UNIVERSIDAD DE MATANZAS

FACULTAD DE INFORMÁTICA

Tema: Extensión de Unity para composición de Inteligencia Artificial Grupal para Videojuegos.

Autor: Dainel García García

Tutor: José Enrique Díaz Ramos

Matanzas, Cuba

El presente trabajo está dirigido al desarrollo de una extensión que se incorpore a la plataforma Unity para crear personajes enemigos y a su vez conformar grupos con estos, dotándolos de una Inteligencia Artificial (IA) para crear un comportamiento grupal en el terreno contra el jugador. Para la realización de este trabajo se hizo un estudio sobre los principales estilos de videojuegos en específico los del género de aventura y acción. Además se analizaron los comportamientos grupales de los personajes enemigos en varios videojuegos de mecánicas en tiempo real para así conformar la propuesta de solución.

Para el desarrollo se empleó la plataforma de desarrollo Unity y para guiar el proceso de utilizó la metodología ágil de desarrollo de software Programación Extrema (XP). Como resultado de este trabajo se obtuvo una extensión que permitirá crear personajes y grupos de enemigos para videojuegos de aventura y acción que utilicen el combate en tiempo real, sobre una base de comportamiento inteligente.

Palabras clave: comportamientos grupales, IA, personajes, extensión.

This paper is intended to show the develop cycle of an extension that includes the platform Unity to create enemies and also make groups with them, give then an Artificial Intelligence (AI) to create group behavior against the player in the field. To make this paper was necessary made an investigation about the most important videogame styles, specially adventure and action. Also analyzed in videogames the behavior of enemies characters, to this way make a proposition for a solution.

For the development was used platform Unity and to lead the process the agile development methodology of software Extreme Programming (XP). As a result of this paper was obtained an extension that allows to create characters and groups of enemies for adventure and action videogames that uses real time battle, on a base of intelligent behavior.

Key words: group behavior, IA, characters, extension.

Índice

Introducción

Con la tecnología, la vida de las personas ha tenido un cambio radical, donde muchos de los procesos de la vida cotidiana se vieron afectados; mientras que algunos se perfeccionaban y se hacían más fáciles con el uso de las nuevas herramientas tecnológicas, otros desaparecían, dándole paso a los procesos automatizados por las máquinas creadas por el hombre.

La informática es una ciencia en continua evolución que con el transcurso de los años se ha aplicado a casi todos los sectores sociales. El aumento del conocimiento acumulado sobre esta y el creciente desarrollo tecnológico de las Tecnologías de la Informática y las Comunicaciones (TIC), le ha dado un nuevo giro al mundo, proporcionando nuevas oportunidades al desarrollo científico.

El creciente avance de las TIC y desarrollo en la creación de equipos informáticos y el estudio e investigación de las técnicas de gráficos generados por computadoras, dieron surgimiento a la realidad virtual y a la Inteligencia Artificial (IA). De esta manera se dio un giro a la forma de entretenimiento de las personas, permitiéndoles conocer lugares a los que nunca habían ido antes e interactuar con otras personas, surgiendo así disímiles géneros de videojuegos.

Estos videojuegos cada vez se hacen más complejos, en cuanto a gráficos, alcance, dificultad, etc., dándole una mayor experiencia de juego al usuario, ganando a la vez más seguidores. Esta dificultad se complejiza a medida que se hacen mejoras, perfeccionándose la IA de los personajes que aparecen en estos.

Es apropiado pensar en la IA como el comportamiento inteligente exhibido por el equipo que ha sido creado, o tal vez los cerebros artificiales detrás de ese comportamiento inteligente. El estudio de la IA no es necesariamente para el propósito de crear máquinas inteligentes, sino con el propósito de obtener un mejor conocimiento de la naturaleza de la inteligencia humana.

La prueba de fuego para la IA es lo cerca que está a la inteligencia humana. Una IA capaz de resolver un problema que requiere de inteligencia, si llegara a resolverse por un ser humano, no es suficiente, también debe aprender y adaptarse para ser considerada inteligente. La IA que cumple estos requisitos se considera IA Fuerte. A

diferencia de esta, existe la IA Débil que incluye una gama más amplia de propósitos y tecnologías, incluyendo a los videojuegos en esta categoría.

La conclusión es que la definición de IA es bastante amplia y flexible. Todo lo que da la ilusión de inteligencia a un nivel adecuado hace los videojuegos mas inmersivos, desafiantes y, lo más importante, más divertidos.

La inteligencia artificial en un videojuego, se refiere a las técnicas utilizadas en computadoras y videojuegos para producir la ilusión de inteligencia en el comportamiento de los personajes no jugadores (NPC). Es un agente electrónico que puede pensar, evaluar y actuar en ciertos principios de la optimización y la coherencia para cumplir con una meta o propósito.

Actualmente la IA es uno de los puntos más importantes a la hora de estudiar y criticar un videojuego. Estos sistemas se han ido perfeccionando hasta llegar a crear grupos inteligentes de enemigos que se ayudan entre sí para eliminar el personaje principal. Se han realizado avances en los últimos años de algunos sistemas de IA como es el caso de Némesis pero, según estudios de expertos en el tema, la inteligencia de aliados y enemigos virtuales se ha mantenido sobre la misma línea sin tener mejoras significativas en el caso de los agentes inteligentes que no utilizan redes neuronales. Desde hace algunos años, las compañías han comenzado a crear IA grupales en los enemigos para mejorar la experiencia de usuario en los combates en tiempo real.

Durante el desarrollo de videojuegos, dentro de la gran cantidad de roles que intervienen en este proceso existen trabajadores que se dedican a crear las IA y otros a componerlas en la plataforma de desarrollo para crear la interacción entre los personajes, en su mayoría utilizan métodos de composición manual para crear los grupos de agentes inteligentes que se van a utilizar en el videojuego.

Cuba en este último año ha estado inmersa en el desarrollo de videojuegos de carácter lúdico. Como pionera en este desarrollo la Universidad de las Ciencias Informáticas (UCI) ha desarrollado un grupo de videojuegos en colaboración con los estudios de animación del ICAIC lo cual ha sido los primeros pasos de una industria lúdica naciente. Actualmente en los centros donde se lleva a cabo el desarrollo de videojuegos en Cuba también se realiza la composición de los agentes inteligentes de manera manual, haciendo que se prolonguen los tiempos de desarrollo en este tipo de tareas y en muchas ocasiones atrasando otros procesos por cuenta de retrasos en la composición.

Teniendo en cuenta la situación antes descrita se plantea como **problema de la investigación**: ¿Cómo agilizar el proceso de creación de comportamientos grupales enemigos para videojuegos de acción y aventura para combates en tiempo real?

Se define como **objeto de estudio** la IA en los videojuegos y como **campo de acción**: La IA grupal en enemigos para combates en tiempo real.

Para dar solución al problema científico de la presente investigación se plantea el siguiente **objetivo**: Desarrollar una extensión para la plataforma Unity que permita crear grupos de enemigos determinando sus características y tácticas en el grupo.

Posibles resultados:

- Extensión con interfaz en Unity que permita crear IA grupales de enemigos para combates en tiempo, real.
- Escenario Demo con los funcionamientos de la extensión.
- Manual de funcionamiento.

Para dar solución a estos resultados, se proponen las siguientes tareas de investigación:

- Elaborar marco teórico de la investigación.
- Estudio de las características de los sistemas de IA grupales para combates en tiempo real.
- Diseño e Implementación de un escenario demo para la validación.
- Diseño e implementación de la extensión.
- Comprobación del funcionamiento de la extensión propuesta mediante las pruebas de aceptación.

Para dar cumplimiento a estas tareas de investigación se emplearon **métodos científicos teóricos y empíricos**.

Método Teórico

- **Histórico-Lógico:** Este método teórico se utilizará para realizar el estudio del estado del arte. Además, para conocer los tipos de comportamiento grupales de los personajes enemigos en los videojuegos.
- **Analítico-Sintético:** Se utilizará para estudiar cómo se puede vincular el comportamiento inteligente con los personajes creados por la extensión.
- **Modelación:** Este método se utilizará para crear un prototipo funcional de la extensión.

Métodos Empíricos:

- **Análisis documental:** Para seleccionar la información necesaria para la construcción del marco teórico.
- **Pruebas:** Para comprobar el desempeño de la extensión elaborada.

El presente documento está estructurado en 3 capítulos. En el Capítulo 1 se presentan los elementos teóricos que sirven de base a la investigación del problema planteado. En el Capítulo 2 se propone un método de solución basado en la creación de personajes inteligentes. Finalmente en el Capítulo 3 se muestran los resultados de la extensión desarrollada, y se exponen las pruebas que se le realizaron.

1. Fundamentación Teórica

En este capítulo se abordarán temas referentes a los videojuegos, profundizando en los diferentes tipos, arquitectura, así como la IA presente en los mismos. Se verá el comportamiento de IA en enemigos en los videojuegos de aventura y acción para combates en tiempo real. Se definirán, además, las metodologías y herramientas necesarias para el desarrollo de una propuesta de solución.

1.1 Diferentes Sistemas en los Videojuegos.

A la hora de crear un videojuego, las compañías desarrolladoras toman en cuenta las tendencias a nivel mundial referentes a los gustos de los usuarios y técnicas empleadas, estilos de videojuego. En la actualidad la mayoría de los videojuegos son cada vez más complejos, ya sea por su dificultad, nivel gráfico, historia, incluso por la IA presente en ellos.

Esto les permite ser clasificados en diferentes géneros teniendo en cuenta además factores como el sistema de videojuego, el tipo de interactividad con el jugador, sus objetivos, etc. A medida que han ido evolucionando los videojuegos, ha surgido una gran variedad de géneros, a veces, en relación con los avances que la tecnología permite. Entre los géneros de videojuegos más populares están los de acción, estrategia, rol, aventura, rompecabezas, simulación, deporte y otros, cada uno de ellos con varios subgéneros. Por otro lado, hoy en día hay videojuegos que combinan elementos de más de un género, dando lugar a géneros mixtos (por ejemplo, rol - acción, aventura - acción, etc.).

Existen además otras formas de caracterizar los videojuegos como puede ser por su temática (fantástico medieval, futurista, de guerra), su complejidad (videojuegos AAA, videojuegos casuales), su finalidad (educativos, promocionales, artísticos), etc.

Por otra parte, también se diferencian unos videojuegos de otros, incluso dentro de un mismo género, por la perspectiva visual que adoptan (la posición de la cámara). Así, hay videojuegos con perspectiva 2D (ya sea con proyección paralela, vista lateral o vista cenital), 2.5D (mediante proyección isométrica, oblicua, entre otras), y 3D (en perspectiva fija, en primera persona, o en tercera persona) (Facultad de Diseño y Comunicación., 2014)

1.1.1 Arquitectura de un videojuego.

Las compañías desarrolladoras de videojuegos presentan sus propias arquitecturas de videojuego, las cuales no se dan a conocer al público, siendo esto un secreto y su mayor fortaleza, puesto que mientras más fuerte y sólida sea la estructura con la que se programa el videojuego, mayor será el prestigio de la empresa. No obstante, teniendo en cuenta los estudios hechos a una serie de videojuegos de acción, tales como *Killing Floor, World Of Warcraft, Assassin's Creed*, siendo este estilo de videojuego el principal para lo que se desarrollará el presente trabajo; se puede llegar a una arquitectura base de videojuego, partiendo esta de un *Game Manager*. Esto se hace antes de comenzar a programar el videojuego para saber en qué parte se debe atacar, dónde poner un *script*, cómo configurarlo, etc. El *Game Manager* es el corazón del videojuego, este contiene todas las clases públicas y objetos que van a controlar todo, ya sea, cambiar una escena, guardar o salvar el videojuego; son métodos globales que van a servir a lo largo del mismo, generalmente es un *don't destroy on load*, esto es para que no se destruya en todo el videojuego y perdure el flujo de información entre las clases. De aquí se derivan el *UI Manager*, el *Sound Manager* y el *Data Manager*, siendo estas las principales clases administradores de un videojuego. El *UI Manager* es el que controla todo lo que se mostrará en la interfaz de usuario, que a la vez es de conocimiento del *Game Manager*, el *Sound Manager* controla todo lo referente a sonido, ya sean efectos, ambiente, banda sonora y voces. El *Data Manager* se encarga de controlar todos los datos del videojuego, todas las variables que se van a guardar o que se van a retener. Todas estas clases van a ser controladas por el *Game Manager*. Partiendo de esto, existen clases separadas que van a coexistir por sí solas, pero van a hacerle preguntas al *Game Manager* e interactuar con él de esta forma, por ejemplo, tenemos la clase *character*, que de ella se derivan los *players* y los *NPC*. De los *NPC* se derivan dos tipos, los *hostile*, que son los enemigos con los que se encuentra el jugador y los *friendly*, con los que interactúa, recibiendo misiones o información por parte de estos. Existen también las clases *Chronos Manager* y *Collision Trigger Event*. *Chrono Manager* se encarga de controlar los efectos de atmósfera, es decir, si cambia el clima, si está nublado el día, si hay sol, si es de noche, etc. *Collision Trigger Event* se encarga de controlar los disparadores de eventos de colisiones, por ejemplo, el jugador se encuentra con una palanca, la baja y esto es un disparador que activa algo específico, pasa por un puente, este se destruye, o sucede alguna otra acción, y se activa una cinemática que va en ese momento. Son objetos vacíos completamente que el usuario no ve, pero están presentes en el escenario, que van a ser un tipo de colisión y van a

disparar un evento en algún momento. De manera general, esta viene siendo la estructura principal del videojuego, no obstante, hay otras clases y estructuras más específicas que dependen del tipo de videojuego que se va a realizar y el propósito que este tenga (Pérez Ozete, 2016)

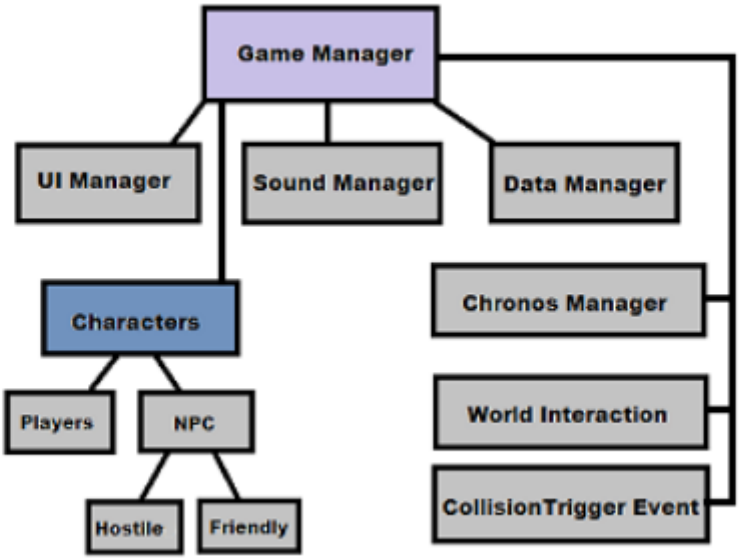


Figura 1.1 Arquitectura de Videojuego.

El presente trabajo se enfoca principalmente a la clase *NPC*, siendo esta sobre la cual se realizará la extensión para la plataforma *Unity*. Creando grupos de personajes y asignándoles los comportamientos grupales para incluirlos a la hora de realizar un videojuego de aventura y acción para combates en tiempo real y sean capaces de actuar como un equipo y derrotar o infligir el mayor daño posible al jugador.

1.1.2 Géneros de Videojuegos

A diario se crean videojuegos o se mejoran los ya existentes para complacer los gustos de los millones de usuarios alrededor del planeta. Naturalmente, los videojuegos simulan entornos y situaciones virtuales en los que el jugador controla a uno o varios personajes (o cualquier otro elemento de dicho entorno), para superar una o varias metas u objetivos definidos ya por las reglas del videojuego.

Teniendo en cuenta el tipo de juego en cuestión, una partida puede ser efectuada entre el usuario y la máquina, dos o más personas en la misma, e incluso múltiples jugadores

conectados a una red *Local Area Network* (*LAN*) o en línea por Internet, compitiendo, de esta manera entre sí o de forma cooperativa contra el ordenador.

Desde que surgieron los videojuegos, sus realizadores han ido creando una amplia gama de géneros con tal de ganar más seguidores y satisfacer los gustos de todos los usuarios.

El término género de un videojuego, se refiere a la manera en cómo se pueden clasificar los mismos, dependiendo fundamentalmente a la mecánica de juego, aunque existen otros factores como son la temática que representa, la estética visual, los cuales influyen a la hora de definir un género en específico. Algunos de los géneros más representativos son los videojuegos de acción, rol, estrategia, simulación, deportes, aventura.

El presente trabajo se enfoca solamente en los videojuegos de los géneros de acción y de aventura, puesto que el componente a desarrollar es para los comportamientos grupales en estos videojuegos para combates en tiempo real. Es por esta razón que se define como videojuego de acción al tipo de videojuegos en los que el jugador tiene que combinar ciertas habilidades como son la agilidad, destreza, velocidad y tiempo de reacción. Este género es el más amplio de todos, abarcando subgéneros como videojuegos de lucha, videojuegos de disparos en primera persona y videojuegos de plataformas. Estos subgéneros se caracterizan por usar la violencia como principal atractivo o rasgo de interactividad, principalmente en combates con armas de fuego o cuerpo a cuerpo. (El Otro Lado, 2017)

Por otro lado, los videojuegos de aventura fueron, en cierto modo, los primeros videojuegos que se vendieron en el mercado, empezando por *Colossal Cave Adventure* en los años 1970. Este tipo de videojuego se hizo famoso con los videojuegos de la serie *Zork* y consiguió alcanzar cierto nivel de popularidad en los años 80 que duró hasta mediados de los 90. El jugador encarna a un protagonista que por lo general debe resolver incógnitas y rompecabezas con objetos diversos. Los primeros videojuegos de aventura eran textuales (aventuras textuales, aventura conversacional o ficción interactiva). En estos, el jugador utiliza el teclado para introducir órdenes como “coger la cuerda” o “ir hacia el oeste” y el ordenador describe lo que pasa. Cuando el uso de gráficos se generalizó, los videojuegos de aventura textuales dejaron paso a los visuales (por ejemplo, con imágenes del lugar presente) que substituyeron de este modo las descripciones por texto, que se habían vuelto casi superfluas. Estos videojuegos de aventura con gráficos seguían, no obstante, sirviéndose de la introducción de texto.

Además, sigue existiendo una comunidad de autores y jugadores activos de ficción interactiva, aunque la participación de grandes empresas comerciales es más bien rara. (Facultad de Diseño y Comunicación., 2014)

Como subgénero de los videojuegos de aventura se encuentran los de aventura de rol. Estos se caracterizan por la interacción con el personaje, una historia profunda y una evolución del personaje a medida que la historia avanza. Para lograr la evolución generalmente se hace que el jugador se enfrasque en una aventura donde irá conociendo nuevos personajes, explorando el mundo para ir juntando armas, experiencia, aliados e incluso magia. La inclusión del *CD-ROM* permitió contar la historia más detallada, utilizando videos de duración media que hacen que el jugador se sienta como dentro de una película.

Aunque la mayoría de videojuegos de aventura incluyen una dosis baja de *Role Playing Games* (*RPG*), los videojuegos de rol puros se enfocan específicamente en subir experiencia y personalización del personaje, en juegos como la saga *The Elder Scrolls*, el crear y personalizar tu personaje puede llevar hasta 30 horas. Los *RPG* clásicos, inspirados en los juegos de tablero, realizan las batallas por turnos, es decir, el jugador usa su equipo y habilidades aprendidas para atacar mediante una serie de comandos y después debe quedar estático y esperar a recibir el ataque del otro jugador o de la IA. El mejor ejemplo de esto es *Final Fantasy*, y *Dungeons & Dragons.*

Si bien en la actualidad todavía existen videojuegos de rol clásicos, la mayoría ahora usa el combate en tiempo real, es decir, no hay pausas y ambos atacan al mismo tiempo. Aunque la mayoría de videojuegos de rol están basados en mundos medievales fantásticos al estilo del Señor de los Anillos de *Tolkien* o *Dungeons & Dragons*, hasta considerar videojuegos de rol sólo aquellos que tuviesen relación con este tipo de realidades. Existen videojuegos que usan otro tipo de trasfondos como *Mass E*ff*ect*, que está basado en un mundo posterior al viaje extraplanetario o *Fallout* que está basado en un universo postnuclear.

En otra subcategoría de los videojuegos de rol se han puesto de moda los *RPG* en línea o *Massive Multiplayer Online RPG* (*MMORPG*) donde cada jugador crea un personaje y mediante una conexión a internet, entra a un mundo donde miles de jugadores se unen a la aventura, exploran, intercambian y evolucionan juntos. Hoy por hoy el juego más conocido y jugado de este subgénero es *World of Warcraft*, basado en el mundo creado por *Blizzard* para sus juegos de estrategia de la saga *Warcraft*. También existen otros

videojuegos conocidos de este género como *Final Fantasy XI, Warhammer online, EVE* o *RuneScape*. Estos juegos también son conocidos por lo adictivos que acostumbran ser además de la gran cantidad de cultura popular que suelen generar a su alrededor (Facultad de Diseño y Comunicación., 2014).

El objetivo principal en estos géneros de videojuegos es superar una serie de niveles hasta llegar al último con el jefe principal, para ello, el jugador, debe ir superando oleadas de enemigos, incluyendo mini jefes. Un mini jefe es quien da paso al final de un nivel o serie de niveles.

Para vencer a los jefes se utiliza el reconocimiento de patrones y la velocidad de reacción física. En la mayoría de los videojuegos, los jefes se programan mediante un patrón de ataques simple o con movimientos que el jugador aprende a través de la experiencia. Estos patrones de ataque regularmente contienen movimientos especiales que requieren de las habilidades del jugador para que este sea capaz de saltar, esquivar o bloquear ataques para luego golpear en ciertos puntos claves, siempre teniéndose un control sobre el manejo de los patrones para atacar. (Creative Commons, 2018)

Los jugadores buscan sentir nuevas emociones, nuevos desafíos. Es por esto que los desarrolladores deciden emplear las técnicas de IA en la mejora de los personajes, haciéndolos capaces de actuar por sí solos e incluso presentar cierto grado de dificultad a la hora que el usuario se enfrente a ellos.

1.1.3 Inteligencia Artificial para los Videojuegos.

La industria de los videojuegos ha servido como base para probar y desarrollar muchas técnicas de IA; beneficiándose a la vez con realistas modelos de inteligencia en sus productos de entretenimiento, que los hacen más atractivos para los usuarios.

Con la gran cantidad de videojuegos que existen hoy en el mundo, es posible afirmar que la clave del éxito entre dos videojuegos de similares características visuales, se encuentra precisamente en la capacidad de entretener al usuario, presentándole rivales capaces de retar sus habilidades con un adecuado nivel de inteligencia.

La principal diferencia entre la IA clásica y la que es desarrollada para videojuegos radica en la optimización de los resultados. La primera trata de hallar la mejor solución

a un problema, sin importar el costo computacional; mientras que la segunda, limitada por la disponibilidad de recursos para realizar sus cálculos en un videojuego en tiempo real, muchas veces opta por obtener una solución aceptable pero no óptima. Por otra parte, los videojuegos deben ser interesantes y divertidos, no imposibles de ganar, por lo que se debe mantener un balance adecuado. Es por esto que en ocasiones hay que conseguir un punto intermedio entre lo inteligente que puede ser un jugador virtual y el tiempo que se necesita para calcular su comportamiento.

En la implementación de la IA para videojuegos en ocasiones se realizan ciertos trucos o trampas para que el sistema pueda parecer lo más “vivo” posible, pero siempre alcanzando un compromiso entre la calidad de las decisiones y el tiempo necesario para calcularlas. Otra diferencia que existe con la IA clásica es que, en ocasiones, hay que lograr que la IA sea más humana. Por ejemplo, conseguir que de vez en cuando el sistema sea capaz de fallar, aun cuando este pueda calcular la trayectoria perfecta para acertar siempre. En otro caso, el jugador se aburriría si ve que nunca es posible superar a un enemigo controlado por la IA. (Homer Reynoso, 2009)

La calidad de la IA del videojuego se controla muchas veces mediante diferentes niveles de dificultad, que básicamente indican hasta qué punto se debe dotar de inteligencia a los elementos y cuántas probabilidades existen de que el comportamiento que se calcule como el mejor se lleve a cabo efectivamente.

Las técnicas a utilizar dependen mucho del tipo de videojuego que se esté diseñando, de la importancia que se le desee dar a la IA dentro del videojuego, así como de la cantidad de recursos que se encuentren disponibles para ella. Estas se pueden clasificar en:

- **Deterministas:** El comportamiento del agente inteligente es especificado por completo; o sea, los programadores tienen que codificar todas las acciones explícitamente, dificultando y retrasando el proceso de desarrollo del videojuego. Las más usadas son las máquinas de estado finito, los árboles de decisión, y los sistemas de reglas de producción.
- **No deterministas:** Es todo lo contrario, el grado de incertidumbre es mayor, por lo que no se puede predecir el comportamiento que seguirá el agente. Generalmente usan técnicas como redes neuronales, algoritmos evolutivos o redes bayesianas, que facilitan el aprendizaje y la adaptación al entorno de los elementos inteligentes. No hay que codificar explícitamente todas las posibles

situaciones en el videojuego, ya que los elementos inteligentes pueden incluso extrapolar sus comportamientos y desarrollar otros nuevos o emergentes (Inteligencia Artificial en Videojuegos, 2008)

Técnicas Deterministas.

Los desarrolladores de videojuegos han experimentado con la mayoría de las técnicas de IA; pero sin dudas las más sencillas, eficientes, fáciles de implementar, entender y depurar son las deterministas, por lo que han sido las más usadas en este campo. Además, el tiempo del que disponen las compañías para producir un videojuego es insuficiente para que los desarrolladores (más enfocados en la calidad de los gráficos) se decidan a experimentar con técnicas de IA no deterministas que son muy difíciles de entender, implementar y probar.

De este grupo de técnicas se utilizan frecuentemente las máquinas de estado finito, que definen una serie de estados en los que puede permanecer un elemento, así como las condiciones para que se produzcan transiciones entre ellos. Muchas veces se combinan con lógica difusa, para representar en lenguaje computacional conceptos imprecisos o nociones subjetivas como “cercano/lejano”.

En el videojuego “*The Sims*”, la IA es implementada con máquinas de estado finito difusas, además utilizan técnicas específicas de *A-Life2* para simular el comportamiento de organismos vivos (en este caso, personas que viven en familia). La base de la inteligencia en este videojuego es su motor de comportamiento, el cual asocia acciones posibles a cada objeto. Un *NPC* debe ser capaz de reconocer y comprender el espacio que le rodea, empleando para ello técnicas de percepción. Además debe ser capaz de buscar caminos para navegar, por lo que se le presta una especial atención a los algoritmos de búsqueda como *Dijkstra* o A*.

Los videojuegos de tablero como el ajedrez y el *backgammon* han usado árboles de búsqueda heurística con formidables resultados. Tradicionalmente en los videojuegos se han utilizado técnicas de planificación o guiones (del inglés *script*) para definir la conducta de los agentes, al igual que los sistemas basados en reglas y los comportamientos grupales o de manadas. Los sistemas expertos son un tipo de sistema basado en reglas que definen el conocimiento para que los agentes autónomos se comporten de manera similar a un jugador experto (Inteligencia Artificial en Videojuegos, 2008).

Técnicas no Deterministas.

En la actualidad el poder computacional se ha incrementado notablemente. Existen potentes tarjetas gráficas que liberan al procesador de la máquina para que pueda encargarse de otras tareas dentro del videojuego, por ejemplo la física y la IA. Esto, unido a la demanda de los usuarios de videojuegos más desafiantes, al deseo de los desarrolladores de lanzar un videojuego que marque la diferencia, así como al mayor interés de los "académicos" de la IA en los videojuegos como área de experimentación relativamente barata y sin riesgos, ha posibilitado que se comiencen a aplicar más frecuentemente técnicas no deterministas.

El reto actual de los desarrolladores es crear videojuegos novedosos, divertidos, realistas y con mayor vida útil. Un buen paso en ese sentido es lograr que los *NPC* aprendan, evolucionen, se adapten a nuevas situaciones y exhiban un comportamiento genuino. Estos resultados son posibles de alcanzar con métodos no deterministas que algunos desarrolladores investigan con gran interés a pesar de su complejidad.

Se han alcanzado excelentes resultados en videojuegos que usan redes neuronales, algoritmos genéticos o métodos probabilísticos como *Dirt Track Racing*, *Creatures*, *Black & White*, *Battlecruiser 3000AD*. Generalmente se combinan con métodos deterministas más tradicionales y estudiados, conformando una especie de sistemas híbridos (ibíd.).

1.2 Inteligencia Artificial en Videojuegos de Aventura y Acción.

Hoy en día la mayoría de los videojuegos incorporan algún tipo de IA. Los desarrolladores han usado durante años la IA para dar vida a personajes aparentemente inteligentes en innumerables videojuegos, principalmente los personajes enemigos. En los videojuegos no siempre se da a los personajes no jugadores inteligencia de nivel humano. Tal vez se escribe código para controlar las criaturas no humanas, tales como dragones, robots, o incluso roedores; hacer algunos personajes no inteligentes puede agregar al videojuego variedad y riqueza.

Aunque es cierto que a menudo la IA de los videojuegos está llamada a resolver problemas bastante complejos, la IA se puede emplear para cosas más simples como tener personajes con diferentes personalidades o que actúen según nuestras acciones. Es aquí donde juega un papel importante la inclusión de la IA en los enemigos *NPC*,

logrando que estos actúen, ya sea, individuales o como equipo para atacar al jugador y logren causarle el mayor daño posible, haciendo esto que el usuario gane interés en el videojuego y quiera superarse y vencer a estos enemigos.

Este tipo de *NPC* puede presentar comportamientos pasivos en algunos casos, aunque generalmente poseen comportamientos hostiles hacia el jugador, sirviendo esto como desencadenante de peleas entre jugador humano y personaje no jugador, ayudando a incrementar los puntos de habilidades y experiencia al personaje del usuario (Marrero, 2010).

1.2.1 Comportamiento de IA en Videojuegos de Aventuras y Acción.

En los combates llevados a cabo en estos tipos de videojuegos, la IA debe ser eficiente y más humana, o al menos parecerlo. Una de las características más importantes que debe poseer en este tipo de videojuegos es la habilidad para cazar. En un principio la IA reaccionaba de manera sencilla, por ejemplo, si el jugador se encontraba en un área específica, esta reaccionaba, ya sea de manera ofensiva o defensiva.

No obstante, a medida que avanza el desarrollo de los videojuegos y la IA en los *NPC* se hace cada vez más fuerte y casi real, la idea de caza ha tomado auge. En esta, la IA buscará marcadores, tales como los sonidos hechos por el personaje o huellas que pudieron haber dejado atrás. De esta forma, el jugador puede tener en cuenta si se acerca o evita a un enemigo.

Recientemente se ha incorporado a los comportamientos de IA en estos tipos de videojuegos el llamado instinto de supervivencia. Así, el enemigo puede reconocer diferentes objetos en el área donde se encuentre y determinar si les son útiles o no para su supervivencia. Al igual que el usuario, la IA se cubre, chequea su estado de salud y se defiende. Esta establece una serie de marcadores que le ayudan a reaccionar de una forma determinada.

Por ejemplo, si en la IA se ejecuta un comando para comprobar los puntos de vida a través de un combate, a continuación, otros comandos se pueden activar de forma tal que reaccione de manera específica a un cierto porcentaje de salud. Si la salud está por debajo de un nivel determinado, entonces la IA del *NPC* se puede configurar para que este se aleje del jugador y lo evite hasta que otra función se active. Este comportamiento en los *NPC* hace que luzcan más realistas y humanos, aunque todavía existe la

necesidad de continuar la mejora de estos. Trayendo como principal limitante a la hora de sorprender al jugador, que la IA necesita ser programada para todos los escenarios posibles (Artificial Intelligence in Game Design, 2009).

1.2.2 Comportamiento de IA en Enemigos para Ataques Cuerpo a Cuerpo.

Los *NPC* enemigos presentan una serie de movimientos y combinaciones a la hora de enfrentarse en un ataque cuerpo a cuerpo. En algunos videojuegos, estos tienen un solo ataque, mientras que en otros, poseen múltiples ataques realizando combos de golpes. Así se evidencian algunos factores importantes como el daño que son capaces de producir, la capacidad de bloquear o interrumpir los ataques del jugador y defenderse.

Muchos de los videojuegos ofrecen al usuario la ventaja de enseñarle cómo pelear contra los enemigos, pero hay casos de videojuegos que no presentan esto y resultan ser un tanto más difíciles de jugar, ganando de esta manera seguidores que gustan de los retos y de la experiencia de usuario. Existe una característica fundamental en la mayoría de los enemigos, la cual se conoce como “aviso”, esta se basa en hacer saber al jugador que están a punto de atacar, siendo esto importante a la hora de entender la legibilidad del sistema de combate.

Mientras más fuerte sea el enemigo y mayor repertorio de ataques posea, más claro será el “aviso” posibilitando que el jugador responda, ya sea, evitando el ataque, bloqueándolo, o simplemente respondiendo y pelear contra el enemigo. Esta característica en algunos videojuegos, permite al jugador prepararse para el ataque, incluso le muestra cómo evitar o bloquear algunos golpes. En otros videojuegos, se hace más difícil de evidenciar, puesto que hay un mayor número de enemigos en un mismo ataque y el jugador debe estar a la expectativa, siendo esto un grado de dificultad en el videojuego, lo cual llama la atención de los jugadores (Enemy design and enemy AI for melee combat systems., 2015).

Es importante que, en este tipo de combate, la IA de los enemigos se diseñe con una serie de propósitos, que posibilite que el *NPC* enemigo desafíe al jugador constantemente, esto es con el objetivo de que el usuario se vea cada vez más motivado por el videojuego. Existen cuatro roles o formas principales para clasificar y separar a los enemigos en los combates cuerpo a cuerpo, estas son:

- **Enfatizadores:** estos enemigos se enfocan en hacer que los jugadores utilicen contra estos los ataques y habilidades específicas para derrotarlos, no obstante, es posible vencerlos además de manera regular.
- **Ejecutores:** estos enemigos hacen que el jugador se vea obligado a usar un tipo específico de ataque y de armamento para derrotarlos, siendo los demás ataques inútiles contra este tipo de personajes.
- **Destrozadores:** este tipo de enemigo es más fácil de derrotar y presenta la característica de permitir al jugador divertirse destrozándolos.
- **Retadores:** esta clase es la más temeraria, puesto que constantemente están desafiando al jugador y probando sus habilidades, esto conlleva que el usuario se esfuerce a la hora de derrotar a este enemigo.

Estas clases se combinan en los videojuegos de ataque cuerpo a cuerpo para dar una mayor experiencia de usuario, usualmente se presentan los enemigos que más fácil se pueden derrotar, brindando esto confianza al jugador, después se combinan los de dificultad media y fuerte, haciendo que el jugador pruebe una serie de ataques y armamentos específicos y aprenda cuáles debe usar para poder vencer a estos *NPC* (Artificial Intelligence in Game Design, 2009).

De esta manera, se busca llamar la atención de los usuarios, haciendo más atractivo el modo de videojuego, aumentando la dificultad y ganando seguidores a lo largo del mundo del videojuego.

1.2.3 Comportamiento de IA en Enemigos para Ataques a Distancia.

En este tipo de ataques, la IA en los enemigos cambia, llevándolos a no enfocar la atención en los ataques cuerpo a cuerpo. De esta forma se diferencian los enemigos unos de otros al darles un arma característica de cada uno, tanto para los enemigos en combates cuerpo a cuerpo como para los enemigos de combates a distancia.

En la mayoría de los videojuegos que incluyen ataques a distancia, el jugador pelea en primera persona. Mientras que este está ocupado peleando con enemigos en un combate cuerpo a cuerpo, pueden aparecer enemigos en el ataque que no usen esta característica, sino que se centren en atacar desde lejos al jugador, con ataques esporádicos pero que combinados con los de los otros enemigos pueden infligir bastante daño al personaje.

En estas combinaciones de ataque entre enemigos de uno o varios tipos, ya sea cuerpo a cuerpo o distancia, la IA juega un papel importante, ya que se encarga de controlar cómo serán los ataques y quiénes los realizaran, puesto que los enemigos están conscientes de que hay otros atacando y estos pueden o esperar o atacar desde lejos. Dando paso así a los comportamientos grupales en los ataques combinando diferentes enemigos (ibíd.).

1.3 Comportamientos Grupales en Ataques para Videojuegos de Aventura y Acción.

Existen videojuegos de acción y aventura en que los enemigos *NPC* se agrupan, forman equipos y se coordinan, resultando ser impredecibles en algunas ocasiones, dificultando su derrota por parte del jugador. Estos equipos pueden estar conformados entre 3 y 5 *NPC*, con ciertas características y habilidades que juntos, ponen resistencia al usuario. Entre estos personajes, se encuentran tres tipos básicos, los de ataques cuerpo a cuerpo, los de ataque a distancia y los de magia.

Un grupo conformado por personajes de estos tres tipos puede tener varias formas de coordinarse. Los *NPC* de ataque cuerpo a cuerpo pueden estar rodeando al personaje principal, atacar y moverse a su alrededor, de forma tal que el jugador se enfoque en ellos mientras que los *NPC* de ataque a distancia usan sus habilidades para atacar, ya sea con diferencia de algunos segundos entre los ataques de los enemigos de ataque cuerpo a cuerpo como simultáneamente y variar su posición o permanecer estáticos. Junto a estos, los magos pueden lanzar ataques que ralenticen al jugador, o lo debiliten, de manera que puedan curar a sus compañeros. Siendo esta una de las tácticas de ataque más empleada en la mayoría de los videojuegos, tales como *WorldOfWarcraft, DarkSould, DungeonSide*, etc.

Figura 1.2 Comportamiento Grupal en Ataques.

En videojuegos tales como *Reckoning*, como se muestra en la imagen, los grupos de enemigos son pequeños, no exceden los cinco integrantes. Estos pueden o no caracterizarse como un equipo de enemigos, es decir, pueden atacar por iniciativa propia y tener libre movimiento o tener uno o varios jefes que digan cuándo atacar y a quién atacar. La mayoría de los ataques ocurren de manera grupal, caracterizándose por existir dos grupos, uno cercano al jugador y otro más distante. El grupo más cercano de enemigos, generalmente compuesto por algunos miembros, representa la amenaza más inmediata que debe enfrentar el jugador. Mientras que el otro grupo que se encuentra más allá del jugador, contiene el resto de los enemigos. De esta manera los enemigos se pueden coordinar entre sí para derrotar al jugador, ya sea realizando combos de ataques entre enemigos de corto, mediano y largo alcance. Estos ataques son coordinados mediante máquinas de estado o arboles de decisión, siendo estos los principales algoritmos de IA encargados de controlar el tiempo entre ataques y cómo serán las combinaciones usadas para vencer o intentar derrotar al jugador. Intercambiándose de esta forma los enemigos de ambos grupos, moviéndose alrededor del personaje, ajustándose al movimiento del mismo (Enemy design and enemy AI for melee combat systems., 2015).

1.4 Metodologías de Desarrollo.

El desarrollo de un *software* de alta calidad, en el tiempo planificado, con los menores costos posibles y que además satisfaga las necesidades y expectativas del cliente, requiere que durante todo el ciclo de vida de este, se trabaje de forma organizada. Para esto se debe controlar y documentar toda la información relacionada con el proyecto, prever los posibles riesgos que se puedan presentar durante su ejecución y definir las medidas para mitigarlos. La correcta realización de estas tareas depende en gran medida del empleo de una metodología eficaz que se adapte a las características del producto deseado. Es por ello, que un cambio en algún momento del desarrollo de un *software*, que no sea la fase inicial cuando se realiza la captura de requisitos, puede devenir en atrasos en los cronogramas trazados o pérdidas monetarias. En base a este problema existen metodologías que son en cierta medida más o menos sensibles a estos cambios. Las mismas son conocidas como **tradicionales**, para el caso en que estas son mucho menos adaptables a un cambio en los requerimientos y **ágiles**, en el caso que estas sean más abiertas al mitigar atrasos o pérdidas monetarias por un cambio en la concepción del producto por parte del cliente (Pressman, 2010).

Las metodologías tradicionales, están guiadas por una rigurosa planificación durante todo el proceso de desarrollo, en el cual se establecen estrictamente las actividades a realizar. Están orientadas a proyectos de gran envergadura, para los cuales se definen una gran cantidad de roles y artefactos a generar, contando con una detallada documentación (Canós, 2003). Entre estas metodologías se destaca *Rational Unified Process* (*RUP*).

- *RUP*: Basada en componentes e interfaces bien definidas. Esta metodología define un marco de trabajo genérico, el cual puede especializarse para una gran variedad de sistemas de *software*, en diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto. Esta comprende tres principios claves: dirigido por casos de uso, centrado en la arquitectura e iterativo e incremental. Esto significa que los casos de uso describen los requisitos funcionales del sistema, desde la perspectiva del usuario. Estos no solo inician el proceso de desarrollo sino que también proporcionan un hilo conductor, en el que se verifica, luego de la implementación, que el producto implemente adecuadamente cada caso de uso. Por otra parte, la arquitectura de un sistema permite tener una visión común entre los desarrolladores y los usuarios, teniendo una vista del diseño completo del *software* con las características más importantes resaltadas. Además *RUP* propone un proceso iterativo e incremental, donde el trabajo se divide en partes más pequeñas. Cada una de estas partes se puede ver como una iteración de la cual se obtiene un incremento que produce un crecimiento en el producto.

Las metodologías ágiles, son aquellas que están orientadas a la producción de código, con ciclos muy cortos de desarrollo. Se dirigen por grupos pequeños, haciendo hincapié en aspectos humanos asociados al trabajo en equipo. Están orientadas a proyectos pequeños, los cuales pueden presentar cambios durante su realización e involucran activamente al cliente en el proceso de desarrollo (Canós, 2003). Entre estas metodologías se encuentran:

- Programación Extrema o *Extreme Programming* (*XP*): Basada en la simplicidad, la comunicación y la reutilización de código. Esta metodología trata de darle al cliente el *software* que él necesita y cuando lo necesita, lo que implica responder rápidamente a las necesidades de este. Además tiene como objetivo potenciar al máximo el trabajo en grupo, donde los clientes y los desarrolladores son parte del

equipo de trabajo y están involucrados en el proceso de desarrollo del *software* durante todo su ciclo de vida. Esta metodología de desarrollo de *software* se recomienda para proyectos de corto plazo, poniendo más énfasis en la adaptabilidad que en la previsibilidad. Es definida especialmente como una metodología adecuada para proyectos con requisitos imprecisos y muy cambiantes. Esta metodología intenta reducir la complejidad del *software* por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción (Fernández, 2002)

- SCRUM: Esta metodología requiere trabajo duro puesto que no se basa en el seguimiento de un plan, sino en la adaptación continua a las circunstancias de la evolución del proyecto. Esta metodología además, está especialmente indicada para proyectos con un rápido cambio de requisitos. El desarrollo de *software* se realiza mediante iteraciones, denominadas *sprints*, con una duración de 30 días. Donde el resultado de cada *sprint* es un incremento ejecutable que se muestra al cliente. Otro aspecto importante de esta metodología son las reuniones diarias de 15 minutos que esta propone. En estas se debate lo que se ha hecho desde la última reunión, los posibles obstáculos que atentan contra la productividad del equipo, así como las tareas que se realizarán antes de la siguiente reunión (Pressman, 2010).

1.5 Herramienta de Desarrollo.

Unity es un motor de videojuegos para PC (Personal Computer o Computadora Personal) y Mac que viene empaquetado como una herramienta para crear juegos, aplicaciones interactivas, visualizaciones y animaciones en 2D, 3D y en tiempo real. Unity puede implementar contenido para múltiples plataformas como PC, Mac, Nintendo Wii e iPhone. También puede publicar juegos basados en web usando el componente Unity Web Player.

El contenido del videojuego es construido desde el Editor del programa usando scripts. Esto significa que los desarrolladores no necesitan ser expertos en C++ para crear videojuegos con Unity, ya que las mecánicas de juego son compiladas usando una versión de JavaScript o C#. Unity utiliza además mallas de navegación, estas son estructuras de dato abstractas, utilizadas en aplicaciones de IA para asistir a los agentes en la búsqueda de caminos a través de espacios complicados (Technologies, 2017).

1.5.1 Extensión e interfaces.

Una extensión es un programa que puede anexarse a otro para aumentar sus funcionalidades (generalmente sin afectar otras funciones ni afectar la aplicación principal). No se trata de un parche ni de una actualización, es un módulo aparte que se incluye opcionalmente en una aplicación (Pérez Ozete, 2016).

En Unity, para crear una extensión, se utiliza el Editor, de esta manera se podrá añadir una nueva funcionalidad a la barra de herramientas. Para ello se deben seguir tres pasos (Technologies, 2017):

- Crear un script que se derive de la Ventana del Editor.
- Usar el código para activar la ventana y mostrarla.
- Implementar el código GUI para la herramienta que se está desarrollando.

Script que deriva de la Ventana del Editor.

Para crear una nueva ventana en el Editor, el script debe ser almacenado adentro de una carpeta llamada “Editor”. Luego se debe hacer una clase en este script que derive de la Ventana del Editor para establecer los controles GUI en el interior de la función OnGUI (Technologies, 2017). De esta manera:

```
//C# Example

using UnityEngine;
using UnityEditor;
using System.Collections;

public class Example : EditorWindow

    {
        void OnGUI () {
            // The actual window code goes here
           }
    }
```

Figura 1.3 Ejemplo de Cómo Derivar la Ventana del Editor. Tomado de la ayuda de Unity (Technologies, 2017)

Esto creará una ventana de editor estándar acoplable, que guarda su posición entre las invocaciones. Este fragmento puede ser utilizado en diseños personalizados.

Implementar el GUI de la Ventana.

Los contenidos de la ventana son visualizados al implementar la función OnGUI. Se pueden utilizar las mismas clases UnityGUI que usa el GUI dentro del par (GUI y GUILayout). Adicionalmente, se proporcionan algunos controles GUI, ubicados solamente en las clases del editor EditorGUI y EditorGUILayout. Estas clases se suman a los controles que ya están disponibles en las clases normales, por lo que se pueden mezclar y combinar a voluntad (Technologies, 2017). De esta manera:

```
//C# Example
using UnityEditor;
using UnityEngine;

public class MyWindow : EditorWindow
{
    string myString = "Hello World";
    bool groupEnabled;
    bool myBool = true;
    float myFloat = 1.23f;

    // Add menu item named "My Window" to the Window menu
    [MenuItem("Window/My Window")]
    public static void ShowWindow()
    {
        //Show existing window instance. If one doesn't exist, make one.
        EditorWindow.GetWindow(typeof(MyWindow));
    }
    void OnGUI()
    {
        GUILayout.Label ("Base Settings", EditorStyles.boldLabel);
        myString = EditorGUILayout.TextField ("Text Field", myString);

        groupEnabled = EditorGUILayout.BeginToggleGroup ("Optional Settings", groupEnabled);
            myBool = EditorGUILayout.Toggle ("Toggle", myBool);
            myFloat = EditorGUILayout.Slider ("Slider", myFloat, -3, 3);
        EditorGUILayout.EndToggleGroup ();
    }
}
```

Figura 1.4. Ejemplo de Cómo Agregar Elementos GUI a la Ventana del Editor. (Technologies, 2017)

Una vez concluida la implementación de la nueva ventana, esta quedaría así:

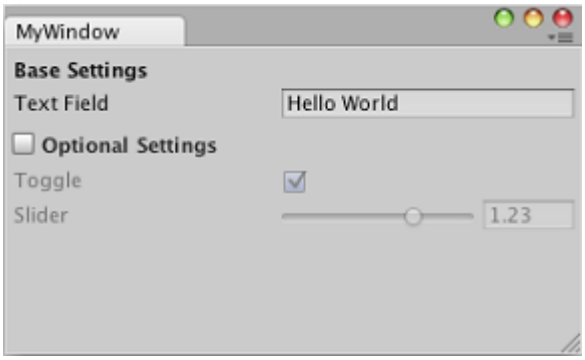


Figura 1.5. Ejemplo de Cómo Quedaría la Ventana (Technologies, 2017).

1.6 Entorno de Desarrollo.

Unity integra MonoDevelop como entorno de desarrollo ya que es libre y gratuito y este fue diseñado primordialmente para C# y otros lenguajes .NET. Además este es el entorno de desarrollo incluido por defecto en Unity (Technologies, 2017). Características de MonoDevelop que potencian su uso:

- Ambiente sumamente intuitivo y simple.
- La ayuda es muy completa e incluye una amplia gama de ejemplos.
- Posee autocompletado de sintaxis.
- Posee un navegador incorporado.
- Es multiplataforma.

La extensión se realizará sobre esta y una vez terminado podrá ser integrado a la misma sin presentar problemas de compatibilidad. Utilizando las mallas de navegación y sus propiedades para que los personajes creados puedan moverse en la escena.

1.6.1 Lenguajes de Programación

El motor de videojuegos Unity brinda a los programadores la posibilidad de contar con varios lenguajes de programación para el desarrollo de aplicaciones (Technologies, 2017). Estos lenguajes son:

- **C# (C Sharp):** Es el lenguaje de mayor confianza que utiliza Unity para el desarrollo de componentes en el Editor. Es un lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. La sintaxis y estructuración de este lenguaje es muy similar a la de C++, ya que la intención de Microsoft era facilitar la migración de códigos escritos en este lenguaje y su aprendizaje por

parte de los programadores habituados a C++. Además, el lenguaje de programación C# combina las mejores características de lenguajes preexistentes como Visual Basic, Java y C++ (Technologies, 2017).

- **JavaScript:** Es un lenguaje personalizado inspirado en la sintaxis ECMAScript. Es un lenguaje ligero e interpretado, orientado a objetos con funciones de primera clase, más conocido como el lenguaje de script para páginas web, pero también usado en muchos entornos sin navegador, tales como node.js o Apache CouchDB. Es un lenguaje script multi-paradigma, basado en prototipos, dinámico, soporta estilos de programación funcional, orientado a objetos e imperativo (Technologies, 2017).

Los lenguajes de programación mencionados anteriormente brindan al programador potencialidades similares al trabajar con Unity, estos permiten exportar aplicaciones a las plataformas seleccionadas. Para desarrollar la extensión se seleccionó el lenguaje de programación C#, ya que se cuenta con experiencia desarrollando en este lenguaje y se posee un mayor dominio de este que con JavaScript.

Conclusiones parciales

Debido a que el proyecto no se considera de gran envergadura, el equipo de desarrollo es pequeño y no se requiere la generación de gran cantidad de artefactos; se decidió seleccionar una metodología ágil y no una metodología tradicional, en este caso la metodología XP pues brinda mejor flujo de trabajo para estudios de tesis. La extensión se desarrollará sobre la plataforma de videojuegos *Unity* y de esta manera una vez terminado podrá ser integrado al mismo sin presentar problemas de compatibilidad. Para desarrollar la aplicación se seleccionó el lenguaje de programación *C#*, ya que el equipo de desarrollo cuenta con experiencia desarrollando en este lenguaje y tiene un mayor dominio de este que del resto.

2. Propuesta de Solución

Introducción

El presente capítulo tiene como objetivo hacer una descripción de las principales características de la *extensión* a desarrollar. Se hará mención de las primeras fases de la metodología de desarrollo a utilizar para la implementación de la solución que se propone y se expondrán los artefactos generados durante el transcurso de estas.

2.1. Propuesta de Solución.

Para dar cumplimiento al objetivo de la investigación se propone la *extensión Enemy Behavior.* Esta *extensión* estará compuesto por tres elementos fundamentales:

- Crear Enemigo.
- Conformar Grupo.
- Lista de *NPC*.

Estos elementos permitirán al desarrollador crear una serie de enemigos *NPC* con sus características, habilidades y comportamiento de IA específicos, crear grupos con estos personajes de manera que se comporten como un equipo y se coordinen como tal. Hechos estos grupos, podrán ser incorporados a cualquier tipo de videojuego de acción y aventura para combates en tiempo real.

La opción Crear Enemigo estará activada por defecto, en esta existirán tres tipos de enemigos, Ataque Cuerpo a Cuerpo, Distancia y Magia, estos enemigos poseerán atributos en común, pero además, otros que los distingan entre sí. Una vez que se establezcan los valores para cada enemigo, este se creará y se le añadirá la malla del personaje modelado en dependencia del tipo que sea.

Figura 2.1 Prototipo de creación e NPC

Una vez creado el *NPC*, en la opción Conformar Grupo, se podrán conformar grupos de 3 a 5 enemigos, en dependencia del equipo que se desee crear. Se les asignará un comportamiento Activo o Variado. Donde el comportamiento Activo consistirá en que todos los personajes que compongan el grupo creado, serán asignados como jefes. Esto traerá consigo que cuando el jugador se encuentre con cualquier *NPC*, este podrá llamar al resto de sus compañeros hasta la posición del jugador y comenzarán a atacarlo. Por otra parte, el comportamiento Variado, consistirá en que el desarrollador podrá seleccionar a la hora de crear un grupo, los personajes que desee como jefes. Esto traerá consigo que cuando el jugador se encuentre con los *NPC* jefes, estos llamarán al resto de sus compañeros hasta la posición de él y comenzarán a atacarlo. En caso de que se encuentre con un *NPC* que no es asignado como jefe, este solo lo perseguirá y lo atacará mientras se encuentre en su rango de acción.

Figura 2.2 Prototipo de conformar grupo.

Mientras tanto, en el componente Lista de *NPC*, se podrán listar todos los *NPC* que se hayan creado, ya sea, a la hora de ejecución de la *extensión* como los ya creados en otras ocasiones. Permitirá además eliminar los personajes que se consideren innecesarios.

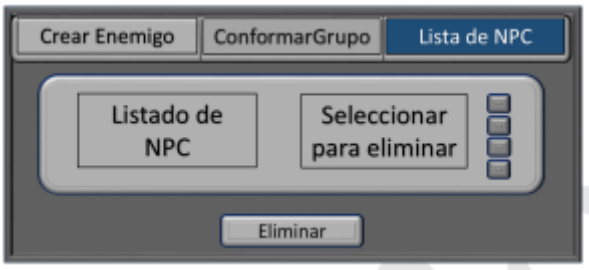


Figura 2.3. Prototipo de Listar NPC

2.1.1 Requisitos funcionales de la extensión.

La captura de requisitos es la disciplina mediante la cual se recopila la información y se transforma en un conjunto de requerimientos que son los que darán límite al alcance del sistema. En el marco inicial del presente trabajo esto fue posible gracias al análisis de algunos videojuegos de impacto internacional como fueron Kimdom of Amalur y la saga Dark Sider, además del intercambio con algunos expertos en diseño y desarrollo de videojuegos, quienes aportaron una importante guía en la formalización de esta base de conocimientos.

Para que la extensión de la plataforma Unity se desarrolle con las condiciones y calidad necesarias se ha requerido que se cumplan los siguientes requisitos:

- Crear enemigo según el tipo de ataque.
- Crear grupo de 3 a 5 enemigos. El grupo debe poseer la característica de avistamiento pasivo o activo.
- Listar cantidad de NPC disponibles.
- Crear comportamiento de inteligencia artificial para los NPCs.

Teniendo ya estos requisitos como los pilares sobre la cual se va a desarrollar la extensión se puede continuar con las siguientes fases de la metodología para guiar todo el proceso de desarrollo.

2.2. Fase de Exploración.

En esta fase los clientes plantean a grandes rasgos las Historias de Usuario (HU) que son de interés para la primera entrega del producto. Asimismo, el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Además, se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología (Letelier, 2006).

2.2.1 Historias de Usuario

Las HU son utilizadas como herramienta para dar a conocer los requerimientos del sistema al equipo de desarrollo. Son pequeños textos en los que el cliente describe una actividad que realizará el sistema. Estos se redactan utilizando la terminología del cliente, de forma que sea clara y sencilla, sin profundizar en detalles. Se puede considerar que las HU en XP juegan un papel similar a los casos de uso en otras metodologías, pero en realidad son muy diferentes. Las HU solo muestran la silueta de una tarea a realizarse. Por esta razón es fundamental que el usuario o un representante suyo se encuentre disponible en todo momento para dar respuesta a las dudas que puedan surgir, ya que estas no proporcionan información detallada acerca de una actividad específica. Las HU también son utilizadas para estimar el tiempo que el equipo de desarrollo tomará para realizar las entregas. En una entrega se puede desarrollar una o varias HU, esto depende del tiempo que demore la implementación de cada una de ellas (Echeverry Tobón, 2007).

Principales aspectos de una HU:

- **Número:** Número asignado a la HU
- **Nombre de HU:** Atributo que contiene el nombre de la HU.
- **Fecha:** Fecha en la cual fue redactada la HU.
- **Usuario:** El usuario del sistema que utiliza o protagoniza la HU.
- **Prioridad en el negocio:** Contiene el nivel de prioridad de la HU en el negocio. Es Alta en caso de que la HU sea indispensable en el negocio, Media en caso de que su realización no afecte el negocio y Baja cuando no se considera una prioridad.

- **Riesgo de desarrollo:** Contiene el nivel de riesgo en caso de no realizarse la HU. Es Alta, si el riesgo de no realizar la HU incide en el funcionamiento de la plataforma, Media si el riesgo de no realizarla es medianamente importante, y Baja en caso de que no se considere un riesgo tardar en la realización de la HU y no incida en el funcionamiento de la plataforma.
- **Puntos estimados:** Este atributo es una estimación hecha por el equipo de desarrollo sobre el tiempo de duración de la HU. Cuando el valor es 1 equivale a una semana ideal de trabajo, y un día equivale a 0.2 puntos.
- **Iteración asignada**: Especifica la iteración a la que pertenece la HU correspondiente.
- **Descripción:** Posee una breve descripción de lo que realizará la HU.
- **Observaciones:** Aquellos detalles relevantes que serán resueltos tras la conversación del equipo desarrollador con el cliente.

Tabla 2.1. Historia de usuario # 1

Historia de Usuario	
Número: 1	**Nombre Historia:** Crear enemigo.
Usuario: Especialista	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3.0	**Iteración asignada:** 1
Programador responsable: Dainel García García	
Descripción: La extensión será capaz de conformar un NPC con los parámetros que defina el usuario, creando un objeto de tipo *prefab* de este enemigo.	

Observaciones:

Tabla 2.2. Historia de usuario # 2

Historia de Usuario	
Número: 2	**Nombre Historia:** Crear grupo.
Usuario: Especialista	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 3.0	**Iteración asignada:** 2
Programador responsable: Dainel García García	
Descripción: Con los personajes creados, la extensión podrá conformar grupos de 3 hasta 5 integrantes. A cada grupo se le podrá definir el comportamiento que tendrá, mediante el cual, sus miembros reaccionarán de una forma u otra ante el jugador.	
Observaciones: Para conformar un grupo, se pueden elegir personajes creados en el momento de ejecución de la extensión o personajes creados en otras ocasiones, estos se encontrarán guardados en la carpeta que los contendrá.	

Tabla 2.3. Historia de usuario # 3

Historia de Usuario	
Número: 3	**Nombre Historia:** Listar NPC.
Usuario: Especialista	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 1.5	**Iteración asignada:** 2
Programador responsable: Dainel García García	
Descripción: La extensión deberá mostrar en una lista todos los personajes que se hayan creado y brindará la opción de eliminar aquellos que el usuario desee.	
Observaciones:	

Tabla 2.4. Historia de usuario # 4

Historia de Usuario	
Número: 4	**Nombre Historia:** Comportamiento IA
Usuario: Especialista	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta

Puntos estimados: 3.0	**Iteración asignada:** 3
Programador responsable: Dainel García García	
Descripción: Se implementa la IA que estará presente en los personajes que se crearán con la herramienta desarrollada. Esta controlará el tipo de NPC que haya definido el usuario, podrá seleccionar si es jefe o no, en dependencia de esto podrá cumplir un papel específico. Permitirá que el personaje mientras no haya encontrado al jugador, pueda estar caminando por el mapa, siendo este su estado inicial, si es jefe, al encontrarse con el jugador llamará a sus compañeros para atacar, si no lo es, entonces solo este perseguirá y atacará al jugador; en caso de que el jugador se aleje mucho, entonces el NPC volverá a su estado inicial. En caso de que cada NPC esté atacando al jugador, no permanecerán estáticos, sino que se moverán alrededor de este.	
Observaciones:	

2.3. Fase de planificación

En esta fase el cliente establece la prioridad de cada historia de usuario y se acuerda el alcance del producto. Los programadores estiman cuánto esfuerzo requiere cada historia y a partir de esto se define el cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. El equipo de desarrollo mantiene un registro de la velocidad de desarrollo, totalizando el número de historias de usuario realizadas en una iteración. Esta medida ayuda a determinar la cantidad de historias que se pueden implementar en las siguientes iteraciones, aunque no de manera exacta. La revisión continua de esta métrica en el transcurso del proyecto se hace necesaria, ya que las historias varían según su grado de dificultad, haciendo inestable la velocidad de la realización del sistema (Anaya Villegas, 2007).

2.3.1. Plan de Estimación.

El cliente es quien establece la prioridad de cada HU y los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Las estimaciones de esfuerzo asociadas a la implementación de las historias son establecidas por los programadores utilizando como medida el punto, lo que equivale a una semana ideal de programación.

Las historias generalmente valen de 1 a 3 puntos. La planificación se puede realizar basándose en el tiempo o el alcance. Esto es utilizado para estimar el tiempo que el equipo de desarrollo necesitará para realizar las entregas. En una entrega puede desarrollarse una o varias historias de usuario, dependiendo del tiempo que demore la implementación de cada una de ellas (Escribano, 2002)

A continuación se muestra mediante una tabla la planificación de las diferentes HU para cada iteración teniendo en cuenta su prioridad.

Tabla 2.5. Estimación de esfuerzo por historia de usuario

Iteración	**Historias de Usuario**		**Duración en semanas**
1	1	Crear enemigo	3.0
2	2	Crear grupo	3.0
	3	Listar NPC	1.5
3	4	Comportamiento IA	3.0
Total			**10.5**

2.3.2 Plan de Iteraciones

Como parte del ciclo de vida de un proyecto guiado por la metodología de desarrollo XP, se crea el plan de duración de las iteraciones que se llevarán a cabo durante el desarrollo del proyecto. Este plan tiene como objetivo fundamental mostrar la duración de cada una de las iteraciones en las que está dividida la fase de desarrollo del proyecto, así como el orden en que serán implementadas las HU en cada iteración según la prioridad asignada por el cliente. El plan de entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se establece una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración. Una vez definidas las HU y estimado el esfuerzo propuesto para la realización de cada una de ellas, se decide por parte del equipo de desarrollo realizar el sistema en 3 iteraciones, las cuales se describen de manera detallada a continuación:

Iteración1: Esta iteración tiene como propósito dar cumplimiento a la HU 1. Esta compone la base del desarrollo de la extensión para la plataforma Unity. Una vez concluida la implementación de esta HU, el sistema podrá crear los personajes NPC. De esta iteración pueden derivarse nuevos requerimientos funcionales. Por último se realizan las pruebas pertinentes a la funcionalidad implementada. Finalizada esto se efectúa la primera entrega de la extensión, para mostrar al cliente lo realizado y recibir sus valoraciones en relación con esta entrega.

Iteración2: El objetivo de esta iteración es dar cumplimiento a las HU 2 y 3. Al finalizar la implementación de dichas HU el sistema deberá ser capaz de conformar los grupos y listar todos los NPC creados. Luego se pasará a solucionar los errores y no conformidades que fueron detectadas en esta iteración. Una vez concluida esta iteración, la extensión contará con sus principales funcionalidades implementadas y se realizará la segunda entrega a los clientes.

Iteración3: En esta iteración se implementa la HU4. Una vez finalizada la implementación de esta HU, el sistema tendrá incorporada una clase encargada de controlar la IA correspondiente a cada NPC. Luego de probar las funcionalidades implementadas y corregir los errores detectados en esta iteración, se constará con una versión estable del producto lista para entregar la primera versión completa al cliente.

2.3.3 Plan de Duración de las Iteraciones

Como parte del ciclo de vida de un proyecto guiado por la metodología de desarrollo XP, se crea el plan de duración de las iteraciones que se realizarán durante el desarrollo del proyecto. Este plan tiene como objetivo fundamental mostrar la duración de cada una de las iteraciones en las que está dividida la fase de desarrollo del proyecto, así como el orden en que serán implementadas las HU en cada iteración según la prioridad asignada por el cliente.

Tabla 2.6 Plan de duración de las iteraciones.

Iteración	**Historias de Usuario**		**Duración en semanas**	
1	1	Crear enemigo	3.0	
2	2	Crear grupo	3.0	4.5
	3	Listar NPC	1.5	
3	4	Comportamiento IA	3.0	
Total			**10.5**	

2.3.4 Plan de entrega

Como resultado de la fase de planificación se genera el plan de entrega que plantea una fecha para la entrega de cada iteración, tomando dos días como holgura para completar la misma.

Tabla 2.7. Plan de entregas

Iteración	Entrega	Fecha
1	Versión 0.3	20 de marzo del 2018
2	Versión 0.8	20 de abril del 2018
3	Versión 1.0	20 de mayo del 2018

2.3.5. Tareas de Ingeniería.

Las historias de usuario son descompuestas en tareas de ingeniería y asignadas a los programadores para ser implementadas en cada iteración (Canós, 2003).

Tabla 2.8. Tarea de ingeniería # 1

Tarea	
Número de tarea: 1	**Número de historia de usuario:** 1
Nombre de la tarea: Implementar un componente encargado de visualizar la forma en que se recogerán los datos que conformaran cada personaje que se desee crear.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.5
Fecha de inicio: 1 de marzo de 2018	**Fecha de fin:** 10 de marzo de 2018
Programador responsable: Dainel García García	
Descripción: Se implementa un componente que conforma la interfaz con la que interactúa el desarrollador a la hora de crear un personaje.	

Tabla 2.9. Tarea de ingeniería # 2

Tarea	
Número de tarea: 2	**Número de historia de usuario:** 1
Nombre de la tarea: Implementar una clase que sea la encargada de procesar la información de la entrada de datos por el desarrollador y crear el personaje correspondiente.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.5
Fecha de inicio: 11 de marzo de 2018	**Fecha de fin:** 20 de marzo de 2018

Programador responsable: Dainel García García
Descripción: Se encarga de procesar la información y crear el objeto personaje con los datos entrados por parámetros.

Tabla 2.10. Tarea de ingeniería # 3

Tarea	
Número de tarea: 3	**Número de historia de usuario:** 2
Nombre de la tarea: Implementar un componente que permita crear grupos con los personajes creados anteriormente.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.0
Fecha de inicio: 25 de marzo de 2018	**Fecha de fin:** 31 de marzo de 2018
Programador responsable: Dainel García García	
Descripción: Se implementa un componente que conforma la interfaz con la que interactúa el desarrollador a la hora de crear grupos de personajes.	

Tabla 2.11. Tarea de ingeniería # 4

Tarea	
Número de tarea: 4	**Número de historia de usuario:** 2
Nombre de la tarea: Implementar una clase que reciba los datos procesados cuando se crean los personajes y visualice cada uno de estos permitiendo crear los grupos y asignarles un comportamiento	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.5
Fecha de inicio: 3 de abril de 2018	**Fecha de fin:** 12 de abril de 2018
Programador responsable: Dainel García García	
Descripción: Se encarga de visualizar los personajes creados y de crear grupos con estos asignándoles su comportamiento.	

Tabla 2.12. Tarea de ingeniería # 5

Tarea	
Número de tarea: 5	**Número de historia de usuario:** 3
Nombre de la tarea: Implementar un componente que permita la visualización de los *NPC* creados.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.0
Fecha de inicio: 13 de abril de 2018	**Fecha de fin:** 19 de abril de 2018
Programador responsable: Dainel García García	
Descripción: Se implementa un componente que permita la visualización de todos los *NPC* creados por el desarrollador.	

Tabla 2.13. Tarea de ingeniería # 6

Tarea	
Número de tarea: 6	**Número de historia de usuario:** 3
Nombre de la tarea: Implementar una clase que reciba la información de cada grupo creado y se encargue de mostrarlos.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.0
Fecha de inicio: 21de abril de 2018	**Fecha de fin:** 30 de abril de 2018
Programador responsable: Dainel García García	
Descripción: Se encarga de mostrar en una lista cada *NPC* creado con su información referente.	

Tabla 2.14. Tarea de ingeniería # 7

Tarea	
Número de tarea: 7	**Número de historia de usuario:** 4
Nombre de la tarea: Implementar una clase encargada de crear la IA referente a los *NPC*.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3.0
Fecha de inicio: 1de mayo de 2018	**Fecha de fin:** 26 de mayo de 2018

Programador responsable: Dainel García García
Descripción: Esta clase es la encargada de crear la IA que controlará los *NPC* creados, dependiendo del tipo definido, estos tendrán un comportamiento específico.

2.4. Fase de Diseño.

En *XP* se considera que no es posible tener un diseño completo y sin errores del sistema desde el principio, dada la naturaleza cambiante del proyecto. Por lo tanto, hacer un diseño muy extenso en las fases iniciales del proyecto para luego modificarlo, se considera un malgasto de tiempo. Es por ello, que esta tarea es permanente durante la vida del proyecto, partiendo de un diseño inicial el cual va siendo corregido y mejorado durante el proceso de desarrollo (Echeverry Tobón, 2007).

2.4.1. Arquitectura.

La arquitectura de *software* es la estructura de un sistema, la cual comprende los componentes por los que está formado, las propiedades de estos componentes visibles externamente, y las relaciones entre ellos. En el contexto del diseño arquitectónico, un componente de *software* puede ser tan simple como una *extensión*, pero también puede ser algo tan complicado como incluir bases de datos que permita la configuración de una red de clientes y servidores. Las propiedades de los componentes son aquellas características necesarias para entender cómo los componentes interactúan con otros componentes, dejando de lado las propiedades internas específicas de cada uno. Las relaciones entre los componentes pueden ser tan sencillas como una llamada de procedimiento de una *extensión* a otro, o tan complicadas como el protocolo de acceso a bases de datos.

La arquitectura constituye un modelo relativamente pequeño y comprensible de cómo está estructurado el sistema y de cómo trabajan sus componentes en conjunto. Esta brinda la posibilidad de analizar la efectividad del diseño para darle cumplimiento a los requisitos fijados. De esta forma, el diseño arquitectónico y los patrones arquitectónicos pueden ser aplicados en el diseño de otros sistemas y representados a través de un conjunto de abstracciones que facilitan la descripción de la arquitectura de un modo predecible (Pressman, 2010).

Patrón Arquitectónico por Capas.

En una arquitectura basada en capas el sistema se organiza jerárquicamente, de manera tal que cada capa brinde servicios a la capa superior y se sirva de las prestaciones que le brinda la inferior. En esta arquitectura los componentes de la capa externa se encargan de la interacción con el usuario mediante una interfaz. En la capa interna, los componentes realizan operaciones exclusivas al ámbito del *software*, como la obtención de datos, la interacción con el sistema operativo o el control de un dispositivo. Mientras, las capas intermedias proporcionan mecanismos necesarios para que la capa externa muestre los datos obtenidos por la capa inferior, controlando la lógica del sistema.

Esta arquitectura permite la creación de un diseño basado en varios niveles de abstracción, en el cual un problema complejo puede dividirse en partes más pequeñas para darle solución. A cada nivel se le confía una misión simple, lo que permite el diseño de una arquitectura escalable, la cual puede ampliarse con facilidad en caso que las necesidades aumenten. En una arquitectura basada en capas, cada nivel puede ser modificado de forma transparente. Conociendo la interfaz de la capa inferior y manteniendo la utilizada por la capa exterior, puede ser modificado el funcionamiento de la capa intermedia con el fin de optimizar un algoritmo o modificar el tratamiento de los datos. A continuación se muestra la arquitectura por capas del componente para *Unity* a desarrollar.

Figura 2.4. Arquitectura de la *extensión*.

Capa de Presentación.

Esta es la capa con la que interactúa el usuario. En esta se encuentran las clases que permiten la interacción con la *extensión*, así como las estructuras a través de las cuales son devueltos los datos capturados.

Capa lógica.

En esta capa se controla el proceso de recolección de datos entrados por el usuario para la generación personajes. Además se encuentra en esta la clase encargada de conformar y controlar la IA de los *NPC* creados.

Capa de soporte.

En esta capa se encuentran las clases que provee el motor de videojuegos *Unity*, las cuales son utilizadas para almacenar la información que provee la *extensión*.

2.4.2. Elementos del Diseño.

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Son reusables ya que pueden ser aplicados en otros diseños o problemas (Gamma, 1995).

Para el diseño del componente de software para la creación de IA de *NPC* en comportamientos grupales, se tuvieron en cuenta los siguientes patrones *General Responsibility Assignment Software Patterns* (*GRASP*): Experto, Bajo acoplamiento, Alta cohesión, Controlador y Creador.

Se hizo uso del patrón Experto, esto se pone de manifiesto en la clase EnemyBehavior ya que esta clase es la encargada de todo lo referente al componente. Esta contiene toda la información necesaria para el trabajo con el mismo.

Se evidencia el uso del patrón Bajo Acoplamiento y Alta Cohesión, debido a que las clases fueron diseñadas de tal forma que existiera la menor relación entre ellas, a la vez logrando que la información almacenada en cada clase fuera coherente. Con este diseño se logra que de existir una modificación tenga la mínima repercusión posible en el resto de las clases, se garantiza la reutilización de código y disminuye la dependencia entre las mismas.

Se utiliza el patrón Controlador, su uso se evidencia en la clase EnemyBehavior la cual es la encargada de la capa lógica del negocio, aumentando así la reutilización de código y un mayor control. Esta clase es la encargada de controlar todo el proceso de negocio asignando las responsabilidades a las clases correspondientes.

Se utiliza el patrón Creador, su uso se evidencia en la clase EnemyBehavior la cual es la encargada de crear y almacenar instancias de otras clases, ya que ella posee la información necesaria para este proceso, a la vez de controlar todo el proceso de negocio.

2.4.3. Tarjetas Clase - Responsabilidad - Colaboración.

En la metodología *XP* no es necesaria la descripción del sistema a través de diagramas de clase utilizando notación *Unified Modeling Language* (*UML*), sino que se guía por técnicas como las tarjetas Clase - Responsabilidad - Colaboración (CRC). Las tarjetas CRC son fichas en las que se escriben brevemente las responsabilidades de la clase y una lista de los objetos con los que colabora para llevar a cabo estas responsabilidades. (Pressman, 2010)

Tabla 2.15. Tarjeta CRC # 1

Tarjeta CRC	
Clase: Grupo*NPC*	
Responsabilidad	**Colaboración**
Crea la interfaz principal de la *extensión* a desarrollar, con las funcionalidades que permitan al usuario crear los *NPC* así como los grupos de enemigos.	• Comportamiento_IA • PrefabUtility

Tabla 2.16. Tarjeta CRC # 2

Tarjeta CRC	
Clase: Comportamiento_IA	
Responsabilidad	**Colaboración**

Crear el comportamiento de IA que controlará los *NPC* con los que se trabajará.	• NavMeshAgen

Conclusiones parciales

Con esta propuesta de solución, se espera cumplir el objetivo del presente trabajo. Pues se analizaron las primeras fases de la metodología de desarrollo de software utilizada. Durante el transcurso de estas se documentaron los artefactos generados por ellas, los cuales facilitaron la creación del diseño de la *extensión*, mostrando los elementos más importantes a tener en cuenta durante su desarrollo.

3. Implementación y prueba

En este capítulo se analiza la propuesta de solución desde el punto de vista de la calidad, para evaluar cómo se le da cumplimiento al problema de la presente investigación y verificar si fueron implementadas de forma correcta cada una de las HU. Se procede a diseñar un conjunto de pruebas correspondientes a la metodología XP, las cuales se dividen en tres partes: carga y estrés, unitarias y de aceptación. A través de métricas se evalúa el diseño propuesto, en aras de comprobar su correcta elaboración. Por otro lado, se describe de forma detallada todo el proceso de pruebas y se muestra un resultado final, que explica y muestra al usuario que el sistema está implementado correctamente y que cuenta con las funcionalidades descritas en el capítulo anterior.

3.1. Fase de Implementación.

En esta fase se realiza la implementación de las HU correspondientes a cada iteración, se chequea el plan de iteraciones por si es necesario realizar modificaciones y se crean las tareas de programación para implementar exitosamente cada HU.

Se realizó el diseño e implementación de la extensión, el cual está almacenado en una carpeta principal con el nombre Assets, la misma contiene la herramienta desarrollada, distribuida en varias subcarpetas. La clase encargada de crear y controlar la interfaz de la extensión se encuentra en la subcarpeta Editor, mientras que los script que controlan el comportamiento de IA, los eventos y animaciones, además de un ejemplo de personaje para el escenario demo, se encuentran distribuidos en subcarpetas específicas dentro de la subcarpeta Script. En esta misma subcarpeta se encuentra el script encargado de crear la base de datos que contendrá los personajes creados por el módulo. Los materiales con los que estarán compuestos los personajes, ya sean, texturas, modelos o animaciones, están almacenados en subcarpetas específicas de cada uno en la subcarpeta Art. A su vez, los prefab de cada personaje creado mediante la herramienta, así como los prefab de los grupos y de los ataques, están en subcarpetas específicas dentro de la subcarpeta Prefabs. El escenario demo en el cual se mostrarán los personajes o grupos creados con su comportamiento de IA y un ejemplo de personaje, se encuentra en la subcarpeta Scene.

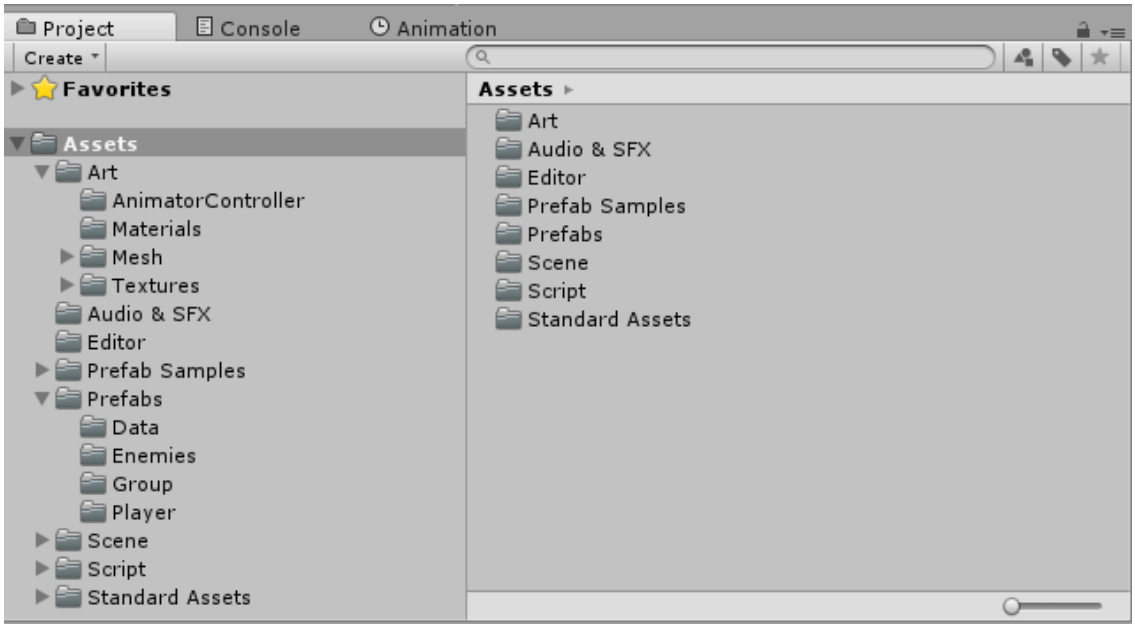


Figura 3.1. Distribución de la *extensión*.

Al quedar estructurado y distribuida la extensión en las carpetas que lo conformarán, se procede a la implementación de las clases. La interfaz con la que se encontrará el desarrollador está compuesta por tres funcionalidades, Crear Enemigo (en la extensión Enemy Creator), Conformar Grupo (en la extensión Group Creator) y Lista de *NPC*(en la extensión Enemy List). Estas se implementaron en el *script EnemyBehavior.cs*. Esta clase contiene los métodos referentes a cada funcionalidad de la *extensión*, así como los que permitirán la visualización del mismo y las variables que recogerán los datos que el usuario entre por parámetro. Una vez finalizada la implementación de esta interfaz, se obtiene la herramienta funcional.

Cuando el desarrollador despliegue la herramienta se encontrará con una ventana que permitirá seleccionar la funcionalidad con la que desee trabajar. En la funcionalidad *Enemy Creator*, se carga la malla 3D que conformará el personaje, se selecciona el tipo de enemigo, se le adjunta el componente *AnimatorController* para las animaciones del personaje y se establece un nombre para este. Hecho esto, se procede a configurar los demás atributos de ataque, tales como, puntos de vida, puntos de ataque, distancia de ataque, distancia de avistamiento y cargar el prefab que contendrá el mismo. Establecidos todos los parámetros se puede crear entonces el personaje presionando el botón “Create Enemy”. Finalizado este proceso, el *extensión* crea el objeto *prefab* del *NPC* en la capeta *Prefabs*. Cada personaje es guardado en una base de datos local, la cual se utilizará para mostrarlos o eliminarlos en la funcionalidad *Enemy List*.

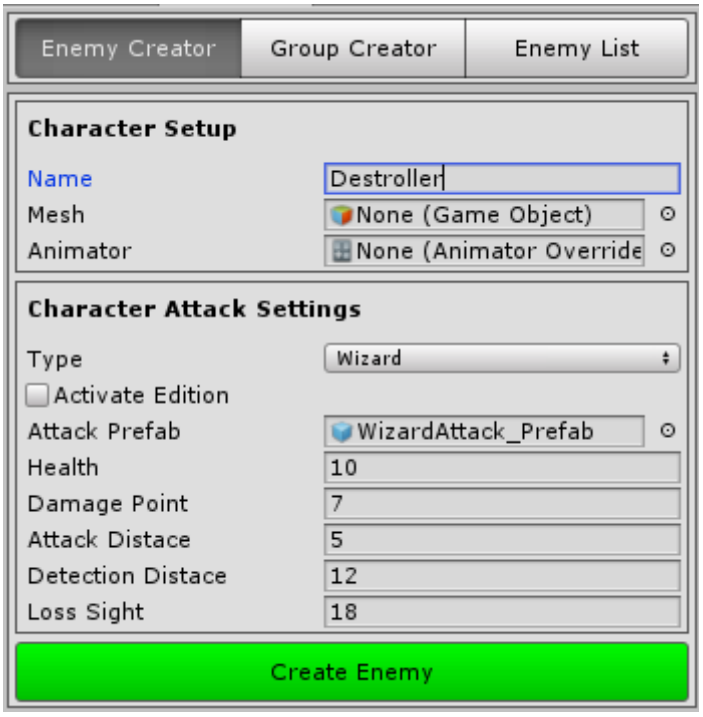


Figura 3.2. Funcionalidad Crear Enemigo

En caso que el desarrollador seleccione la pestaña de la funcionalidad Conformar Grupo, se asigna un nombre para cada grupo a crear y se procede a configurar el mismo. Para esto se implementó un componente que permite seleccionar la cantidad de personajes que conformarán un grupo. Este puede ser de 3 a 5 miembros, una vez seleccionada esta opción se despliega una lista de campos en los que se pueden cargar los objetos generados por la función Crear Enemigo. Una vez cargados los personajes, se asigna un comportamiento al grupo, siendo este Activo o Variado. Para esto se implementó un componente que permite seleccionar qué comportamiento tendrá el grupo a crear; además de mostrar una información referente a cada uno. Si se selecciona el comportamiento Activo, todos los integrantes del grupo serán jefes; mientras que al seleccionar el comportamiento Variado, solo el primer NPC que conforme el grupo será el jefe. Establecidos estos parámetros se puede crear el grupo presionando el botón “Create Group”. Finalizado este proceso, la extensión crea el objeto prefab del grupo en la carpeta Prefabs y limpia los parámetros de la ventana posibilitando crear nuevos grupos.

Figura 3.3. Funcionalidad Conformar Grupo

Mientras que en la pestaña de la funcionalidad Lista de *NPC*, se carga una base de datos que contiene un listado de todos los personajes creados, permitiendo eliminar este si se desea.

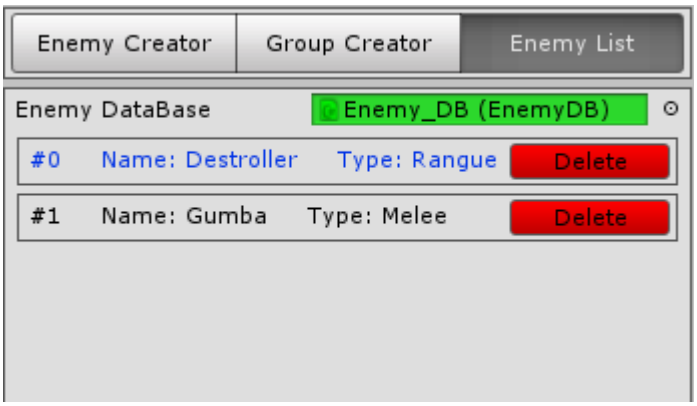


Figura 3.4. Funcionalidad Listar Enemigos

Concluida la implementación de la interfaz de la *extensión* se procedió a la implementación de las clases que controlan la IA de los *NPC* y sus animaciones así como el daño que pueden ocasionarle al jugador. La clase y el prefab encargados de controlar el *player* se tomó de los *assets* que Unity provee por defecto, en este caso el prefab *FPSController,* que simula un personaje en primera persona. A continuación se muestra un diagrama de los estados principales del script *AIBehavior.cs* encargado de controlar los personajes enemigos en el demo.

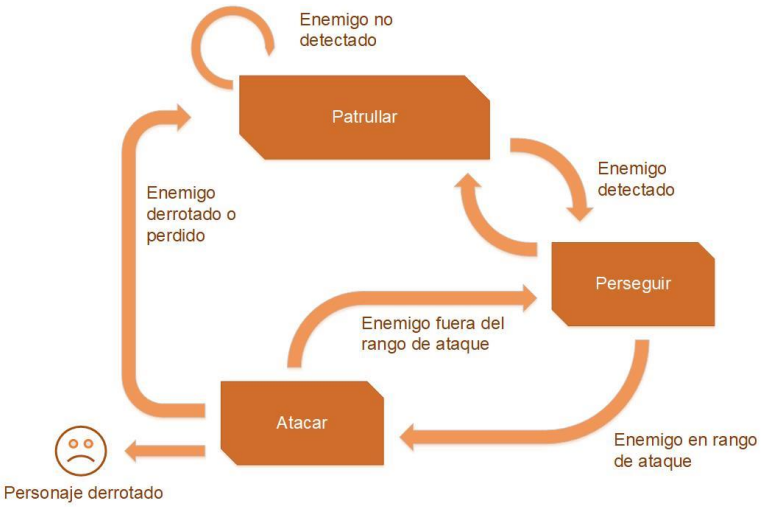


Figura 3.5. Diagrama del funcionamiento del script AIBehavior.cs

Finalizada la implementación de la clase encargada de la IA de los personajes, se conforma un escenario demo en el cual se procede a comprobar el funcionamiento de la extensión al crear los personajes y los grupos, vinculados con la IA asociada a estos. Esto se puede ver en la siguiente imagen que refleja la interfaz de la *extensión* y un escenario donde hay algunos *NPC* atacando al *player*.

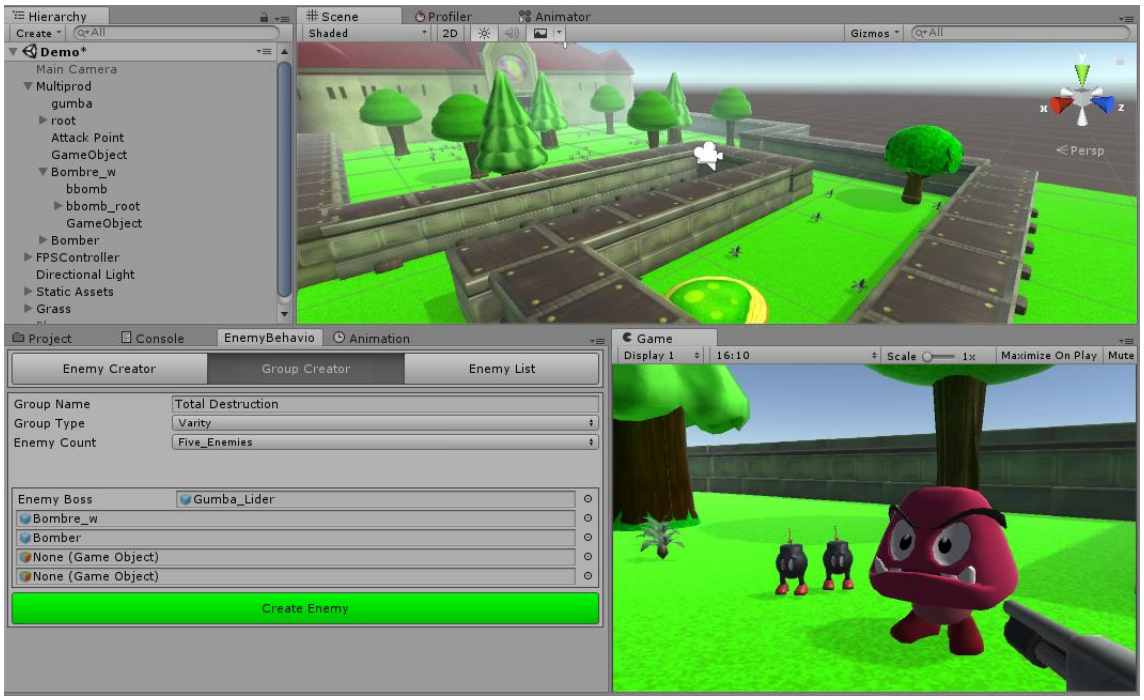


Figura 3.6. Escenario demo

De esta manera se procede a realizar las pruebas al sistema implementado y corregir cualquier error detectado, así como ajustar la IA de los *NPC*.

3.2. Pruebas del sistema.

Las pruebas del sistema se realizan con el objetivo de comprobar la calidad requerida y de detectar tantos errores como sea posible en la aplicación una vez que se finaliza la fase de implementación. Estas son realizadas por los desarrolladores en colaboración con el cliente, buscando corregir cualquier no conformidad que pudiera surgir (Echeverry Tobón, 2007).

3.2.1 Pruebas unitarias

Las pruebas unitarias son las pruebas diseñadas por los programadores y están enfocadas al código, consisten en verificar de manera manual o automatizada, si una parte específica del código, funciona de acuerdo con los requisitos del sistema. Deben ser definidas antes de realizar el código y repetirse hasta eliminar todos los errores para aumentar la calidad del desarrollo (Reglas y Prácticas en eXtreme Programming, 2008).

Debido a que en la solución se emplea un gran número de procedimientos encargados de mostrar contenido visual y debido también a que son pocas las funciones que se usan, además de presentar una baja complejidad, se decide realizar las pruebas unitarias de manera manual para adaptarlas a esta situación y verificar si este contenido visual se muestra correctamente.

Se realizaron en cada una de las iteraciones definidas, para evitar el arrastre de errores lógicos a iteraciones posteriores. En la siguiente figura se muestra un ejemplo de caso de prueba unitaria realizada al método CreateEnemy () de la clase EnemyBehavior.cs, el cual en un primer momento arrojó un error. El dar clic en el botón de crear enemigo en objeto que se crea como resultado del método aparece vacío. Dicho error fue solucionado en un segundo momento.

```
if (GUILayout.Button("Create Enemy", GUILayout.Height(30)))
{
    string newPath = "Assets/Prefabs/Enemies/" + enemyName + ".prefab"; //direccion dentro del proyecto
    var prefab = PrefabUtility.CreateEmptyPrefab(newPath); //crear un prefabricado vacio
    GameObject prefabInstance = PrefabUtility.InstantiatePrefab(enemyMesh) as GameObject;  // crear la maya seleccionada


    GameObject n = PrefabUtility.ReplacePrefab(prefabInstance, prefab); // Adicionar la maya seleccionada a ese prefabricado

    //adicionar componentes a la maya seleccionada
    n.AddComponent<NavMeshAgent>();
    n.AddComponent<AIBehavior>();
```

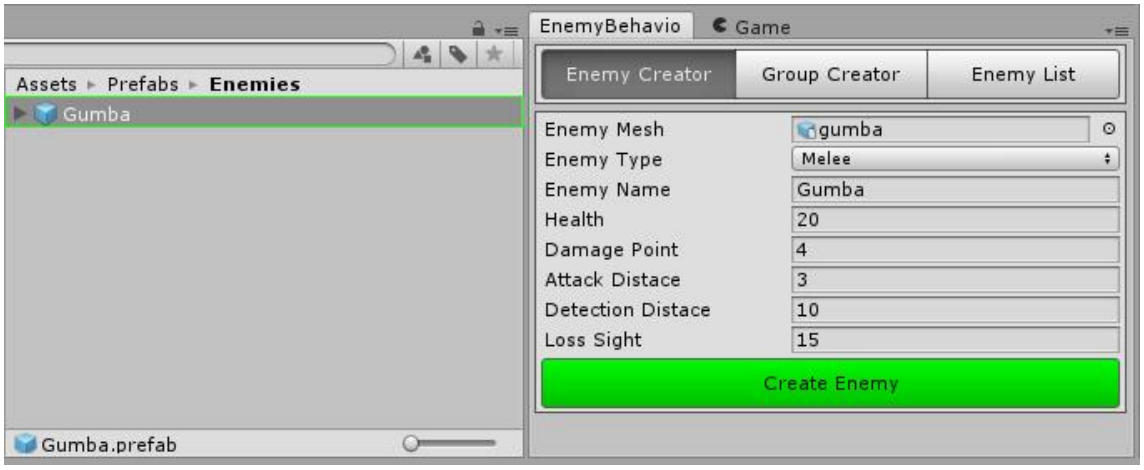


Figura 3.1. Ejemplo de caso de prueba unitaria realizada a la solución.

3.2.2 Pruebas de aceptación.

A continuación se muestran las pruebas de aceptación realizadas a las principales funcionalidades del sistema. Para estas pruebas son escogidas todas las funcionalidades del sistema. Debido a que la prioridad de cada una de estas historias de usuario es alta es necesario comprobar que todas sean aceptadas por el cliente y de no ser así pasar a corregirlas inmediatamente.

Tabla 3.1. Prueba de aceptación # 1

Caso de Prueba de Aceptación	
Código: HU1_P1	**Número de HU:** 1
Nombre: Crear Enemigo.	
Descripción: Es la primera opción con la que se encuentra el desarrollador al interactuar con la extensión. Permite crear el personaje que desee dependiendo del tipo que se seleccione.	

Condiciones de ejecución: No presenta ninguna condición para su ejecución.
Entrada/Pasos de ejecución: 1. Clic en el botón "*Enemy Behavior*." de la barra de tareas. 2. Seleccionar "Editor". 3. Sale en pantalla la ventana principal de la extensión con la funcionalidad "Enemy Creator" seleccionada por defecto. 4. Seleccionar la malla 3D del personaje. 5. Seleccionar el tipo de personaje a crear. 6. Llenar los demás datos que conformaran al personaje. 7. Clic en el botón "Create Enemy".
Resultado esperado: Se crea un objeto prefab, este representa el personaje que se acaba de crear
Evaluación de la prueba: Resultado satisfactorio.

Tabla 3.2. Prueba de aceptación # 2

Caso de Prueba de Aceptación	
Código: HU2_P2	**Número de HU:** 2
Nombre: Crear Grupo.	
Descripción: Se crean grupos de personajes y se asigna un comportamiento a cada grupo.	
Condiciones de ejecución: Los personajes deben haber sido creados anteriormente.	
Entrada/Pasos de ejecución: 1. Si no se ha cerrado la ventana de la extensión, dar clic en la funcionalidad "Create Group". De lo contrario, seguir los dos primeros pasos de ejecución de la prueba anterior.	

2. Agregar los personajes creados, posibilitando seleccionar entre 3 y 5 miembros que se deseen para conformar cada grupo. 3. Se selecciona el tipo de comportamiento que tendrá el grupo. 4. Clic en el botón “Create Group”.
Resultado esperado: Visualiza los personajes creados. Se crea el grupo con los enemigos seleccionados y el comportamiento deseado.
Evaluación de la prueba: Resultado satisfactorio.

Tabla 3.3. Prueba de aceptación # 3

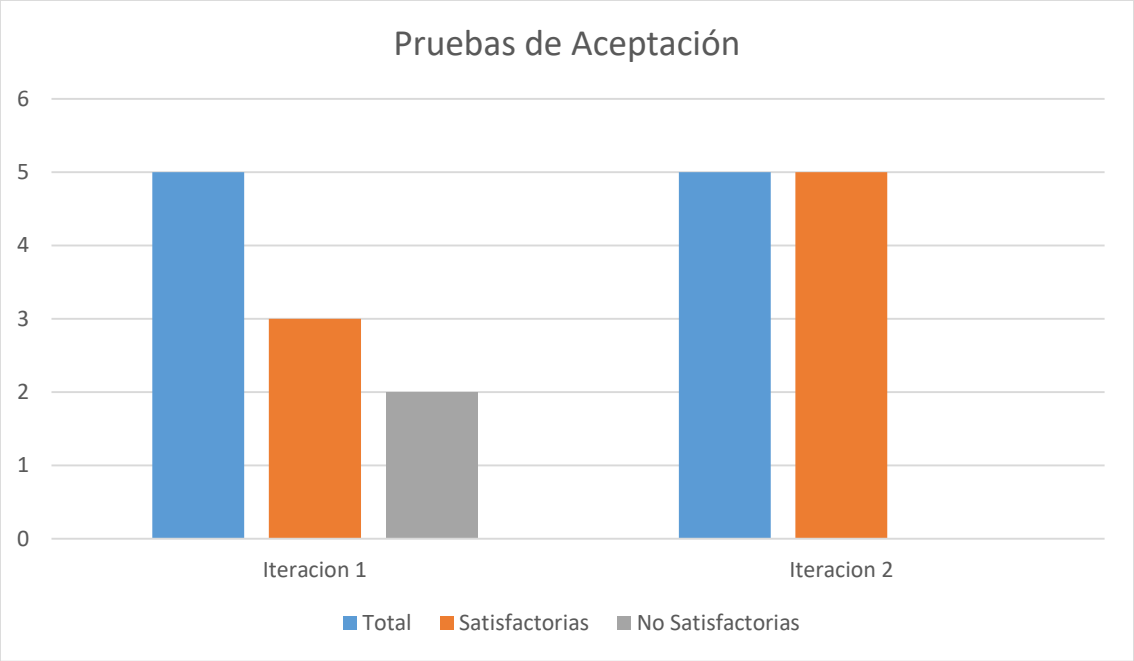
Caso de Prueba de Aceptación	
Código: HU3_P3	**Número de HU:** 3
Nombre: Listar NPC.	
Descripción: Lista todos los personajes creados, permitiendo eliminar los que el desarrollador desee.	
Condiciones de ejecución: Los personajes deben haber sido creados anteriormente.	
Entrada/Pasos de ejecución: 1. . Si no se ha cerrado la ventana de la extensión, dar clic en la funcionalidad “Listar NPC”. De lo contrario, seguir los dos primeros pasos de ejecución de la primera prueba. 2. Visualiza todos los personajes creados, posibilitando seleccionar tantos como se desee para eliminarlos. 3. Clic en el botón “Eliminar”.	
Resultado esperado: Se eliminan los personajes seleccionados	
Evaluación de la prueba: Resultado satisfactorio.	

Tabla 3.4. Prueba de aceptación # 4

Caso de Prueba de Aceptación	
Código: HU4_P4	**Número de HU:** 4
Nombre: AIBehavior.	
Descripción: La IA asociada que controla cada personaje	
Condiciones de ejecución: Los personajes deben haber sido creados anteriormente.	
Entrada/Pasos de ejecución: 1. El personaje se encuentra caminando aleatoriamente por el terreno. 2. El personaje encuentra al jugador. 3. Persigue al jugador. 4. Ataca al jugador. 5. Si en su rango de ataque se encuentra un compañero, modifica su posición para atacar al jugador. 6. El jugador se encuentra con el jefe del grupo. 7. Se lanza una alerta a los demás personajes para que vengan hasta él. 8. Llegan y comienzan a atacar. 9. El jugador huye y todos lo persiguen y lo atacan.	
Resultado esperado: El comportamiento de IA hace que el personaje persiga y ataque al jugador.	
Evaluación de la prueba: Resultado satisfactorio.	

Al realizar las pruebas de aceptación en una primera iteración de un total de 5 pruebas, 3 fueron satisfactorias y 2 no satisfactorias, estas fueron las pruebas realizadas a la historia de usuario AIBehavior. En estas cuando en el NPC detectaba el personaje lo perseguía pero no atacaba, además de en el caso que fuese el jefe no realizaba la llamada a sus compañeros correctamente por lo que fue necesaria una segunda iteración de pruebas. En esta se corrigieron los errores de la anterior iteración y de un

total de 5 pruebas, 5 fueron satisfactorias y ninguna no satisfactoria, por lo que no fueron necesarias más iteraciones. De esta manera, se lograron integrar de manera exitosa los objetos creados por la extensión con el comportamiento de IA.



Gráfica 3.1 Resultados de las pruebas de aceptación

3.3 Análisis de Resultados.

La solución propuesta se implementó de manera que cualquier desarrollador de videojuegos pudiera fácilmente crear personajes, agruparlos y asignarles un comportamiento de IA, de esta manera, cuando el usuario se encuentra frente a la aplicación tendrá tres opciones a realizar Crear Enemigo, Conformar Grupo, Listar NPC.

Para el análisis se va tomar una muestra de tres desarrolladores de videojuegos para realizar una comparación teniendo en cuenta el tiempo que demoran estos para integrar el comportamiento de IA a los enemigos y crear los grupos de forma manual contra la utilización de la extensión. De esta manera, se puede ver si la utilización de la extensión puede agilizar o no el proceso de desarrollo.

3.3.1 Medición del Tiempo de Desarrollo.

Se tomó una muestra de 3 desarrolladores de videojuegos del equipo Green Rune, participantes y ganadores de varias ediciones del evento internacional Global Game Jam en su sede en Cuba. Se realizaron dos pruebas con cada uno de los

desarrolladores para determinar el tiempo que se demoraba cada uno en realizar una composición de un NPC y un grupo de forma manual y la otra utilizando la extensión para de esta forma comparar los tiempos de composición y determinar si la extensión desarrollada es capaz o no de agilizar el proceso de creación de NPC y Grupos de NPC para ser utilizados en videojuegos que posean combates en tiempo real. A continuación se muestran los resultados de la prueba.

Tabla 3.5 Análisis de composición con el desarrollador # 1

Análisis de composición con el desarrollador 1	
Nombre del desarrollador	Roberto E. Pérez Ozete
Experiencia	Diseñador y desarrollador de videojuegos del equipo Green Rune. Director y desarrollador del videojuego “Superclaria” para plataforma android.
Tiempo de composición manual	30 minutos
Tiempo de composición con la extensión	5 minutos
Observaciones del autor: Inicialmente se tomó un minuto para analizar la extensión pero unos minutos más tarde ya había creado un grupo de enemigos satisfactoriamente.	
Resultado: La realización de un grupo funcional de NPCs se realizó con más rapidez que de forma manual.	

Tabla 3.6 Análisis de composición con el desarrollador # 2

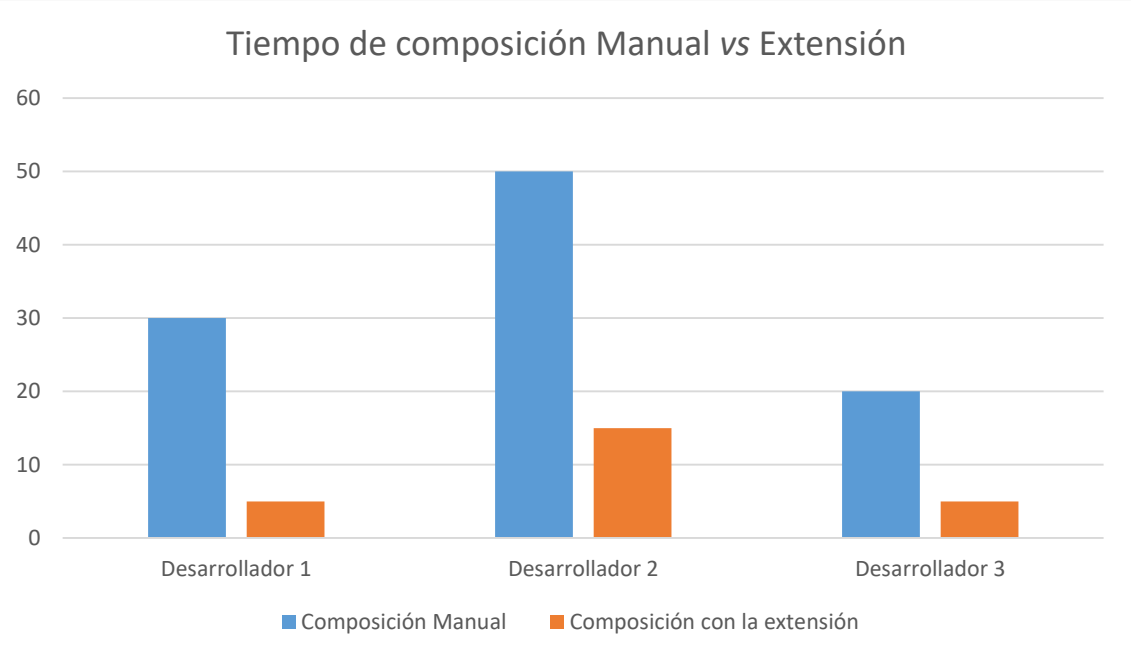
Análisis de composición con el desarrollador 2	
Nombre del desarrollador	Carlos Alberto Gavillas Cruz

Experiencia	Artista conceptual y desarrollador adiestrado del equipo Green Rune.
Tiempo de composición manual	50 minutos
Tiempo de composición con la extensión	15 minutos
Observaciones del autor: Estuvo mucho tiempo componiendo y probando los enemigos de forma manual hasta lograr componer el grupo. Luego al hacer uso de la extensión se demoró unos minutos para dominarla pero al cabo de unos minutos comenzó a crear y probar los enemigos y crear un grupo funcional.	
Resultado: La realización de un grupo funcional de NPCs se realizó con más rapidez que de forma manual. Además el desarrollador asintió de que de esta forma se optimiza tiempo y trabajo	

Tabla 3.7 Análisis de composición con el desarrollador # 3

Análisis de composición con el desarrollador 3	
Nombre del desarrollador	Liuver Espinosa Cabrera
Experiencia	Desarrollador líder de videojuegos del equipo Green Rune. Desarrollador del videojuego “Crysp”, el cual fue ganador en el Global Game Jam 2018.
Tiempo de composición manual	20 minutos
Tiempo de composición con la extensión	5 minutos
Observaciones del autor: Se detuvo un instante a analizar la extensión pero unos minutos más tarde ya había creado un grupo de enemigos satisfactoriamente.	

Resultado: La realización de un grupo funcional de NPCs se realizó con más rapidez que de forma manual, además el desarrollador felicitó al autor por la interfaz amigable de la extensión ya que permite al usuario familiarizarse con ella rápidamente.



3.3.2. Validación del Sistema.

La validación es el proceso mediante el cual se comprueba que el sistema desarrollado cuenta con las especificaciones del cliente y cumpla con las necesidades del usuario. Con las pruebas realizadas se confirmó que la extensión creada puede ser utilizada por desarrolladores de videojuegos a través la herramienta Unity. El mismo permite crear personajes con comportamiento de IA y realizar grupos con estos. De esta manera se agiliza el proceso de desarrollo de videojuegos, al incorporar estos grupos inteligentes en cualquier tipo de videojuego de aventura y acción para combates en tiempo real en fase de producción.

Conclusiones parciales

En este capítulo se realizó la descripción de la solución y el análisis del resultado de las pruebas realizadas a la extensión para Unity. Del análisis del resultado obtenido se llegó a la conclusión de que se cumplió con el objetivo de la investigación al lograr correctamente el desarrollo, tanto de la herramienta como de la inteligencia artificial que controlará cada tipo de personaje que se cree. Además se logró la aceptación por parte del cliente de la herramienta realizada.

Conclusiones Generales

Con la realización de este trabajo se arribó a las siguientes conclusiones:

- Se implementó una extensión para *Unity* que permite crear personajes enemigos y agruparlos sobre una base de comportamiento inteligente con el propósito de comportarse como un equipo.
- Con la extensión implementada, los desarrolladores de videojuegos podrán crear enemigos e incluirlos en grupos con comportamiento inteligente para ser insertados en videojuegos de aventura y acción en escenas de combate en tiempo real.
- Se desarrolló un escenario demo con varios grupos de enemigos y un ejemplo de jugador, fabricados a partir de la extensión desarrollada para validar su funcionamiento. Los personajes creados se comportan como un grupo, ya sea Activo o Variado, realizando las funciones implementadas correctamente.

Recomendaciones

La extensión creada es una herramienta que se incorpora a *Unity*, con esta los desarrolladores pueden crear videojuegos capaces de estar a la altura de las tendencias de IA grupales a nivel mundial. De esta manera, se recomienda que en futuros trabajos se desarrolle y perfeccione el mismo, con el propósito de agregar nuevas funcionalidades y comportamientos que puedan ser incorporados a los videojuegos que se creen.

Bibliografía

Anaya Villegas, Adrian. 2007. *A propósito de programación extrema XP.* 2007.

Artificial Intelligence in Game Design. **Schreiner, Tim. 2009.** 2009.

Calero, Manuel. 2003. *Una explicación de la programación extrema (XP).* Madrid : s.n., 2003.

Canós, José H, Patricio Letelier y Ma Carmen Penadés. 2003. *Metodologías Ágiles en el desarrollo de software.* 2003.

Creative Commons. 2018. Creative Commons. [En línea] 2018. http://www.gamerdic.es/tema/generos.

Echeverry Tobón, Luís Miguel y Luz Elena Delgado Carmona. 2007. *Caso práctico de la metodología ágil XP al desarrollo de software.* 2007.

El Otro Lado. 2017. [En línea] 2017. http://www.elotrolado.net/wiki/G%C3%83%C2%A9neros_de_los_videojuegos.

Enemy design and enemy AI for melee combat systems. **Vossen, Bart. 2015.** s.l. : Gamasutra's Community, 2015.

Escribano, Gerardo Fernández. 2002. *Introducción a Extreme Programming.* 2002.

Facultad de Diseño y Comunicación. 2014. Universidad de Palermo. [En línea] Clasificación Géneros de Videojuegos., 2014. http://fido.palermo.edu/servicios_dyc/blog/.

Fernández, Gerardo Escribano. 2002. *Introducción a Extreme Programming.* 2002.

Gamma, Erich. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software. .* s.l. : Addison Wesley, 1995.

Homer Reynoso, Ever A. e Iriam A. González Boffill. 2009. *Sistema de Toma de Decisiones Para Videojuegos.* 2009.

Inteligencia Artificial en Videojuegos. **Trujillo Rivero, Andy y Marvyn A. Márquez Rodríguez. 2008.** La Habana : s.n., 2008.

Jacobson, Ivar, Grady Booch y James Rumbaugh. 1999. *The Unified Software Development Process.* 1999.

La evolución de la inteligencia artificial en los videojuegos. **Rodriguez, Guillermo. 2016.** s.l. : Vix, 2016.

Letelier, Patricio. 2006. *Metodologías ágiles para el desarrollo de software: Extreme Programming (XP).* 2006.

Marrero, Adrian Guerra. 2010. *Introducción a la IA en los videojuegos.* 2010. Razon Artificia.

Penadés, Carmen. 2003. *Metodologías Ágiles en el desarrollo de Software.* 2003.

Pérez Ozete, Roberto Elías. 2016. *Arquitectura de un Videojuego.* 2016.

Pressman, Roger S. 2010. *Sotware Engineering. A Practitioner's Approach.* 2010.

Reglas y Prácticas en eXtreme Programming. **JOSKOWICZ, J. 2008.** España : Manitoba, 2008.

Technologies, Unity. 2017. Unity . *Unity* . [En línea] Unity Technologies, 2017. http://unity3d.com/es/unity .