción de la creación de vida artificial en el contexto de la estrategia del juego, la introducción de una arquitectura de inteligencia artificial realmente sólida que tiene sus raíces en la ciencia cognitiva conocida como belief-desire-intention (BDI) ý sobre todo la introducción de los árboles de decisión y las redes neuronales con un gran éxito.

Figura 1.7 Black and White, un juego con un gran nivel de IA

Uno de los 10 juegos más influyentes dentro de la IA [11], también surgió a principios de este siglo: Los sims.

Los sims es un juego de simulación de las actividades diarias de una familia o personajes individuales. El jugador puede construir una casa y guiar a su sim a través del día. Los sims también son capaces de tomar decisiones por ellos mismos, y dependiendo de sus necesidades su humor puede cambiar, interactúan con otros sims.

En conclusión muchos juegos han dado una buena muestra de lo que una inteligencia artificial puede lograr, las nuevas tecnologías y dispositivos de mayor capacidad han permitido mejorar la IA de NPCs con comportamientos más humanos, y juegos más versátiles. Con esto se ha logrado una mayor inmersión del jugador en el videojuego y se evita el aburrimiento del mismo.

Figura 1.8 Los Sims. En los Sims cada personaje tiene una personalidad diferente, y tienen cierta autonomía para cumplir sus necesidades cuando el jugador las ignora.

1.2 TIPOS DE IA EN JUEGOS

La IA aplicada en videojuegos se puede agregar en tres tipos :

- **Hacks**

  Se refiere a los efectos puntuales, no representan realmente una técnica de Inteligencia Artificial. Esta opción está basada en el aprendizaje de la inteligencia humana para luego crear una máquina que lo replique. Por ejemplo los juegos de ajedrez no se basan en técnicas que simulen comportamientos humanos, sino en algoritmos que buscan la mejor jugada mirando en una cierta cantidad de movimientos futuros. Un videojuego que hace uso de esta técnica es PacMan, en sus fantasmas.

- **Heurísticas**

  Son las técnicas o reglas aplicadas que pueden ayudar para resolver un problema puntual. Este tipo de IA se basa en que los personajes de un videojuego realicen acciones que en el pasado les han resultado beneficiosas en una determinada situación. La búsqueda es terreno habitual de heurísticas. Un ejemplo es el atacar la unidad enemiga más débil.

- **Algoritmos**

  Aunque los dos tipos de IA mencionados anteriormente son implementados mediante algoritmos, con esta denominación nos referimos a aquellas técnicas, en donde se aprecia una inteligencia más "computacional". Son algoritmos que sustentas comportamiento inteligente, basados en técnicas generales aplicadas en varios escenarios. Como el pathfinding, planificación, análisis de terrenos, movimientos coordinados en grupos, etc.

1.3 Necesidades básicas de IA en videojuegos modernos.

La IA en gran cantidad de juegos modernos satisface tres necesidades básicas: la capacidad de mover personajes, de tomar decisiones acerca de hacia dónde moverse y la habilidad de pensar táctica o estratégicamente.

En la Figura 1.9 podemos observar un modelo de IA en los videojuegos en donde se utilizan estas tres necesidades.

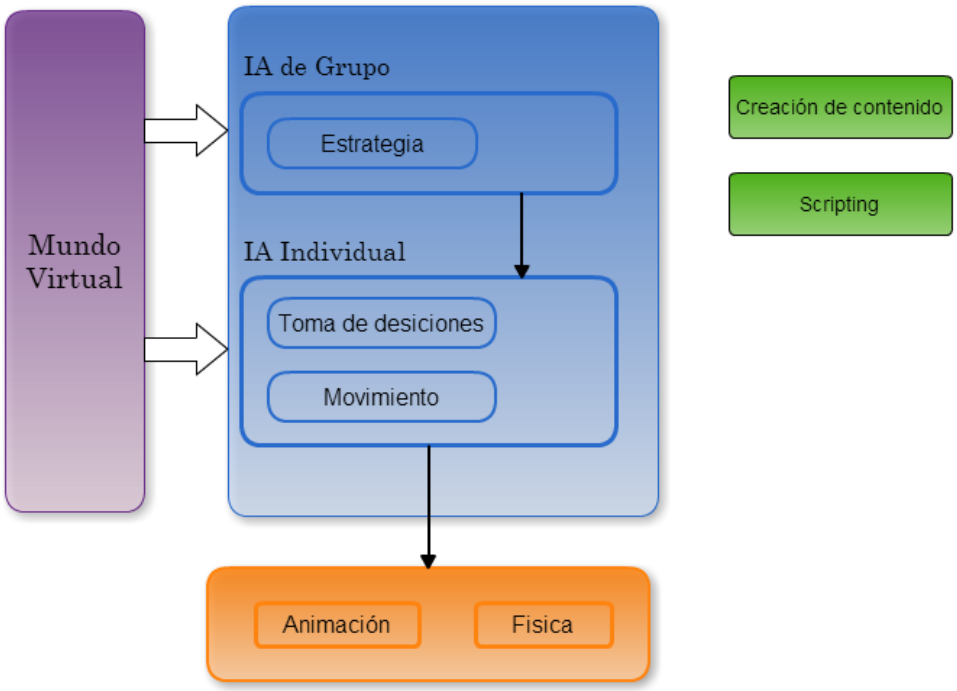

Figura 1.9 Necesidades básicas de la IA

El modelo se separa en tres secciones: movimiento, toma de decisiones y estragegia. Las dos primeras secciones contienen algoritmos que corren en cada personaje por separado, mientras que la ultima puede ser para un grupo de personajes.

Aunque no siempre es necesario utilizar las tres secciones. Por ejemplo, en un juego de ajedrez, solo se requiere el nivel estrategico. Las figuras no tienen que preocuparse por hacer sus propiosas desciones individualmente, ni tampoco acerca de como moverse, ya que existen reglas establecidas.

A continuacion se explicara cada una de estas secciones.

1.3.1 Movimiento

Esta sección se refiere a los algoritmos y técnicas para el movimiento de personajes (moverse, buscar, perseguir, esquivar, pasear, etc). Se considera al movimiento el aspecto más importante de un videojuego.

Una buena parte de la actividad científica se centra en aspectos de movimiento:

- Técnicas de búsqueda de caminos.
- Técnicas de generación de waypoints de navegación.
- Movimientos coordinados de grupos de unidades.
- Movimientos de multitudes(crow).
- Movimientos creíbles
- Planificación de movimientos.

Todos los algoritmos de movimiento tienen la misma forma básica: toman datos acerca de su estado y el estado del mundo y obtienen una salida que representa el movimiento que harán. La siguiente figura muestra un esquema de esta idea

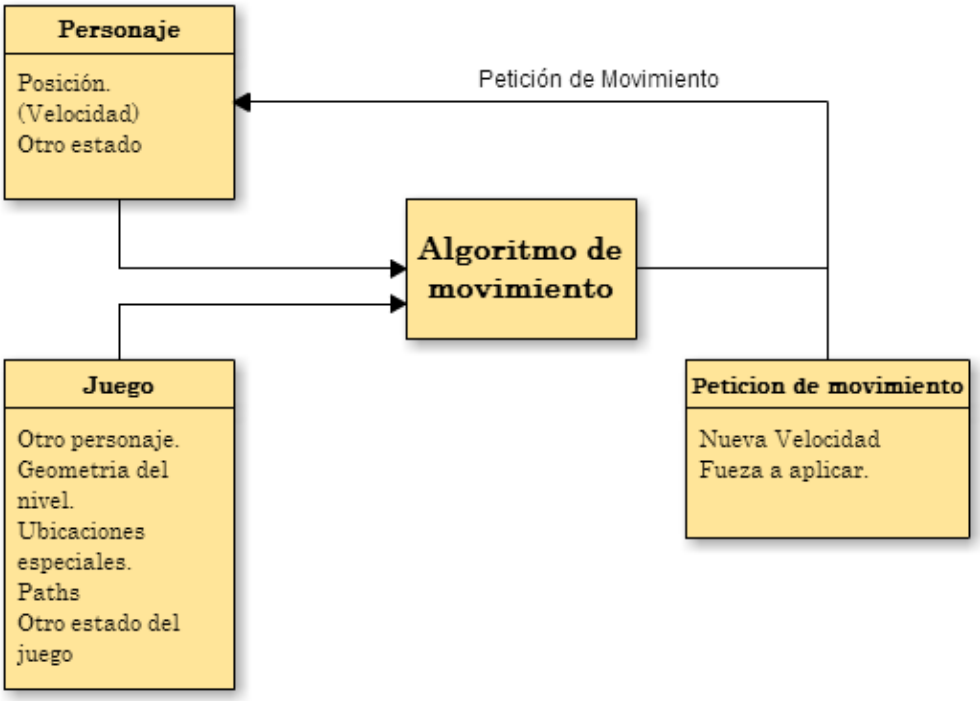

Figura 1.10 Esquema de movimiento

En este esquema la velocidad se muestra como opcional debido a que solo se necesita para determinados algoritmos de movimiento.

1.3.2 Toma de decisiones

La toma de decisiones permite a un personaje saber que es lo siguiente que tiene que hacer, cada personaje tiene un rango de diferentes comportamientos que pueden realizar: atacar, quedarse quietos, esconderse, explorar, patrullar, etc. El sistema de toma de decisiones tiene que decidir cuál de estas posibilidades es la más apropiada para cada momento del juego.

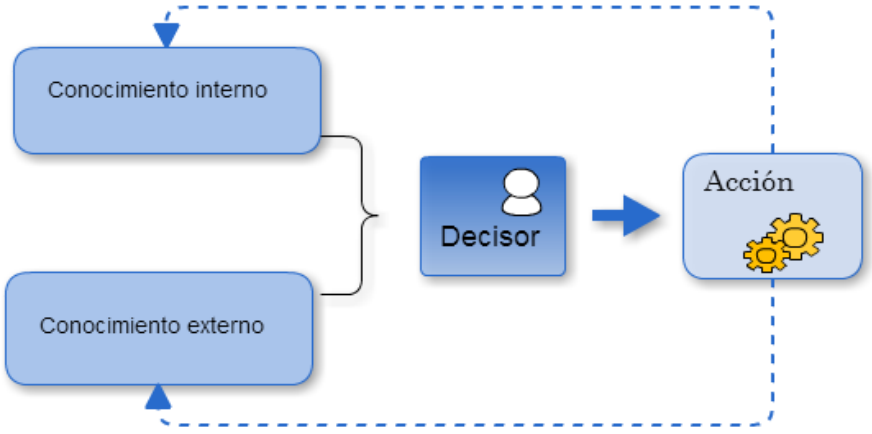

Figura 1.11 Toma de decisiones

Existen varias técnicas clásicas para implementar la toma de decisiones:

- Arboles de decisión: Son estructuras de conocimiento que permiten asociar acciones con grupos de piezas de conocimiento externo e interno.
- Máquinas de estados : Se utiliza para modelar el comportamiento basado en estados de actividad.
- Lógica difusa : Permite modelar propiedades difusas, es usada en varios videojuegos.
- Arboles de comportamiento : Son estructuras jerárquicas de tareas, cada tarea realiza las computaciones pertinentes, las tareas pueden ser simples o compuestas.
- Comportamiento basado en metas : Las acciones del agente se orienten al cumplimiento de ciertas metas, esto se denomina Goal Oriented Behavior (GOB)

1.3.3 Estrategia

La estrategia se refiere a la forma en la que un grupo se acerca o avanza hacia sus objetivos o enemigos. Para lograr esto existen algoritmos que influencian el comportamiento de grandes grupos de personajes. Cada personaje del grupo puede tomar decisiones por si mismo y llevar a cabo algoritmos de movimiento, pero su toma de decisiones será influenciada por la estrategia de grupo.

Para implementar este tipo de IA se pueden usar arboles de decisión o máquinas de estado. En la actualidad se ha apostado por el uso de algoritmos evolutivos, debido a que estos algoritmos no son deterministas y de esta forma los juegos pueden adoptar infinidad de formas de jugar dando una experiencia más rica y variada a los jugadores.[18]

Como se mencionó antes no es necesario usar las tres secciones de IA, con el movimiento y la toma de decisiones se pueden realizar muchas cosas. Los juegos que más usan esta capa son los que entran dentro de la categoría con el mismo nombre, "Estrategia". Hoy en día son uno de los estilos más jugados e importantes dentro del mundo de los videojuegos. Permiten muchas posibilidades tales como juegos online, múltiples bandos aliados o enemigos combatiendo a la vez. Un ejemplo es el "Age of Empires".

1.4 APORTACIONES

En este proyecto de tesis, se propone la creación de un motor de comportamientos, que mediante diversos scripts, pueda ser incluido en los próximos trabajos de videojuegos, bajo Unity 3D. Estos comportamientos podrán ser usados en videojuegos donde existan grupos de personajes NPCs.

Como se dijo antes el movimiento de personajes es uno de los aspectos más importantes de un videojuego. Un aspecto de esta área es la planificación de movimientos. Los NPCs deben planificar los movimientos entre lugares del mundo virtual, actualmente esto se logra usando algoritmos como A*. En nuestro caso implementaremos una técnica de planificación de movimientos desarrollada en el área robótica, PRM.

Igualmente se implementaran tres comportamientos básicos, que conformaran al motor. Estos comportamientos son : autoguiado, siguiendo al líder y búsqueda de un objetivo.

Este documento está compuesto por cinco capítulos, en el capítulo 2 se habla sobre los comportamientos para caracteres autónomos, y cuáles son los principales comportamientos. En el capítulo 3 se aborda el tema de los Roadmap probabilistas, que son y cómo se construyen. En el capítulo 4 se explica cómo se implementó nuestra propuesta, así como la herramienta que usamos para desarrollarla. En el último capítulo se muestran algunos resultados experimentales y los trabajos futuros a seguir de este trabajo.

Capítulo 2 COMPORTAMIENTOS PARA CARACTERES AUTÓNOMOS

Otro aspecto importante en un videojuego es el comportamiento de los personajes no jugables, para ello hablaremos del concepto de personajes autónomos presentado por Crayg Reynolds presentó en 1999. [12]

Un personaje autónomo 3D, es un tipo de agente, es decir, es una entidad capaz de percibir su entorno, procesar tales percepciones y tomar acciones autónomas de acuerdo al estado del entorno para cumplir sus objetivos de diseño. Es capaz de percibir su medioambiente con la ayuda de sensores y actuar en ese medio utilizando actuadores (elementos que reaccionan a un estímulo realizando una acción).

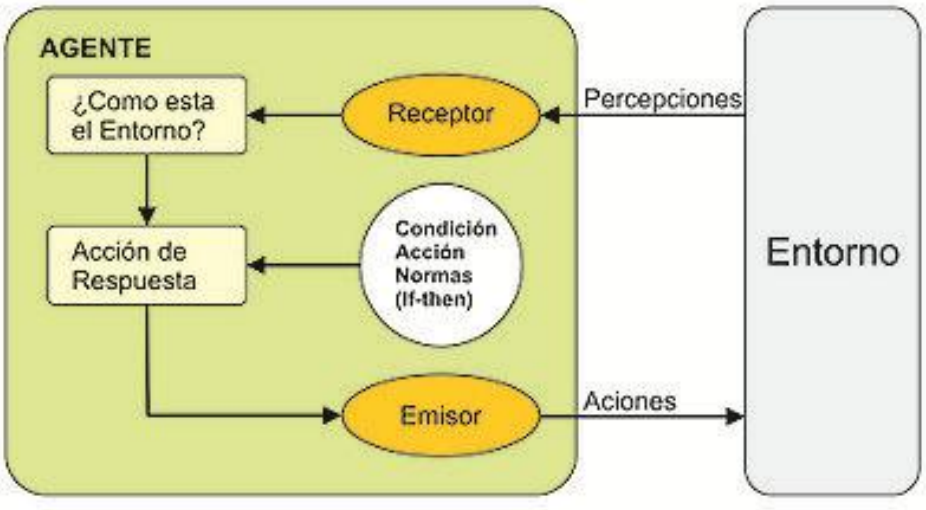

Figura 2.1 Agente

Los entornos virtuales 3D han permitido el desarrollo de agentes 3D omniscientes, estos agentes se desenvuelven en entornos virtuales 3D, un ejemplo son los ambientes de simulación. Estos agentes poseen características propias de los agentes inteligentes, y los entornos virtuales:

- Habitan dentro de un entorno de ejecución 3D.
- Posees una representación gráfica 3D dentro del mundo que habitan y son capaces de percibir, adaptarse y reaccionar a su entorno.
- Son capaces de expresar comportamientos de manera gráfica como un ser vivo.

- Aunque solo existen y funcionan en un entorno especifico, son conscientes de los cambios que se producen a su alrededor y son capaces de responder de manera autónoma.

2.1 Caracteres autónomos.

Los caracteres autónomos son un tipo de agentes autónomos dirigido a la animación por computadora y medios de comunicación interactivos tales como juegos y realidad virtual. Estos agentes representan a un carácter en una historia o juego y tienen algunas habilidades para improvisar sus acciones.

Un carácter autónomo debe combinar aspectos de un robot autónomo con algunas herramientas de un actor humano en un escenario improvisado. Estos caracteres usualmente no son robots reales, y ciertamente tampoco actores humanos, pero comparten características de ambos.

Los caracteres autónomos tienen la habilidad de navegar alrededor de su mundo como en la vida real y con problemas inesperados. Estos comportamientos de manejo o de conducción son en gran parte independientes de los datos usados en las formas de movimiento del carácter, para lograr estos objetivos se utilizan combinaciones de ambientes direccionales (por ejemplo: irse de aquí hacia allá mientras se evitan obstáculos, seguir el pasillo, unir un grupo de caracteres).

El termino comportamiento tiene muchos significados. Puede significar la acción compleja de un humano o de otro animal basada por voluntad propia o instinto. También puede significar en gran parte acciones preestablecidas de un simple sistema mecánico, o la acción compleja de un sistema caótico.

En realidad virtual y aplicaciones multimedia, esto algunas veces es usado como un sinónimo para la animación. En este trabajo utilizaremos el término comportamiento para referirnos a acciones improvisadas y similares a la vida real de un carácter autónomo.

Para entender el comportamiento de un carácter autónomo, se realiza una división en tres capas. La Figura 13 muestra una división del comportamiento del movimiento para caracteres autónomos dentro de una jerarquía de tres capas, selección de acción, dirección y movimiento:

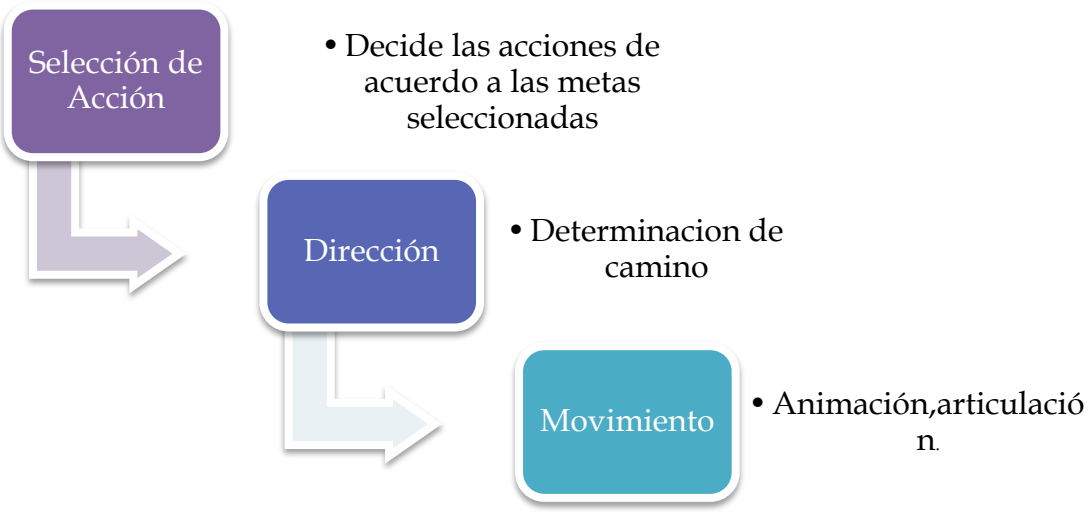

Figura 2.2 Comportamiento de movimiento

Este esquema se puede entender mediante el siguiente ejemplo: Una vaca se aparta del rebaño. El jefe pide a un vaquero recoger al animal. El vaquero rápidamente gira su caballo y lo guía hacia la vaca, posiblemente esquivando obstáculos a través del camino. En este ejemplo, el eje representa la selección de acción, el estado del mundo ha cambiado (una vaca dejo el rebaño) y estableciendo una meta (traerla de vuelta). El nivel de dirección está representado por el vaquero, quien descompone el objetivo dentro de una serie de sub-objetos (aproximarse a la vaca, evitar obstáculos, traerla de vuelta). Un sub-objetivo corresponde al comportamiento de direccionamiento para el equipo formado por el caballo y el vaquero. Usando varias señales de control (comandos vocales, espuelas, riendas) el vaquero dirige su caballo a travez del blanco. El caballo implementa el nivel de movimiento. Tomando las señales de control del vaquero como entrada, el caballo se mueve en la dirección indicada. Este movimiento es el resultado de una interacción compleja de la percepción visual del caballo, su sentido de equilibrio, y de sus músculos aplicando movimiento a su esqueleto.

2.2 Movimiento para caracteres autónomos

La capa de movimiento, es la parte inferior de los tres niveles de jerarquía descrita arriba, representa una encarnación del carácter. Este convierte sus señales de control de la capa de direccionamiento dentro del movimiento del cuerpo del carácter.

El movimiento de un carácter autónomo puede estar basado en la representación animada. Puede estar representado por una simulación de balanceo

dinámicamente basada en la manera física de caminar, o puede tener un modelo muy simple de movimiento donde agrega una representación estatice o pre-animada.

2.2.1 Un vehículo simple

Antes de introducirnos en los comportamientos direccionales, hablaremos de un modelo simple de movimiento. Este modelo de movimiento está basado en un vehículo ideal simple, y es una aproximación de una masa puntual. Una masa puntual está definida por dos propiedades: posición y masa.

El modelo simple vehicular incluye también una propiedad de velocidad. Dicha velocidad esta modificada por fuerzas aplicadas. Dado que es un vehículo, estas fuerzas son generalmente auto aplicadas y por lo tanto simplificadas.

Otra propiedad del modelo simple vehicular es la "fuerza máxima", esta limitación se debe a la interacción entre la aceleración debida al empuje y a la des-aceleración debido al rose. Como alternativa a la simulación realista de estas fuerzas limitadoras, se incluye el parámetro de "velocidad máxima". Finalmente, se tiene el parámetro de "orientación".

Estas propiedades se resumen en la siguiente tabla:

Modelo vehicular simple:

masa	escalar
vector	posición
vector	velocidad
fuerza máxima	escalar
velocidad máxima	escalar

N vectores de orientación básica

La física de este modelo está basada en una integración de Euler. A cada paso de la simulación, se determina el comportamiento de las fuerzas de manejo que son aplicadas a la masa vehicular puntual. De esta forma se produce una aceleración igual a una fuerza de manejo dividida por la masa del vehículo. La aceleración es agregada a una velocidad anterior que produce una nueva velocidad, la cual es truncada por la velocidad máxima y finalmente, la velocidad es agregada a la posición anterior.

fuerza_manejo = truncante (dirección_manejo, fuerza_maxima)

aceleracion = fuerza_manejo / masa

Velocidad = truncate(velocidad+ aceleracion, velocidad_maxima)

Posición = Posición+ velocidad.

2.3 COMPORTAMIENTOS DE CONDUCCIÓN

Los comportamientos de conducción se refieren a una clase de comportamientos de navegación de un personaje/objeto/entidad que vive en un ambiente y tiene un objetivo. Estos comportamientos permiten a un carácter autónomo moverse de una manera más real. La idea detrás de estos comportamientos se basa en una propuesta de Craig. W. Reynolds.

Estos comportamientos se pueden agrupar en dos categorías: individuales y grupales

A continuación se presentaran algunos comportamientos direccionales individuales:

- **Búsqueda**: este comportamiento actúa dirigiendo al carácter hacia una posición específica en un espacio global. Ajusta al carácter para alinear su velocidad radialmente hacia el objetivo .La "velocidad deseada" es un vector de dirección del carácter hacia el objetivo. La "velocidad deseada" puede ser velocidad_maxima. El vector de dirección es la diferencia entre su velocidad deseada y la velocidad actual del carácter.

- **La huida**: Inversa del comportamiento de búsqueda. Consiste en alejarse de una posición. Se calcula del mismo modo a excepción que primero calcula el punto más alejado restando la posición objetivo a la posición del agente.

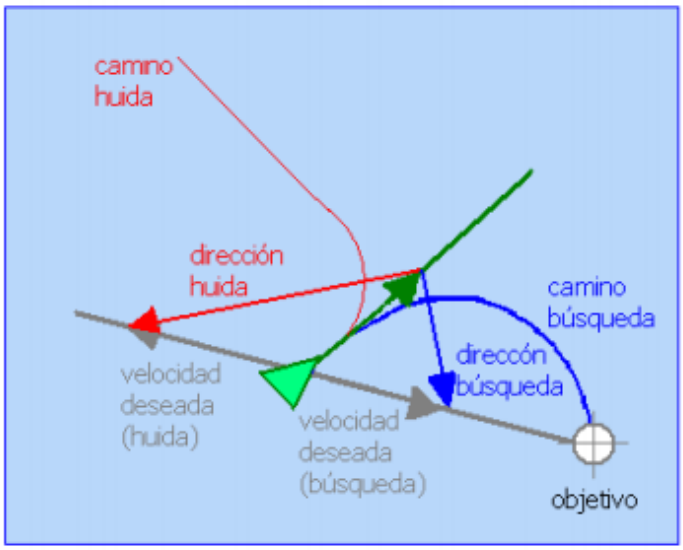

Figura 2.3 Búsqueda y Huida

- **Persecución**: Es similar al comportamiento de búsqueda excepto que el objetivo es otro carácter en movimiento. Una persecución eficaz requiere una predicción sobre la posición futura del objetivo. Para lograr esto se necesita un predictor simple y una reevaluación sobre cada paso de la simulación. Se puede usar un predictor basado en la velocidad lineal. La posición de un carácter T en una unidad de tiempo en el futuro, puede ser obtenida por escalamiento de su velocidad por T y agregando el desplazamiento a su posición actual.

- **Evasión**: Es análogo al comportamiento de persecución, a menos que la huida se use para dirigir lejos de la posición futura predicha del carácter objetivo. Existen técnicas óptimas para la persecución y evasión en el campo de la teoría de control [13].

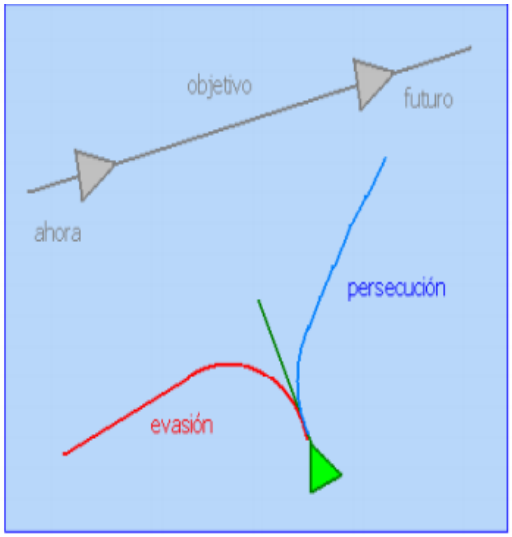

Figura 2.4 Persecución y Evasión

- **Persecución compuesta**: Se refiere a dirigir una trayectoria que pase cerca, pero no directamente dentro del movimiento del objetivo.

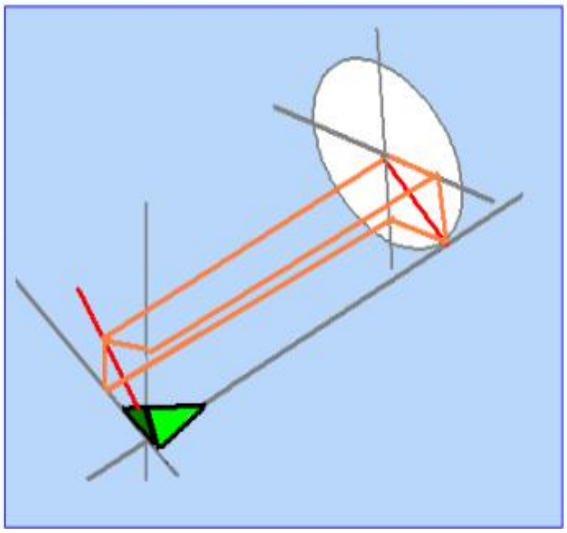

Figura 2.5 Persecución Compuesta

- **Llegada:** Es idéntico al de búsqueda, mientras el carácter está lejos de su objetivo. Pero en lugar del movimiento a través del objetivo a toda velocidad, este comportamiento hace al carácter retraerse al aproximarse al objetivo, una vez cerca, suele ser necesario desacelerar. Para ello solo es necesario ejercer una fuerza contraria a la velocidad.

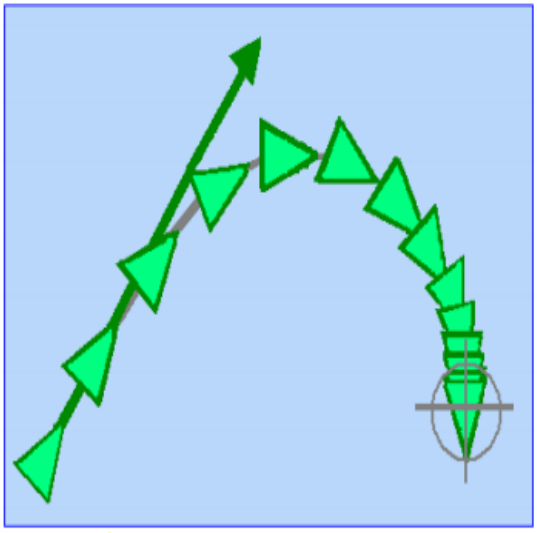

Figura 2.6 Llegada

- **Evasión de obstáculos:** permite a un carácter evitar obstáculos que son dinámicos. Los obstáculos suelen representarse como una esfera para simplificar los cálculos. De esta forma se puede comprobar si hay peligro de colisión teniendo él cuenta la velocidad del carácter y el radio de ambos.

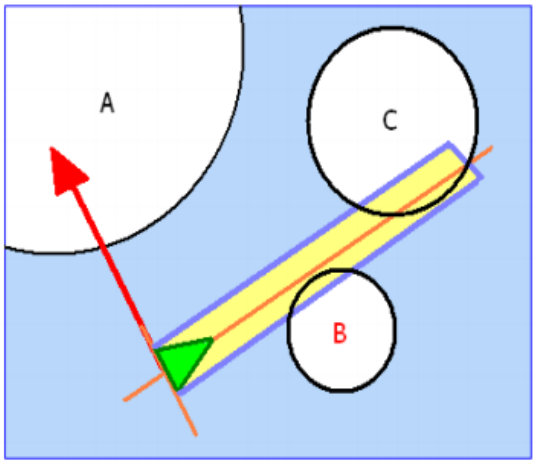

Figura 2.7 Evasión de obstáculos

- **Siguiendo una trayectoria**: permite a un carácter conducirse a través de una ruta predeterminada, como una carrera, pasillo o túnel.

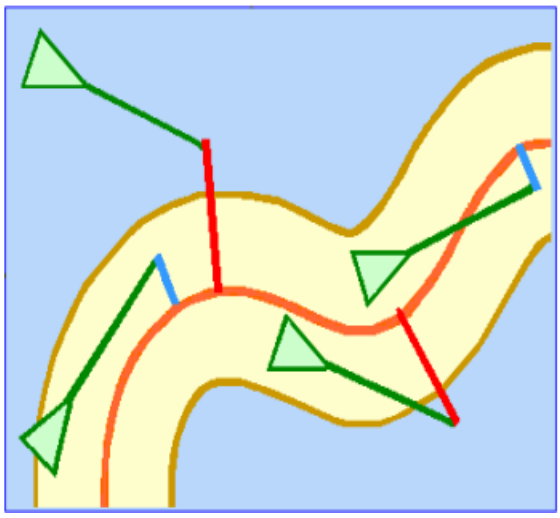

Figura 2.8 Siguiendo una trayectoria

- **Seguimiento de pared:** Aproximarse a una "pared" y mantenerse a una distancia cercana de esta.

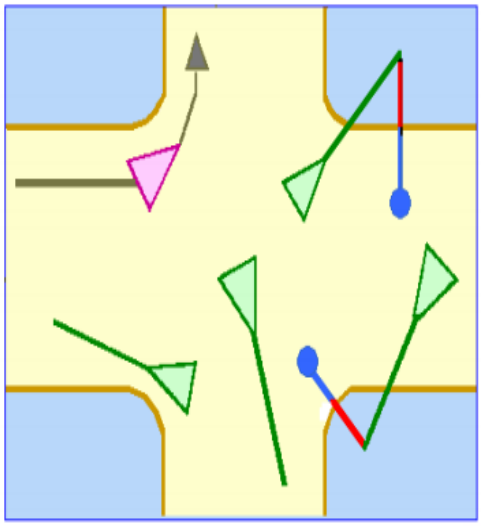

Figura 2.9 Seguimiento de pared y Contencion

- **Contención:** Movimiento en el cual se restringe permanecer dentro de cierta región.

- **Siguiendo el campo de flujo:** Proporciona una herramienta útil para dirigir el movimiento de los caracteres basados sobre su posición dentro de un ambiente.

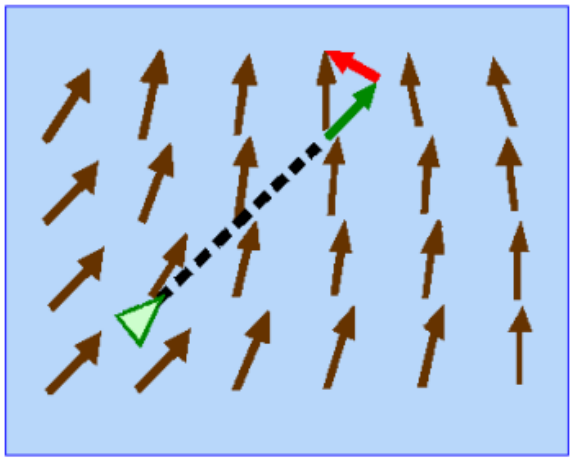

Figura 2.10 Siguiendo el campo de flujo

- **Evitamiento de colisión desalineada:** Intenta mantener caracteres que se están moviendo en direcciones arbitrarias corriendo uno del otro.

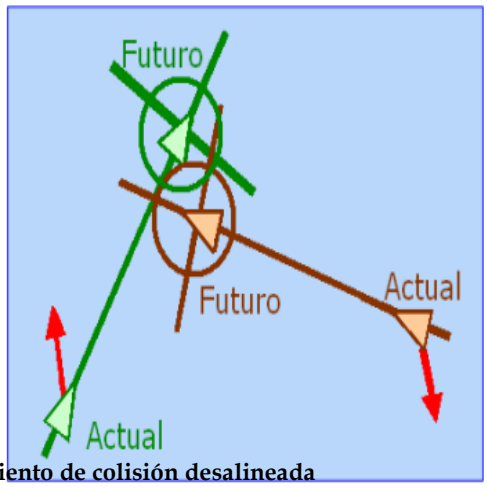

Figura 2.11 Evitamiento de colisión desalineada

Dentro de los comportamientos grupales básicos se encuentran los siguientes:

- **Separación:** Da a un carácter la habilidad para mantener una cierta distancia de separación de los otros cercanos.

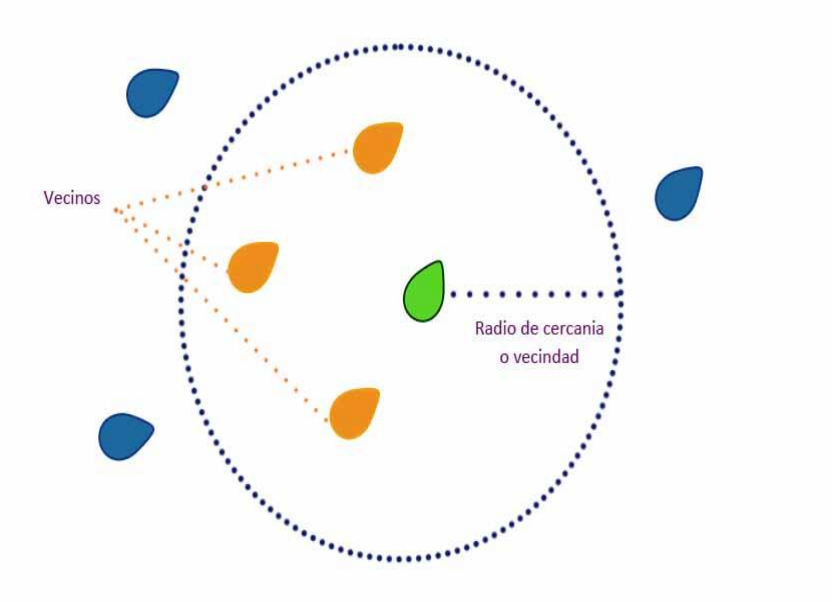

Figura 2.12 Separación

Cohesión: Permite a un carácter la habilidad para cohesionarse con otros caracteres cercanos

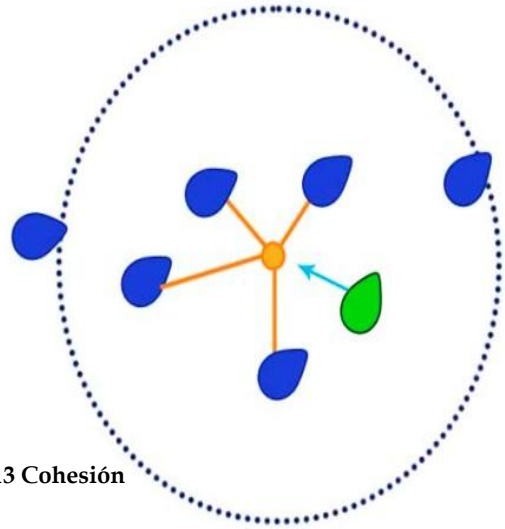

Figura 2.13 Cohesión

- **Alineación:** Permite a un carácter la habilidad para alinearse por sí mismo con otros caracteres cercanos.

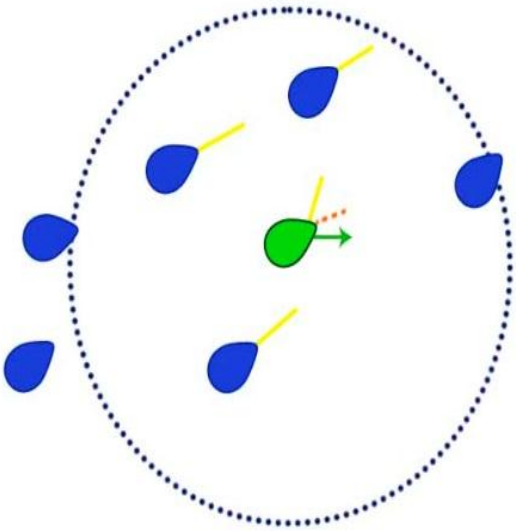

Figura 2.14 Alineación

- **Siguiendo al líder:** Causa que uno o más caracteres sigan el movimiento de otro carácter designado como el *líder*.

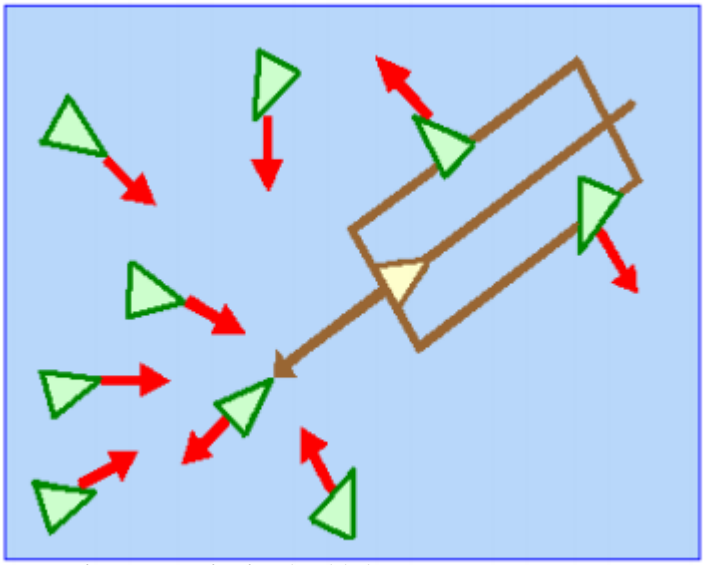

Figura 2.15 Siguiendo al líder

2.4 Combinación de comportamientos

Para obtener comportamientos más reales, es necesario seleccionar y mezclar comportamientos individuales, de esta manera se generan comportamientos más complejos e interesantes.

Existen dos maneras de combinar comportamientos, la primera forma tiene lugar en el nivel de *selección-acción*, el más alto de los tres niveles de jerarquía comportamental. Consiste en hacer cambios discretos entre los comportamientos. Por ejemplo si existe un rebaño de ovejas pastando, y de repente captan a un lobo aproximarse, se encenderá un interruptor y cambiaran de comportamiento, es decir se olvidarían de pastear y tratarían de escapar del depredador.

Por otro lado, se pueden mezclar comportamientos en paralelo. Esta mezcla de comportamientos tiene lugar en el nivel medio de la jerarquía comportamental: *dirección*. Una forma sencilla de llevar esto a cabo es calcular cada uno de los comportamientos direccionales y sumarlos, con un factor de peso para cada uno de ellos. Un ejemplo de esta combinación es cuando las ovejas corren a través del bosque, ellas mezclan evasión y evitamiento de obstáculos.

En las siguientes figuras, se pueden observar algunos delos comportamientos mencionados anteriormente.

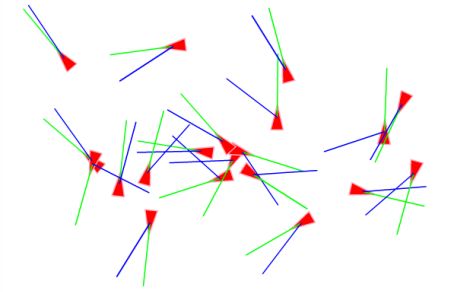

Figura 2.16 Comportamiento Vagar

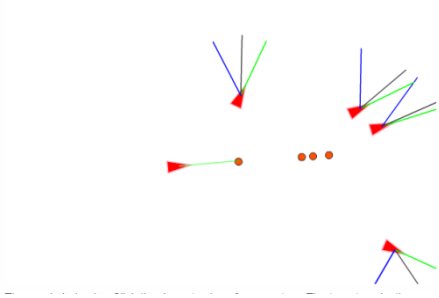

Figura 2.17 Persecución y evasión

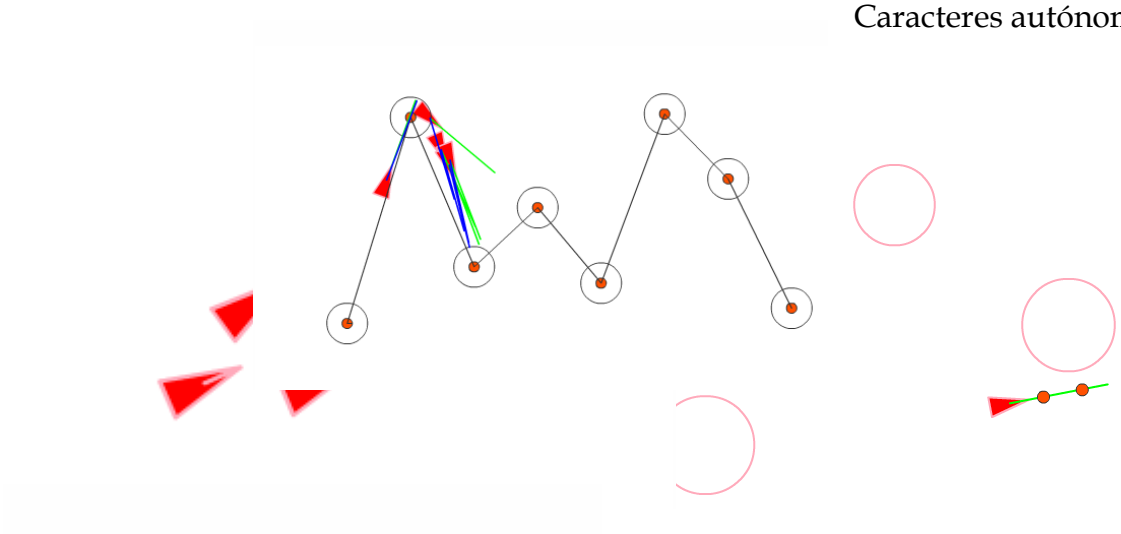

Figura 2.17 Siguiendo al lider

Figura 2.18 Evitamiento de obstáculos

Figura 2.19 Siguiendo una trayectoria

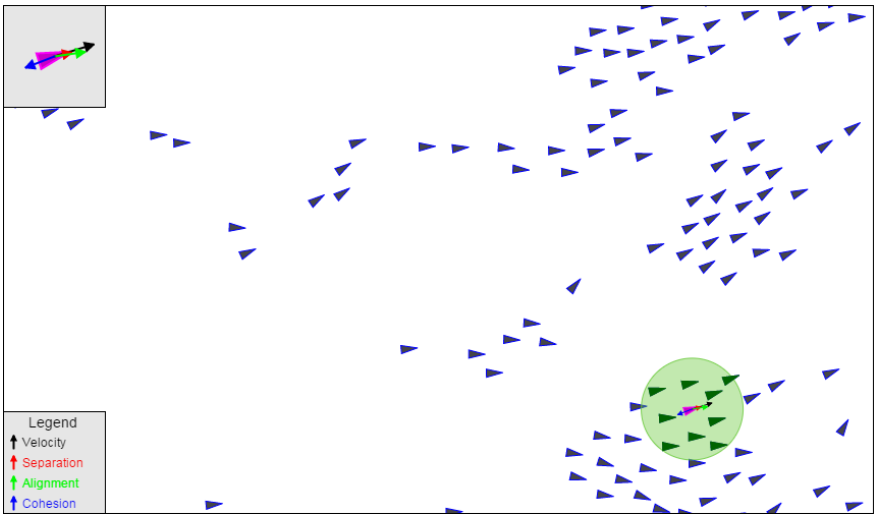

Figura 2.20 Cohesión, Alineación y Separación

Capítulo 3 PRM

3.1 INTRODUCCIÓN

En robótica durante los últimos años el problema de planificación de movimientos ha originado un gran número de trabajos de investigación. La referencia de esta área de investigación es el libro de J.C Latombe que introduce el conjunto de contribuciones de los años 80.

Durante estos años se han desarrollado diferentes enfoques de planificación de movimientos que podrían aplicarse en videojuegos. Nos concentraremos en los métodos de tipo "Roadmap probabilista", a partir de esta frase les llamaremos PRM.

Desde los años 90, los planificadores basados en roadmaps probabilísticos han demostrado un gran potencial para solucionar el problema de encontrar una ruta libre de colisiones en entornos multidimensionales (Kavraki, 1996) para robots con muchos grados de libertad.

La idea central del PRM clásico es llevar a cabo un muestreo aleatorio del espacio de configuraciones (CS_{libre}) de un robot para construir una red de caminos llamada roadmap. Esta red captura la conectividad del espacio libre por medio de un grafo o un árbol. Los planificadores de tipo PRM son simples de implementar y eficientes para múltiples aplicaciones.

A continuación se presenta un estado del arte de los PRM

3.2 ROADMAPS PROBABILISTAS

Un roadmap es un grafo donde los nodos son configuraciones libres y los arcos caminos sin colisión.

El principio de los métodos PRM es muy simple [19]. Se basa en la creación de un número determinado de nodos V de manera aleatoria en el espacio de configuraciones libre de colisiones (CS_{libre}). Con esto se pretende construir un grafo G=(V,E) que captura la conectividad del espacio de configuraciones libre. Los arcos de G, almacenados en el conjunto E, son caminos factibles uniendo los nodos de V, construidos por un método local L.

La implementación de un PRM sólo requiere dos procedimientos geométricos básicos:

- Un planificador local para calcular trayectorias admisibles.
- Un método de detección de colisiones [27][26] que detecta las interferencias y los obstáculos para una configuración dada.

La mayor parte de los métodos PRM, tiene la misma estructura general. El procedimiento es dividido en dos fases: una fase de aprendizaje y una fase de búsqueda.

3.2.1 La fase de aprendizaje.

La fase se aprendizaje se divide en una o varias etapas, en las cuales los nodos son agregados y unidos al grafo. Cada etapa utiliza una estrategia diferente para producir nuevos puntos. La fase de aprendizaje inicia habitualmente una etapa de construcción, en la cual los nuevos nodos son generados por una estrategia de muestreo. Esta etapa utiliza una estrategia de muestreo para generar N puntos en el espacio de configuraciones libre e inserta estos últimos como nodos en el grafo. Cada nuevo punto v utiliza una estrategia de elección de vecinos para seleccionar un conjunto de nodos vecinos a partir del grado. Para cada punto u del conjunto de vecinos, se llama a un método local para encontrar un camino admisible de v a u. Si el método local tiene éxito, este agrega un arco entre los nodos correspondientes en el grado. El algoritmo siguiente describe la etapa de construcción en pseudo código.

Algoritmo 1: Fase de aprendizaje.

1: $V \leftarrow \emptyset$, $N \leftarrow \emptyset$, $E \leftarrow \emptyset$

2: $n_{nodos} \leftarrow \emptyset$

3: **Mientras** que $n_{nodos} < Max_{nodos}$ **hacer**

4: q← generar una configuración libre utilizando una estrategia de muestreo

5: **Si** $q \in CSl_{ibre}$ **entonces**

6: $V \leftarrow V \cup \{q\}$

7: $n_{nodos} \leftarrow n_{nodos} + 1$

8: $V_c \leftarrow$ un conjunto de vecinos de q incluidos en V

9: **Para todo** $v \in Vc$ por orden de distancia creciente **hacer**

10: **Si** v y q no están conectados a G **entonces**

11: **Si** el método local encuentra un camino entre q y v **entonces**

12 $E \leftarrow E \cup \{(q,v)\}$

Max_{nodos} es el número máximo de nodos generados.

Para tener una mayor probabilidad de resolver una consulta, el tiempo en la fase de aprendizaje debe aumentar, de esta manera el grafo tendrá una mejor cobertura del espacio de configuraciones libre. La figura 2.1 muestra el resultado de la fase de aprendizaje es un espacio simple.

Es necesario comentar que la etapa de construcción resulta inadecuada cuando el espacio de configuraciones contiene pasajes estrechos. Un pasaje estrecho necesita una técnica de inserción de manera muy particular, es decir, los nodos deben caer en esas pequeñas regiones del espacio de configuraciones, esto podría tomar

demasiado tiempo con una estrategia simple de muestreo. Por consecuencia la etapa de construcción es seguida de una etapa denominada *etapa de expansión* [14] [15].

3.2.2 La fase de búsqueda.

En esta fase, se utiliza el grafo construido anteriormente, para resolver diferentes consultas de planificación de movimientos. La resolución de un problema se hace de la siguiente manera:

Se da una configuración de partida y un objetivo, teniendo estos dos puntos el algoritmo intenta unirlos utilizando el método local. En caso de tener éxito, solo se efectúa una búsqueda del camino en el grafo. Si el camino existe, es una secuencia de caminos locales y este corresponde a una trayectoria admisible.

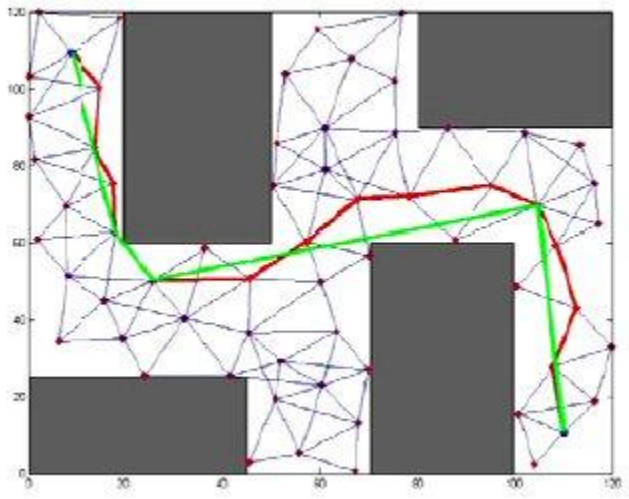

Figura 3.1

3.2.3 Alisado de caminos

Debido al carácter probabilísticos del PRM, los caminos construidos pueden ser largos e irregulares. Para mejorar esto, es necesario alizar sistemáticamente los caminos encontrados. Es se debe utilizar una técnica que reemplace los "pedazos" del camino. Una técnica clásica toma al azar dos puntos de un camino dado, y ejecuta el método local entre la configuración de partida y el primer punto, si el

camino encontrado es más corto en distancia y está libre de colisión, este se substituye, así procede para las demás partes del camino.

Esta operación se repite varias veces, durante un cierto tiempo o hasta que la ganancia del alisado sea imperceptible, de esta manera se pueden obtener caminos alisados más cotos que los caminos construidos inicialmente por PRM.

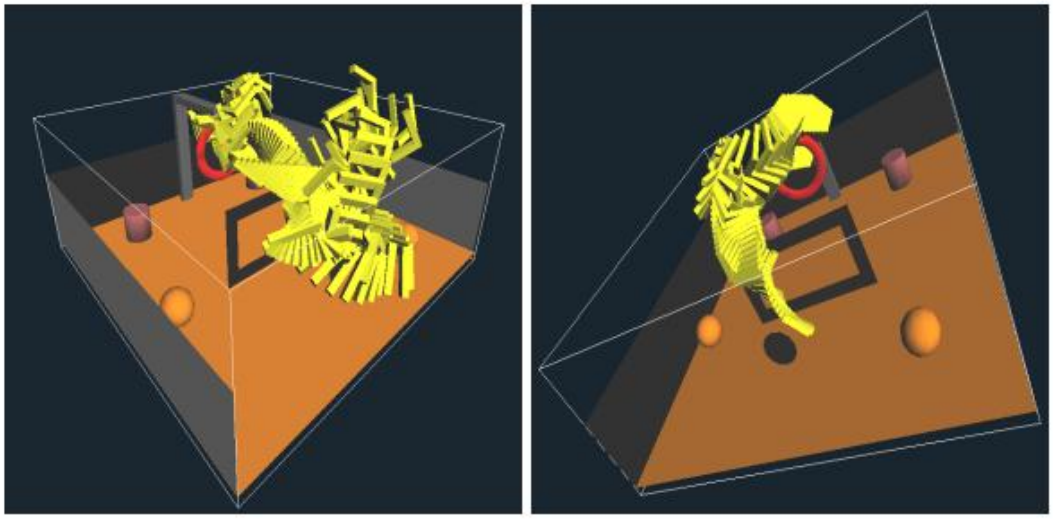

Figura 3.2 Camino encontrado y después de la fase de alisado

3.3 Estrategias de muestreo

Las estrategias de muestreo son utilizadas para generar nuevas configuraciones durante la fase de aprendizaje. Uno de los enfoques más utilizados es el muestreo aleatorio el cual consiste en realizar un muestreo del espacio de configuraciones de manera uniforme, pero existe un problema, esta estrategia funciona adecuadamente cuando no existen pasajes estrechos. Si esta técnica es utilizada cuando existen pasajes estrechos, el número total de configuraciones aleatorias seria enorme.

3.3.1 Muestreo uniforme

Como se mencionó antes esta estrategia produce configuraciones, escogiéndolas aleatoriamente. El proceso se repite hasta que se encuentre una configuración que pertenezca a $CS_{libre.}$ Este muestreo uniforme del espacio de configuraciones produce una fuerte densidad de nodos en cada componente conexa del espacio libre antes de conectarlos exitosamente, con una baja probabilidad de generar configuraciones en el interior de los pasajes estrechos.

3.4 ESTRATEGIAS DE SELECCIÓN DE VECINOS

La elección del conjunto de los futuros vecinos a los cuales el algoritmo intentará conectar una nueva configuración tendrá una fuerte influencia en la topología del grafo, en el desempeño del planificador y en la longitud del camino solución [16] [17].

3.4.1 *MÉTODO DEL BOSQUE*

Si solo nos enfocamos en la habilidad del planificador para resolver una consulta y no en la longitud de la misma, entonces no es necesario agregar arcos al grafo, ya que estos introducen ciclos, y no mejoran la habilidad del roadmap para resolver consultas. El mejor camino para evitar los ciclos es conectar una nueva configuración de los vecinos en el orden de una distancia incremental y sin considerar aquellos vecinos que ya conectan al grafo cola nueva configuración. En este sentido solo se agregan arcos cortos cuando mejoran la conectividad del roadmap. Esta es una buena selección dado que la verificación de las trayectorias es el mayor consumidor de tiempo en el algoritmo PRM y la cantidad de tiempo necesario para verificar una trayectoria es usualmente proporcional a longitud de la trayectoria. Al igual que en el método de los k-vecinos más próximos se detendrá la inclusión de vecinos a utilizar una cierta distancia *maxdist*. En adición a la *maxdist*, es común introducir una constante *maxneighbors* que especifica el número máximo de vecinos con los cuales se intentara conectar un nuevo nodo. Este método se llama método del Bosque debido a que el grafo resultante tiene la forma de una colección de árboles.

3.5 MÉTODOS LOCALES

Los métodos locales pueden ser deterministas o no deterministas. Un método determinista produce siempre el mismo resultado cuando este se aplica al mismo problema, por el contrario un método no determinista producirá resultados diferentes. El método determinista tiene ventajas que no presenta el no determinista ya que este no necesita almacenar las informaciones adecuadas sobre los caminos en el roadmap, ya que podemos utilizarla para reproducir los caminos a medida que sean necesarios.

3.5.1 *CAMINO LINEAL*

El método local más utilizado es el método lineal. Este método pertenece a la clase de métodos estrictamente deterministas. El camino calculado por este método es siempre una línea recta o una interpolación lineal entre qi y qf en el espacio de configuraciones. La Figura 2.7 a), muestra el resultado de un camino producido por este método.

3.5.2 *CAMINOS DE TIPO O A**

Los métodos locales de este tipo son ejemplos de métodos deterministas potentes [1]. Estos utilizan una simple función de potencial para guiar su búsqueda de qi a qf. Este potencial permitirá al camino de cambiar su curso y de deslizarse a lo largo de la región de los obstáculos del espacio de configuraciones, tanto tiempo como la distancia a qi no aumente. La Figura 3.3, muestra un ejemplo de camino producido por un método local de este tipo. En cada paso, el planificador calcula la distancia entre sus vecinos inmediatos en la rejilla a la configuración final, y procede con la configuración vecina libre de colisión que tiene la distancia más pequeña a la configuración final.

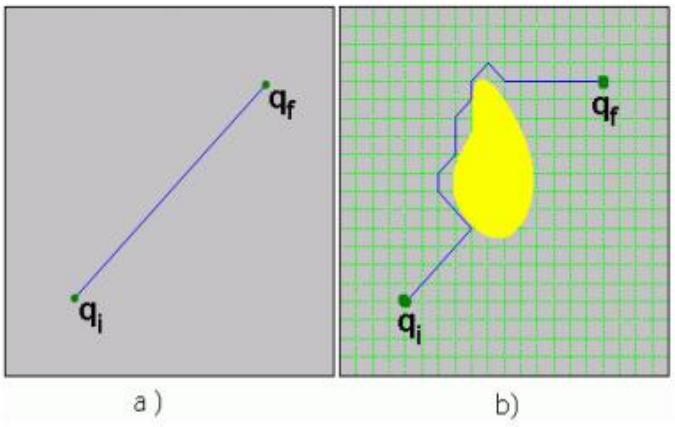

Figura 3.3 Camino producido por el método local lineal b). Resultado de un camino producido por el método local A*

Capítulo 4 MOTOR DE COMPORTAMIENTOS GRUPALES PARA EL DESARROLLO DE VIDEO JUEGOS.

Para desarrollar esta propuesta se hiso uso del engine de video juegos Unity 3D. Los engines son herramientas que hacen facilitan y agilizan el proceso de la creación de un videojuego. Este software provee toda la funcionalidad que se necesita para renderizar gráficos, detectar colisiones, simular física, administrar la comunicación con periféricos, manipular sonido, crear inteligencia artificial, manejar memoria, etc., sin que el programador tenga que comenzar desde cero cada vez que empieza un desarrollo.

Existen diferentes engines, pero entre los más sobresalientes se encuentran: RPG MAKER XP, Game Maker,Cry Engine 3,FPS Creator,Source,Unreal, y Unity 3D.

4.1 UNITY 3D

Unity 3D es un motor de videojuegos multiplataforma creado por Unity Techonologies. Unity permite crear juegos para Windows, OS X, Linux, Xbox 360, PlayStation3,Playstation,Vita, Wii, WiiU, iPad, iPhone, Android y Windows Phone.

Gracias al *plugin* web de Unity, también se pueden desarrollar videojuegos de navegador para Windows y Mac. Está programado en C++ y su última versión estable es la 5.0

Unity ofrece una gran cantidad de ventajas, se enlistaran las más importantes:

- Es multiplataforma.
- Documentación muy completa.
- Dispone de una modalidad de licencia gratuita.
- Permite la programación de scripts.
- Construcción rápida de escenas 2D y 3D.
- Importación de modelos y animaciones realzadas con otras aplicaciones 3D

La programación de scripts es un ingrediente esencial en todos los juegos. Incluso el juego más simple necesitara scripts para responder a entradas del jugador y asegurar que los eventos del juego se ejecutaran en el momento adecuado. Además los scripts pueden ser usado para crear gráficos, controlar el comportamiento físico

de objetos o incluso implementar un sistema de inteligencia artificial para los personajes del juego.

En Unity 3D se pueden usar C#, Java Script, y Boo como lenguajes de programación para crear scripts. Estos scripts son adjuntados como un componente a un GameObject, de esta forma el código será activado cuando se presione Play y empiece a correr el juego. Se pueden agregar más de un script a un GameObject.

4.2 Desarrollo

El motor de comportamientos está conformado por diferentes scripts que se comunican entre sí para llevar a cabo un comportamiento determinado. El script principal es el que se encarga de realizar el PRM, una vez que este script completa la fase de aprendizaje de aprendizaje, notifica a los comportamientos activos que pueden usar el grafo construido para realizar consultas.

En esta propuesta se implementar ron tres comportamientos:

- Autoguiado
- Búsqueda de un objetivo
- Siguiendo al líder.

Aunque estos comportamientos parezcan sencillos, se pueden combinar para crear comportamientos más complejos.

4.2.1 PRM

Como se mencionó en el capítulo uno, el movimiento es uno de los aspectos más importantes en un videojuego. Para lograr que los NPCs se muevan de un punto a otro es necesario encontrar una ruta dentro del entorno, es importante mencionar que esta ruta debe der lo más corta posible.

En robótica se han desarrollado diferentes enfoques para la planificación de movientes, algunos de estos enfoques podrían aplicarse en videojuegos, como es el caso de los PRM.

En nuestra propuesta hacemos uso de un PRM básico para llevar a cabo la planificación de movimientos.

Para desarrollar este PRM se creó un script, en el cual se realiza la fase de aprendizaje y consulta.

Para esta implementación básica, se utilizó un método uniforme para obtener el espacio libre de colisiones, y para encontrar los vecinos de cada nodo se implementó un método del bosque.

Debido a que el tiempo de respuesta en un videojuego es demasiado importante, el escenario del videojuego es dividido en una rejilla de esta manera varias posición caen dentro de un cuadro, y se obtiene un número menor de posiciones.

Mediante el uso de esta rejilla y el manejo de capas que ofrece Unity es más fácil detectar las posibles colisiones con los obstáculos estáticos.

Aparte de lo ya mencionado se hiso uso de algunas herramientas que ofrece Unity para detectar colisiones.

Una vez que la fase de aprendizaje se ha terminado, se notifica que el grafo está listo para usarse. Es entonces cuando los scripts de comportamiento comienzan a ejecutarse, estos scripts pueden hacer consultas enviando un punto inicial y final para obtener una ruta.

Cuando se realiza una consulta de búsqueda el script de PRM lleva a cabo dicha fase y una vez encontrada la ruta, realiza un alisado del camino mediante una curva de Bézier.

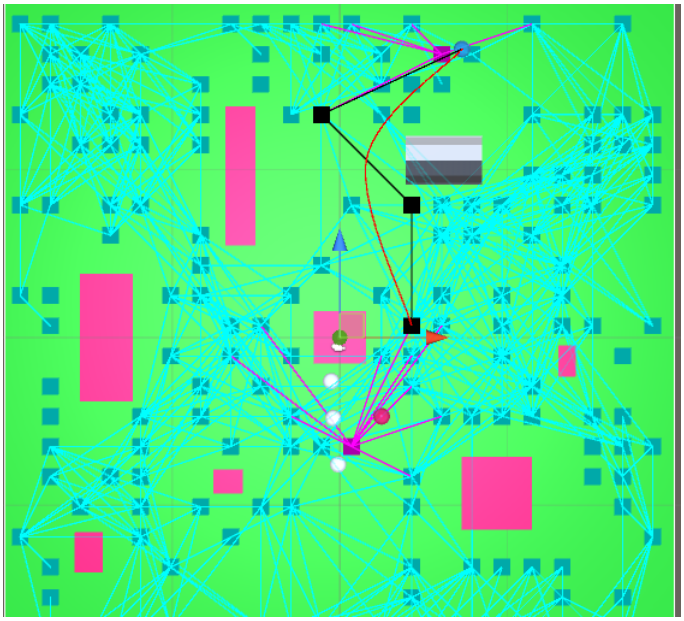

Figura 4.1 Optimización del camino .Una vez que el camino es encontrado (línea negra), se realiza la optimización (línea roja) y este es el camino que se devolverá.

4.2.2Comportamiento autoguiado

Esto comportamiento se apoya de dos sub modelos, uno que representa el comportamiento individual de los miembros del grupo y otro que influye en el comportamiento global del grupo. Para este comportamiento cada miembro del grupo debe:

- Evitar la colisión con otros miembros vecinos del grupo.
- Mantenerse cerca de sus vecinos, donde el vecindario está definido por una distancia.

Una vez que el roadmap es generado, se pueden encontrar caminos de forma rápida y eficiente, de este modo un miembro del grupo busca seguir un camino hasta un punto dado. Posteriormente la trayectoria se divide en submetas basadas en el rango de visión de un miembro del grupo. Este camino de submetas lo guarda cada miembro del equipo, y cuando una submeta está dentro de su rango de visión, la siguiente submeta es marcada como la siguiente meta. Así todos los caracteres se trasladas juntos mientras que se mueven a la siguiente meta.

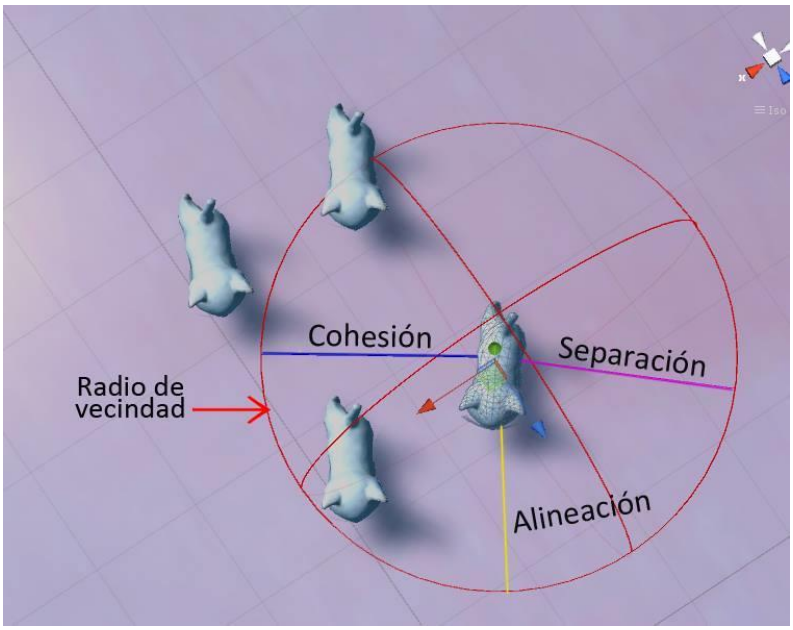

Figura 4.2 Evitar colisión con otros vecinos del grupo. Para evitar esto se puede hacer uso de los comportamientos de cohesión, separación y alineación.

Algoritmo 2 : Autoguiado

```
1:  Crear Posición inicial para cada miembro del grupo.
2:  Dividir el camino en submetas.
3   Guardar el camino dividido en cada miembro.
4:  Mientras (no se encuentre en la posición final) hacer
5:      Para (cada miembro del grupo) hacer
6:          Si (la meta está en el rango de visión) entonces
7:              Quedarse cerca de la meta
8:          Si_no
9:              Si (la submeta actual está en el rango de vision del
                miembro) entonces
10:                 La meta del miembro se fija como la submeta siguiente
11:             Si no
12:                 Sigue el camio hacia la meta actual
13:             Fin_si
14:         Fin_si
15:     Fin_para
16: Fin_Mientras.
```

4.2.3 Comportamiento búsqueda de un objetivo.

Este comportamiento es de tipo exploración, el objetivo de este comportamiento es buscar una meta y moverse hacia ella. En este caso el ambiente es conocido y se hace uso del grafo del roadmap con pesos adaptativos en los nodos. Cada miembro del grupo se comporta de forma independiente y usa e roadmap para vagar alrededor de él, siguiendo los arcos del grafo, en este comportamiento no existen las rutas predefinidas. Cuando un miembro está en un nodo con varios arcos, elige probabilísticamente un arco del roadmap dependiendo del peso de dicho arco.

Los miembros del grupo conocen el ambiente, pero no la localización de la meta. Si un carácter tiene dentro de su rango de visión la meta, comunica a los demás miembros de grupo la localización de esta.

El siguiente algoritmo resume este comportamiento.

Algoritmo 3 : Búsqueda de un objetivo

```
1:  Crear Posición inicial para cada miembro
2:  Mientras (algún carácter no se encuentre en la posición final) hacer
3:      Para (cada miembro del grupo) hacer
4:          Si (la meta está en el rango de visión) entonces
5:              Incrementar los pesos de los arcos de la ruta a la meta
6:          Si no
7:              Si (encontró algún nodo sin conexión) entonces
8:                  Quitar nodos del camino encontrado hasta que una Nueva liga se
                    encuentre
9:                  Decremento los pesos de los nodos que se eliminaron
10:             Si no
11:                 Seleccionar el nodo vecino del nodo actual de menor peso
12:                 Colocar este nodo en el camino
13:             Fin si
14:         Fin si
15:     Fin para
16: Fin mientras
17:  Comunicar a los demás caracteres la ruta encontrada
18:  Para (cada miembro del grupo a excepción del que encontró la meta) hacer
19:     Seguir el camino encontrado
20: Fin para
```

4.2.4 Comportamiento siguiendo al líder.

Este comportamiento consiste en designar a un miembro del grupo como líder, al cual los miembros restantes deberán seguir. Para lograr este comportamiento se utilizan ideas de los comportamientos explicados anteriormente.

Del comportamiento de autoguiado se toma la idea de dividir el camino encontrado por el planificador, y este camino es el que debe seguir el líder para ir de un punto inicial a un punto final.

Para los demás miembros del grupo se utiliza el comportamiento de búsqueda de un objetico, pero en lugar de utilizar la posición del nodo que deben seguir, se coloca la posición actual del líder. Estas ideas se describen en el siguiente algoritmo.

Algoritmo 4: Siguiendo al líder.

```
1:  Crear Posición inicial para cada miembro
2:  Elegir líder
3:  Guardar la trayectoria general en el líder
4:  Mientras (líder no encuentre la meta) hacer
5:      Si (la meta está en el rango de visión del líder) entonces
6:          Quedarse cerca de la meta
7:      Si no
8:          Si (la submeta actual está en el rango de visión del líder)entonces
9:              La meta del líder se fija como la submeta siguiente
10:         Si no
11:             Caminar hacia la meta actual
12:             Para (cada uno de los restantes miembros del grupo) hacer
13:                 Colocar como submeta la posición del líder
14:                 Caminar hacia la submeta actual
15:             Fin para
16:         Fin si
17:     Fin si
18:  Fin mientras
```

Capítulo 5 RESULTADOS

Antes de presentar algunos resultados experimentales se explicara cómo funciona este motor.

Se utilizó la versión de Unity 5.02 para desarrollar todo el proyecto, los script se desarrollaron bajo el lenguaje C#.

En Unity podemos usar GameObjects para insertar nuestros scripts, en estos GameObjects podemos incluir más de un solo script, de esta manera se pueden configurar las variables públicas de dichos script se es necesario.

El GameObject puede convertirse en un prefab, que forma parte de un paquete. En nuestro se creó un GameObject llamado RutaPRM, el cual contiene al script de PRM y Grid.

Los comportamientos y el prefab de RutaPRM se exportaron como un paquete que puede ser importado en otros proyectos de Unity.

En la siguiente figura se muestran los cambios que pueden realizarse en sl script de PRM

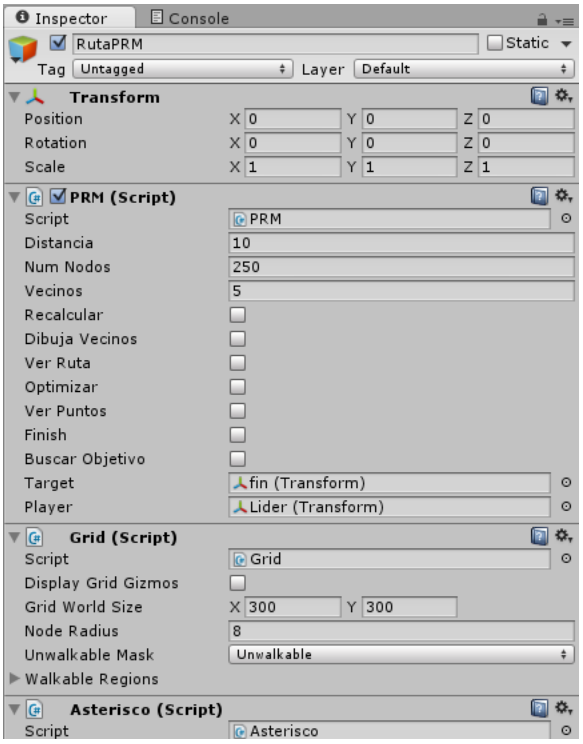

Figura 5.1 RutaPRM

Como se puede ver en el script de PRM se pude cambiar la distancia entre cada nodo para determinas si son vecinos. El número de nodos que se calcularan y el total de vecinos para cada nodo.

Las siguientes variables que se pueden modificar son de booleanas, es por ello que solo tiene un recuadro para seleccionarlas. Dibujar vecinos y ver puntos sirven para visualizar los puntos que conforman al PRM y sus respectivos vecinos. Con ver ruta y optimizar se puede ver la ruta calculada y su optimización.

La variable Buscar Objetivo debe ser seleccionada cuando se quiera utilizar el comportamiento de "Búsqueda de un objetivo"

Las variables Target y Player hacen referencia a los puntos de inicio y final que se utilizaran en la consulta, si estas variables no están asignadas se da por hecho que deben buscarse puntos aleatorios de inicio y fin.

Para que el script de PRM pueda funcionar debe estar configurado el Script de Grid, en este script se debe modificar el Grid World Size en "x" y "y" estas medidas hacen referencia al tamaño de la escena. El tamaño del Node Radius y la máscara en la que están situados los obstáculos. Cuando se marca la casilla de "Display Grid Gizmos" se puede observar la división del escenario.

Es obligatorio que este prefab este insertado en las escenas en las que quieran utilizarse comportamientos, de lo contrarios no se podrá hacer uso de ellos.

En las siguientes figuras se muestra los resultados de marcar algunas de las variables booleanas que se comentaron antes, en un escenario de prueba.

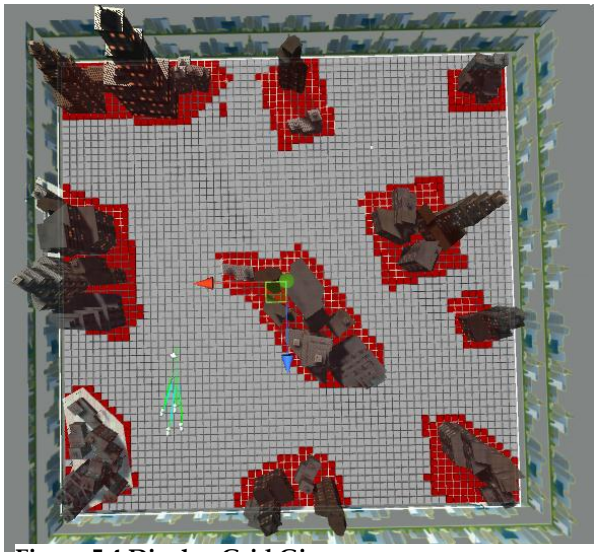

Figura 5.4 Display Grid Gizmos.

Los cuadros rojos representan que existe un obstáculo y que no es posible pasar por esa parte.

Figura 5.3 "Ver Puntos".

Los puntos azules son los nodos creados por el algoritmode PRM

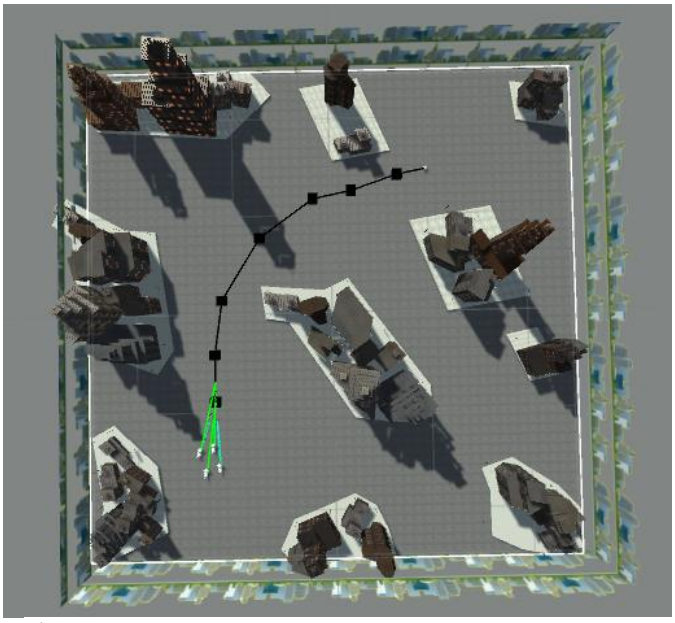

Figura 5.2 "Ver ruta".

Se muestra la ruta encontrada entre el punto inicial y final

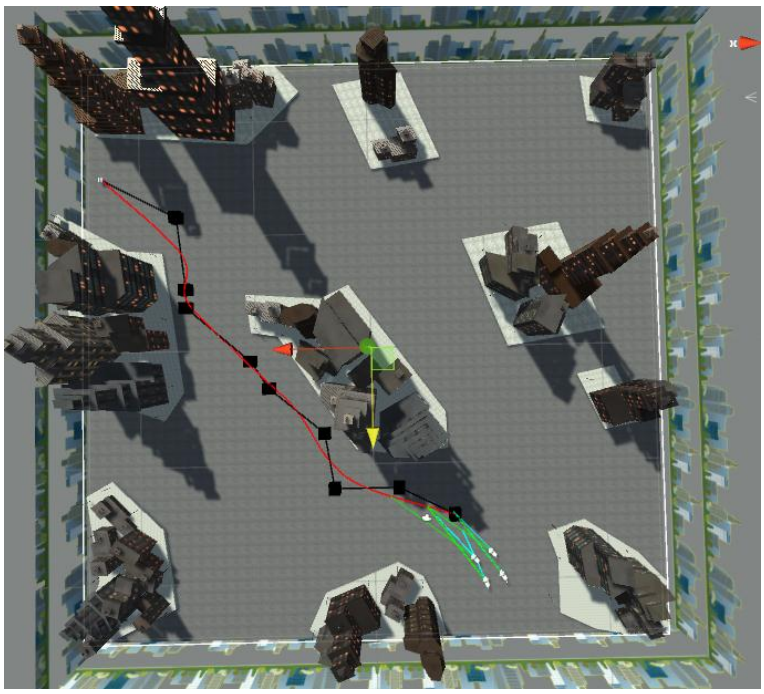

Figura 5.5 "Optimizar".
La línea negra representa el camino encontrado entre el punto inicial y el final, mientras que la línea roja representa la optimización de dicho camino.

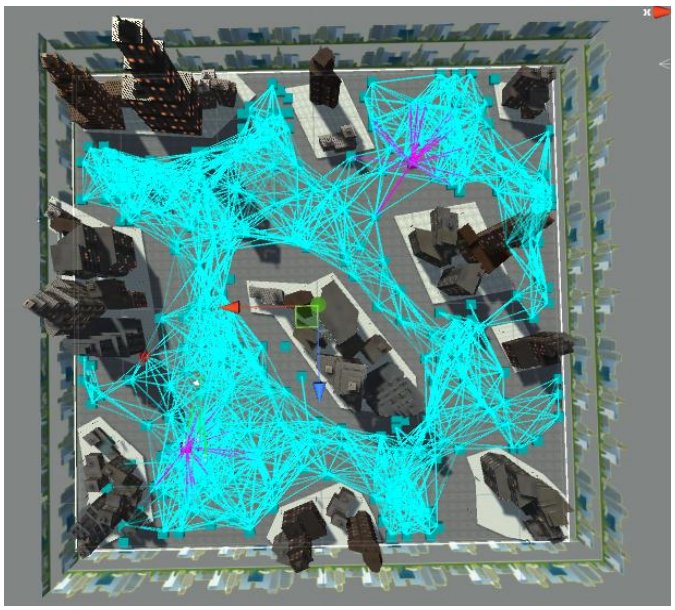

Figura 5.6 "Vecinos".

Las líneas azules representan la conexión entre cada nodo y sus vecinos. Las líneas rosas representan los vecinos de los puntos que fueron marcados como inicio y final.

Todas las pruebas fueron realizadas en una maquina con las siguientes características.

- procesador Intel(R) Core(TM) i 3 CPU M 380 @ 2.53 GHz 2.53 GHz.
- 4 GB de Ram
- SO Windows 7 64 bits

Se crearon tres escenarios con tamaño y obstáculos diferentes en Unity 3D. Se descargaron dos personajes de la "Asset Store", quienes ya tienen incluida la animación del caminado, para poder llevar a cabo las diferentes pruebas.

El primer escenario, figura 5.7, solo contiene una casa y algunas jardineras como obstáculo. El primer comportamiento que se probó fue el de auto guiado, el grupo de gatos que se encontraba en la esquina inferior izquierda tenían que llegar a un punto cerca de la casa.

Una vez probado el comportamiento de autoguiado, se asignó a uno de los gatos como líder, para llevar a cabo el comportamiento de siguiendo al líder.

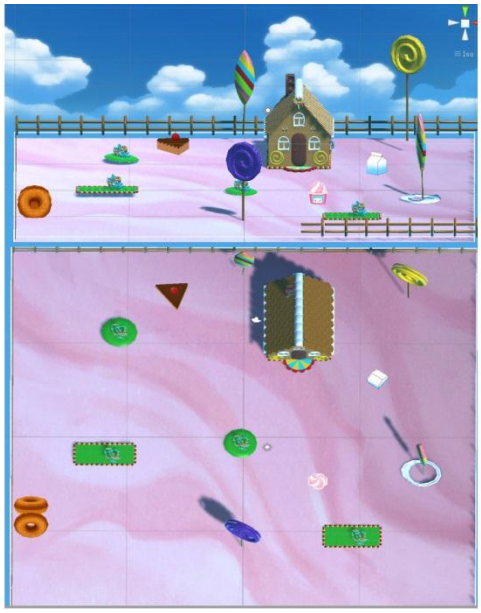

Figura 5.1 Escenario 1

Figura 1.2 Escenario 1. Siguiendo al lider

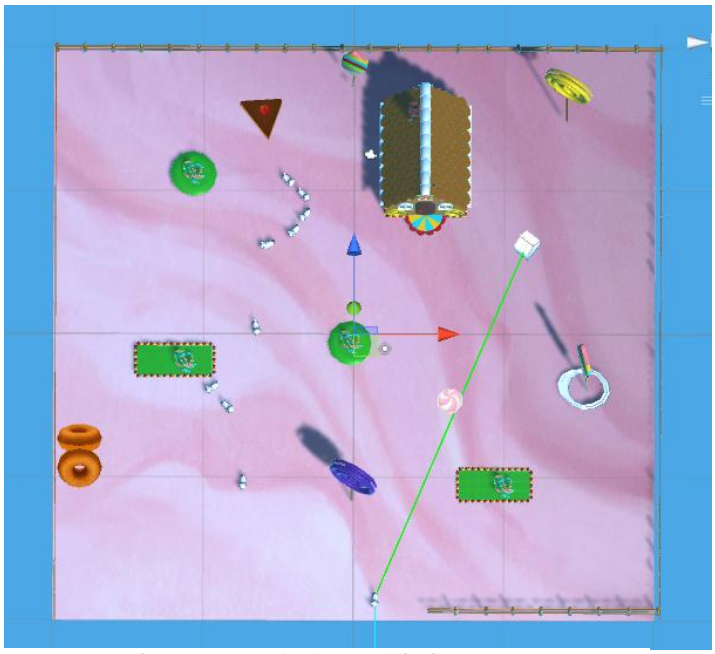

Figura 1.3 Escenario 1. Busqueda de un objetivo

El ultimo comportamiento a probar fue el de “Busqueda de un objetivo”

Para el segundo escenario, figura 5.10, se aumentó el tamaño del plano y se agregaron más obstáculos. Para este caso se construyó una pequeña ciudad, se siguió utilizando gatos como personaje pero se agregó un personaje de mayor tamaño que actuaría como su líder. Esto con la finalidad de mostrar que no importa el tamaño de los personajes los comportamientos siguen funcionando de manera correcta.

Figura 1.4 Escenario 2

En el escenario 3, Figura 5.11, además de hacer más grande el plano, se cambió el personaje, y se aumentó el número de ellos.

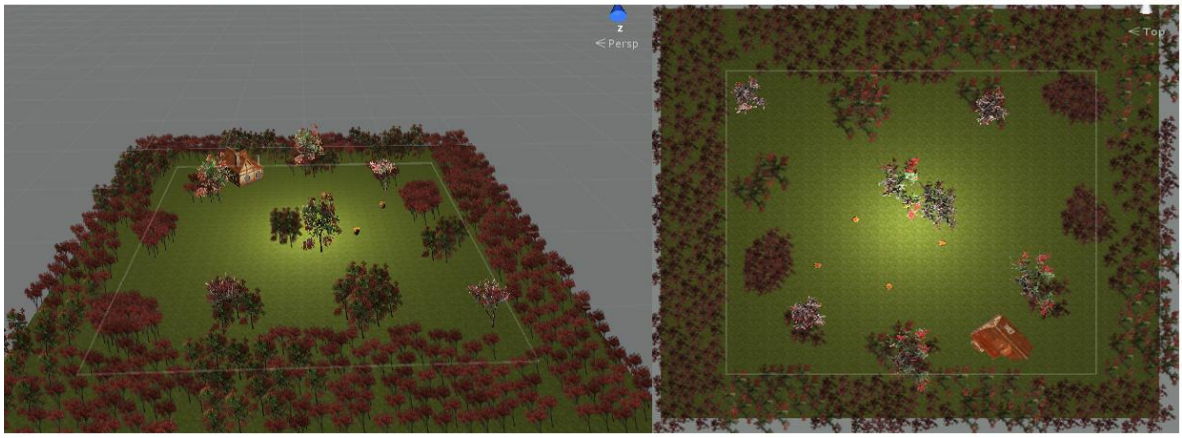

Figura 1.5 Escenario 3

Para este escenario los personajes fueron mariposas, se utilizaron 12 mariposas y se asignó a uno de ellas como líder. La prueba para este escenario cambio un poco, los comportamientos se siguieron utilizando, pero esta vez al llegar al punto final, se calculaba un nuevo punto al cual debían dirigirse, para esto se calculaba una nueva ruta, este proceso se repite durante todo el tiempo de ejecución. En las figuras 5.12 y 5.13, se muestra el cambio de puntos al que deben llegar.

Figura 1.6 Escenario 3 Primera Configuración

Figura 1.7 Escenario 3. Segunda configuración

Para cada escenario se probaron cuatro configuraciones, la primera configuración fue con 200 nodos y 10 vecinos, la segunda 200 nodos y 20 vecinos, la tercera 300 nodos y 10 vecinos y la ultima 300 nodos y 20 vecinos. La parte en la que se notan las diferencias de estas configuraciones es en la parte del aprendizaje, en las siguientes tablas se muestra una media de los resultados obtenidos.

	Fase de aprendizaje	Tamaño del escenario	Núm. Nodos	Núm. Vecinos
Escenario 1	6219 ms	40x40 (4x4)	200	10
Escenario 2	7548 ms	50x50 (5x5)	200	10
Escenario 3	21406 ms	400x400 (40x40)	200	10

	Fase de aprendizaje	Tamaño del escenario	Núm. Nodos	Núm. Vecinos
Escenario 1	11069 ms	40x40 (4x4)	200	20
Escenario 2	12567ms	50x50 (5x5)	200	20
Escenario 3	26774 ms	400x400 (40x40)	200	20

	Fase de aprendizaje	Tamaño del escenario	Núm. Nodos	Núm. Vecinos
Escenario 1	9611 ms	40x40 (4x4)	300	10
Escenario 2	10778 ms	50x50 (5x5)	300	10
Escenario 3	23620 ms	400x400 (40x40)	300	10

	Fase de aprendizaje	Tamaño del escenario	Núm. Nodos	Núm. Vecinos
Escenario 1	18756 ms	40x40 (4x4)	300	20
Escenario 2	20559 ms	50x50 (5x5)	300	20
Escenario 3	42006 ms	400x400 (40x40)	300	20

En los resultados se puede observar que los tiempos no cambian mucho en relación al tamaño del terreno. El tiempo aumenta dependiendo del número de nodos generados y cuantos vecinos habrá para cada uno de ellos.

Capítulo 6 CONCLUSIONES Y TRABAJOS FUTUROS.

El objetivo principal de la elaboración de esta tesis fue el proponer un motor se comportamientos grupales, que pudiera ser utilizado en proyectos de desarrollo de videojuegos bajo Unity 3D, utilizando técnicas PRM. En este documento hemos mostrado que se pude lograr comportamientos de grupos usando la información de un roadmap.

La simulación de comportamientos grupales puede ser utilizada para dar más realismo a un videojuego. Ya sea para complementar un escenario, en el que por ejemplo se necesita un grupo de aves volando, o un grupo de peces en un estanque, o que sean utilizados para mover enemigos.

Teniendo comportamientos simples es posible tener comportamientos más complejos, de esta manera podemos tener cualquier tipo de comportamiento deseado.

Los comportamientos presentados en este documento pueden ser aplicados fácilmente en proyectos realizados con Unity 3D. Solo basta con importar el paquete creado y aplicarlo en los personajes que se requiera. También se pueden modificar ciertos parámetros, para lograr los resultados deseados. Si llega a ser necesario se pueden crear nuevos scripts basados en los ya existentes, para lograr comportamientos más complejos.

Aún nos queda un gran campo por explorar dentro de la IA en videojuegos, para complementar esta primera propuesta es necesario la implementación de más comportamientos, de esta forma se podrán combinar y obtener comportamientos más complejos.

De acuerdo al modelo de IA en videojuegos, descrito en el capítulo uno, este trabajo abarca una parte de la capa de movimiento. Una forma de mejorar este trabajo seria incursionando en la capa de estrategia y toma de decisiones.

Bibliografía

[1] J. McCarthy, «What is Artificial Intelligence ?,» 12 11 2007. [En línea]. Available: http://www-formal.stanford.edu/jmc/whatisai/. [Último acceso: 10 12 2015].

[2] AIIDE, «Artificial Intelligence and Interactive Digital Entretainment Conference».

[3] IEEE, «Simposium on Computacional Intelligence and Games».

[4] S. Rabin, AI Game Programming Wisdom.

[5] H. J. v. d. H. Schaeffer, Games, computers, and artificial intelligence, 2012.

[6] J. G. Raessens, Handbook of computer game studies, 2005.

[7] G. Group, *Universidad de Alberta.*

[8] G. a. A. Group., *Universidad Masstritch, Holanda.*

[9] C. E. Shannon, «Programming a Computer for Playing Chees,» *Philosophical Magazine,* nº 314, 1949.

[10] A. L. Samuel, «Some Studies in Machine Learning Using the Game of Checkers,» *IBM Journal,* pp. 211-229, 1959.

[11] AiGamesDev, «Top 10 Most Influential AI Games,» 9 2007. [En línea]. Available: http://aigamedev.com/open/review/top-ai-games. [Último acceso: 12 2016].

[12] R. C.W, «Steering Behaviors For Auntonomus Characters,» San Jose,California, 1999.

[13] I. Rufus, Differential Games : A Mathematical Theory with Application to Warfare and Pursuit, New York, 1965.

[14] K. L., «Random networks in configuration space fot fast paht planning. PhD thesis,» Stanford University, 1995.

[15] K. L. L. J.C, «Ramdomized preprocessing of configuration space for fast paht planning,» *Robotics and Automation,* pp. 3020-3025, 1996.

[16] S. P. ,. L. J. C. ,. a. O. M. Kavraki L., «Probabilistic roadmaps for fast path planning in high dimensional configuration,» *IEEE Transactions on Robotics and Automation,* pp. 256-580, 1996.

[17] O. M. y. S. p, «A probabilistic learning approach to motion planning,» *Workshop on Algorithmic Foundations of Robotics,* pp. 19-37, 1994.

[18] L. J. G.-M. Murillo Javier, «Implementacion de juegos de estrategia con programacion evolutiva,» Universidad Complutense de Madrid, 2005.

[1] J. McCarthy, «What is Artificial Intelligence ?,» 12 11 2007. [En línea]. Available: http://www-formal.stanford.edu/jmc/whatisai/. [Último acceso: 10 12 2015].

[2] AIIDE, «Artificial Intelligence and Interactive Digital Entretainment Conference».

[3] IEEE, «Simposium on Computacional Intelligence and Games».

[4] S. Rabin, AI Game Programming Wisdom.

[5] H. J. v. d. H. Schaeffer, Games, computers, and artificial intelligence, 2012.

[6] J. G. Raessens, Handbook of computer game studies, 2005.

[7] G. Group, *Universidad de Alberta.*

[8] G. a. A. Group., *Universidad Masstritch, Holanda.*

[9] C. E. Shannon, «Programming a Computer for Playing Chees,» *Philosophical Magazine,* nº 314, 1949.

[10] A. L. Samuel, «Some Studies in Machine Learning Using the Game of Checkers,» *IBM Journal,* pp. 211-229, 1959.

[11] AiGamesDev, «Top 10 Most Influential AI Games,» 9 2007. [En línea]. Available: http://aigamedev.com/open/review/top-ai-games. [Último acceso: 12 2016].

[12] R. C.W, «Steering Behaviors For Auntonomus Characters,» San Jose,California, 1999.

[13] I. Rufus, Differential Games : A Mathematical Theory with Application to Warfare and Pursuit, New York, 1965.

[14] K. L., «Random networks in configuration space fot fast paht planning. PhD thesis,» Stanford University, 1995.

[15] K. L. L. J.C, «Ramdomized preprocessing of configuration space for fast paht planning,» *Robotics and Automation,* pp. 3020-3025, 1996.

[16] S. P. ,. L. J. C. ,. a. O. M. Kavraki L., «Probabilistic roadmaps for fast path planning in high dimensional configuration,» *IEEE Transactions on Robotics and Automation,* pp. 256-580, 1996.

[17] O. M. y. S. p, «A probabilistic learning approach to motion planning,» *Workshop on Algorithmic Foundations of Robotics,* pp. 19-37, 1994.

[18] L. J. G.-M. Murillo Javier, «Implementacion de juegos de estrategia con programacion evolutiva,» Universidad Complutense de Madrid, 2005.