UNIVERSIDAD DE ORIENTE
NÚCLEO DE SUCRE
ESCUELA DE CIENCIAS
DEPARTAMENTO DE INFORMÁTICA

DESARROLLO DE UN VIDEOJUEGO INDIE, 3D, DEL GÉNERO ROLE PLAYER
(Modalidad: trabajo de grado)

JUAN DAVID GALINDO GONZÁLEZ

TRABAJO DE GRADO PRESENTADO COMO REQUISITO PARCIAL PARA
OPTAR AL TÍTULO DE LICENCIADO EN INFORMÁTICA

CUMANÁ, 2016

DESARROLLO DE UN VIDEOJUEGO INDIE, 3D, DEL GÉNERO ROLE PLAYER

APROBADO POR:

_____

Prof. Carmen Victoria Romero

Asesora

_____

Prof. Ana Fuentes

Coasesora

_____

(Jurado)

_____

(Jurado)

CUMANÁ, 2016

ÍNDICE

Pág.

DEDICATORIA

Cuando pienso en este trabajo, concluyo en que no solo ha sido fruto de la investigación, conocimiento y las habilidades desarrolladas por mí durante dos años, sino también, de una serie de correctas decisiones que he tomado en mi vida, no creo que sea un conjunto de hechos afortunados, más bien, pienso en que Dios me ha otorgado el raciocinio y la sabiduría, en conjunto con la educación que me infundieron mis padres, que me han permitido discernir entre lo favorable y lo perjudicial, por eso, a ustedes, que me han hecho ver las respuestas en donde los ojos no pueden, va dedicado este logro, que es solo un punto de partida en un largo camino de metas y aspiraciones, pero como diría Illidan, ahora estoy preparado.

AGRADECIMIENTOS

A mis padres, Yonny y Otilia, por haberme mostrado desde siempre los valores y buenas costumbres que me forjaron como persona durante mis tempranos años de vida, a ellos les debo lo que he logrado ser al día de hoy.

A la profesora Carmen Victoria Romero y a la profesora Ana Teresa Fuentes, por haberme apoyado siempre en la ejecución de la ardua tarea que resulta ser llevar adelante un proyecto de este tipo.

A mis amigos y compañeros, con los que he compartido momentos inolvidables a lo largo de estos años, Claudia, Robert, Alexander, Abraham, Pedro, Marizel y a todos con los que compartí durante mi paso por la carrera.

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

RESUMEN

Se desarrolló un videojuego indie del género *role player* en el motor de juegos Unity, usando herramientas como Blender, Photoshop, iClone, para la creación de gráficos en 3D y el lenguaje de programación C# para la programación de las funcionalidades del mismo. Se tomó como referencia la metodología SUM (Acerenza, Copes, Mesa y Viera., 2009) para videojuegos, una modificación de Scrum para acoplarse mejor a este propósito. Mediante un ciclo inicial de planificación se diagramaron las tareas y actividades necesarias para la elaboración del juego, logrando en un siguiente ciclo iterativo diversas versiones del producto con funcionalidades nuevas y/o más refinadas. Se consiguió un producto beta luego de cuatro fases de testeo en usuarios, en donde, de realizaron mejoras y ediciones en base al *feedback* de los *testers* en cada una de las iteraciones.

Palabras claves: Videojuego, 3D, Unity, Role Player.

INTRODUCCIÓN

Se puede definir un videojuego o juego electrónico como un software que interactúa con una o varias personas a través de un dispositivo o plataforma, con la finalidad de brindar entretenimiento y en algunos casos conocimiento a la clase de usuario a la que está destinado a llegar. Según Geymonat (2014) el término "videojuego" comenzó a utilizarse en la década del 70. Marquès (2001) quien sostiene que, un videojuego es: "todo tipo de juego digital interactivo, con independencia de su soporte (ROM interno, cartucho, disco magnético u óptico, on-line) y plataforma tecnológica (máquina de bolsillo, videoconsola conectable al TV, teléfono móvil, máquina recreativa, microordenador, ordenador de mano, video interactivo)". Citado por Geymonat (2014). En todos es importante la acción del jugador e integran diversas notaciones simbólicas (textos, sonidos, fotografías, videos, música, imágenes en tres dimensiones); son dinámicos y altamente interactivos.

Muchos establecen el inicio de los videojuegos gracias a la construcción de las primeras supercomputadoras tales como el ENIAC, ya que después de aquí empezaron los intentos de implementar programas de carácter lúdico como el ajedrez en estas plataformas. Pero según Donovan (2010), los primeros videojuegos no aparecieron hasta la década de los 60, y a partir de aquí se puede decir que llega su nacimiento y empieza su evolución de la mano de los avances científicos y de la creatividad de los programadores y diseñadores a cargo.

En el año 1972, es creada la primera consola de videojuegos llamada Magnavox Odyssei y en 1977, la famosa Atari 2600 pionera en crear los cartuchos intercambiables que fueron sustituidos años después por el DVD. Ya con la llegada de los *smartphones*, se incrementó exponencialmente la cantidad de productos y jugadores, convirtiéndose en una de las plataformas más populares de videojuegos junto a las consolas.

A pesar de su exponencial crecimiento, los videojuegos siempre tuvieron sus detractores, un ejemplo palpable fue el ámbito académico y educativo en años anteriores. Frasca (2010), indica que, el juego como disciplina ha sido un paria en el mundo de la teoría y la academia. Hasta finales del siglo XX, poca atención se le prestó al juego salvo en disciplinas muy particulares (sociología, antropología, psicología) y sin una visión integradora, por suerte, el siglo XXI inaugura con una visión más amplia e interdisciplinaria de esta actividad, en gran medida debido a la irrupción económica de los videojuegos. Así mismo afirma que, "…los videojuegos se perfilan como la industria cultural dominante en este nuevo siglo, y las universidades rápidamente, en respuesta, se lanzan a explotar este terreno poco conocido…".

Espinoza (2009) explica que "En un videojuego RPG (del inglés *role-playing game*) un jugador desempeña un rol a través de un personaje dentro de un mundo virtual a lo largo de una historia, la cual sigue un curso de acuerdo a sus decisiones y acciones; el personaje puede ganar experiencia, ítems y otras características a lo largo del juego, de acuerdo a las tareas que desempeña en este mundo virtual". Este estilo de juego permite al jugador adentrarse en una matriz de hechos no lineales, a diferencia con otro tipo de juegos en los que transcurre una secuencia de hechos siempre similar o idéntica.

La industria de los videojuegos se equipara a la del cine en cuanto a presupuesto, personas involucradas y ganancias generadas, por lo tanto un juego moderno requiere la dedicación y soporte exclusivo de una gran compañía; sin embargo, en los últimos años gracias a las herramientas de desarrollo disponibles para los usuarios surgen los llamados *Indie* o independiente, que son juegos más pequeños realizados por una o varias personas sin el financiamiento o sin las herramientas que poseen las compañías especializadas y a pesar de ello algunos han logrado más éxito que otros juegos comerciales, ejemplo de ello tenemos títulos como *Terraria* (2011), *I wanna be the guy* (2007), *Day Z* (2011), entre otros.

Hoy en día el término *Indie*, se refiere en general a cualquiera de los movimientos, panoramas, subculturas, atributos estilísticos y culturales, con un acercamiento

autónomo y un nivel de planteamiento que se reduce al lema “Hazlo tú mismo”, Sánchez (2011).

En el estudio realizado por Benito en 2005, se encontró que casi la mitad de la población es usuaria de videojuegos, predominando esta práctica más entre los varones que entre las mujeres, aunque se aprecia un mayor consumo por las mujeres entre las generaciones más jóvenes en juegos de PC, y entre los adolescentes sobre los más maduros. En la Unión Europea uno de cada cuatro (4) individuos juega regularmente. Además de esto, el consumo de videojuegos supone el 35% del total del consumo de ocio audiovisual, situándose por delante de la taquilla de cine, la música grabada y las películas de vídeo.

Debido a la cantidad de jugadores y el tiempo que dedican a ello, comúnmente se han clasificado en dos (2) ramas los tipos de jugadores: casuales y *gamers*. Los casuales refiriéndose más a un público general que utiliza los videojuegos eventualmente, casi siempre desde las plataformas móviles, mientras que los *gamers* se les llama a los jugadores más asiduos.

Las actividades en que los seres humanos invierten su tiempo de ocio cambian constantemente con las épocas, los medios audiovisuales como la televisión, el internet, cine, música y videojuegos; tienen una importante presencia en este conjunto de actividades en la actualidad. Los juegos electrónicos lideran en cuanto a su adquisición para ocupar el tiempo libre. Etxeberria (2008) en su artículo Videojuegos, consumo y educación, asegura, que el éxito creciente de los videojuegos es una realidad en la sociedad actual, sustentando su aseveración en la investigación de aDeSeen (2008), quien demuestra que este tipo de juegos constituyen el 54% del consumo del ocio audiovisual e interactivo en España durante el año 2007 por delante del cine (23%), películas de video (13%) y música grabada (10%).

Por lo visto previamente, se infiere que esta actividad ha influido de alguna manera en la cotidianidad del ser humano. Todo esto ha generado que las personas dedicadas a la creación y distribución de estos productos, se involucren

en este negocio tomando como premisa los requerimientos del jugador o *gamer* sin dejar de lado el objetivo propio de esta industria.

De Aguilera (2004) determinó que, por lo general, la producción de un videojuego requiere la dedicación, durante un periodo de uno o dos años, de un extenso equipo compuesto por diversos profesionales (programadores, artistas, músicos, diseñadores, guionistas, productores y otros) que, siguiendo las pautas que marque el director de proyecto, unen sus diversas perspectivas profesionales para la consecución de un mismo fin: la confección de un videojuego, una obra cultural, susceptible de lograr el éxito comercial.

No solo los diseñadores y desarrolladores cobran un salario gracias a los videojuegos, sino a jugadores más comprometidos y ya catalogados como profesionales, quienes reciben beneficios económicos ya sea por participar en competencias oficiales, o por tener patrocinio de compañías de renombre.

La competencia "*The international*" del videojuego *Dota 2* desarrollado por la empresa *Valve Corporation,* es uno de los ejemplos más destacados de dichos eventos profesionales, se realiza todos los años y participan 16 equipos de todo el mundo. En el 2015, el evento se realizó en el complejo deportivo *Key Arena* de la ciudad de Seattle, y como se comprueba, en el reportaje de Thursten (2015) para la revista *PC Gamer*, entre los equipos que ocuparon los primeros lugares se entregaron más de 18 millones de dólares, siendo esta una cifra record hasta la fecha para competencias de videojuegos.

En varios países del mundo ya se cuenta con escuelas de jugadores, que proporcionan el título de jugadores profesionales tales como China, Corea del sur, Francia, Alemania y Japón. Mientras, en Estados Unidos y España se cuenta con ligas universitarias de videojuegos, una de las más conocida es la del juego *League of Legends* en las universidades de Estados Unidos, donde los jugadores reciben entrenamiento dedicado y becas especiales.

Así como la de los jugadores, los desarrolladores también se han profesionalizado, hoy en día ya se cuenta con carreras universitarias completas para formar a estas

personas tanto en niveles de pregrado como postgrado, así como también cátedras dedicadas al tema en carreras afines, tales como informática, sistemas y computación.

Configurándose la propuesta del desarrollo de un videojuego del género RPG, que posee todas las características deseadas de un juego de este estilo, como lo son la diversidad de los hechos, narración no lineal y libertad del jugador de tomar decisiones dentro de ciertos parámetros establecidos, una opción de trabajo de grado que permite dar inicio a esta área dentro de la carrera de Licenciatura en Informática de la Universidad de Oriente (UDO).

CAPÍTULO I. PRESENTACIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

Los videojuegos viven un auge en los últimos años, sin embargo, en Latinoamérica este campo es aún incipiente comparado con Norteamérica, Europa y Asia. La escuela Da Vinci en Argentina ofrece una carrera de 3 años de duración y otorga el título de “Diseñador y Programador de Simuladores Virtuales”, la Universidad de Talca en Chile también ofrece un plan de estudios similar y otorga a sus egresados el título de “Ingeniero en desarrollo de videojuegos y realidad virtual”

El campo de los juegos electrónicos se ha posicionado de buena manera en el ámbito académico como objeto de estudio e innovación, una cantidad considerable de universidades alrededor del mundo dan oportunidades de estudio y promueven la investigación sobre los simuladores de realidad virtual. Sin embargo, en Venezuela la promoción de esta área es inapreciable, ya que apenas se encuentran algunos pocos antecedentes de juegos educativos como trabajo de grado, siendo lo más resaltante el programa de especialización en creación y programación de videojuegos, ofrecido por la Universidad Simón Bolívar, que otorga el grado de especialista en creación y programación de videojuegos. A nivel laboral tampoco se cuenta con una industria desarrollada ni compañías que le den una gran promoción.

En la Universidad de Oriente, el *pensum* de la Licenciatura en Informática cuenta con algunas materias que pueden servir como base para comenzar a incursionar en el desarrollo de videojuegos, como es el caso, de la asignatura Simulación y Modelos (230-4174), la cual a pesar de que no ofrece alguna unidad específica sobre videojuegos, da la opción de desarrollar uno como proyecto final, sin embargo, no está establecida un área de investigación que oriente a los estudiantes en este aspecto. Existe una asignatura de carácter no obligatoria con el nombre de Teoría de Juegos (230-4184) (ver anexo), la cual pertenece al área

de análisis e investigación de operaciones (AIO), y se dedica al estudio de los modelos matemáticos que fundamentan esta teoría sin llegar a la implementación computacional de los mismos, siendo importante aclarar que esta asignatura nunca se ha implementado, por lo que en el Departamento de Informática del Núcleo de Sucre no se tiene experiencia sobre la misma.

Para el desarrollo de videojuegos, es necesaria la implementación de métodos y algoritmos estudiados en distintas áreas de investigación de las contempladas en las asignaturas de la Licenciatura en Informática de la UDO, para lo que deben ser ajustados y adaptados para que cumplan la finalidad del videojuego aportando una funcionalidad al mismo. El paradigma de programación orientada a objetos es el estilo de programación utilizado actualmente por la mayoría de los motores de juego, para definir entidades como: personajes, escenario, y todos los objetos que toman forma dentro del ámbito de trabajo, siendo este paradigma de desarrollo abordado por el área de programación y de sistemas, en casi todas sus asignaturas. Por lo que el desarrollador debe poseer la habilidad y destreza para poder dar funcionalidad al videojuego, manejando y reestructurando a su antojo la composición de los objetos con los cuales está tratando, ya que dichos objetos se encuentran organizados en árboles y clases heredadas.

Además de lo expuesto anteriormente, se estila a crear una clase controladora para cada objeto que toma vida dentro del videojuego, la finalidad de esta clase es definir métodos que son llamados cuando el objeto interacciona con algún agente externo a él y realiza los procedimientos correspondientes, uno de estos agentes externos puede ser la entrada por teclado del usuario, para asignar una orden al objeto; dicho objeto debe reaccionar a esta entrada con algún método definido en su clase controladora. Esta clase también puede estar acompañada de una máquina de estado finito para complementar su funcionamiento, el controlador puede hacer modificaciones en los parámetros de la máquina de estado para que esta los evalué y haga el cambio de estado pertinente, generalmente cada estado está asociado a las animaciones que puede realizar un objeto, por lo que es

necesario conocer y manejar los conceptos al modelo de máquinas de estados así como los métodos asociados.

Los controladores no solo pueden estar en capacidad de actuar luego de una orden del usuario, sino también pueden estudiar el ambiente y reaccionar, de modo que pueden ser programados los comportamientos de cualquiera de los objetos en el juego, creando una forma de autómatas virtuales, se puede conseguir así, desde las más simples hasta las más complejas inteligencias artificiales. Para el comportamiento de los personajes es común el uso de sistemas de búsqueda de caminos (*pathfinding*) y algoritmos de toma de decisiones.

Como todos los controladores deben ejecutarse relativamente en tiempos iguales, se debe tener conocimiento acerca de la programación concurrente para lograr una sinergia entre las entidades que interactúan entre sí, para ello se puede hacer uso de las co-rutinas, iteradores, entre otros.

Normalmente los jugadores pueden guardar el estado de sus partidas y retomarlas en el momento que deseen, para tal requerimiento, son usados lenguajes de representación de datos como XML y Json.

Por todo esto, con este proyecto propuso la creación de un videojuego *indie* rpg con un concepto de entretenimiento pero que a su vez buscó fomentar el acercamiento hacia el mundo del desarrollo de videojuegos, y dar a conocer cuáles son las habilidades y conocimientos necesarios, y orientarlos hacia la investigación y el emprendimiento de proyectos destacados tanto dentro de la Universidad de Oriente como en la región y así dar pie al crecimiento de esta área en lo académico.

1.2 ALCANCE Y LIMITACIONES

1.2.1 Alcance

A nivel de investigación, el objetivo que se planteó fue la culminación de cada uno de los productos que fueron generados en cada una de las iteraciones realizadas, como es establecido por la metodología de desarrollo de videojuegos Sum. El

primer producto conseguido fue el plan de proyecto generado en la culminación de la etapa de planificación, el cual se cumplió a cabalidad y no sufrió cambios a través del proceso iterativo, en el transcurso de cada sprint se consiguieron productos pequeños que ya podían ser sometidos a pruebas y con las funcionalidades de gameplay bien definidas, hasta llegar a la fase beta en donde se consiguió un producto final, probado y revisado por los usuarios.

1.2.2 Limitaciones

Para la elaboración del videojuego se contó con un equipo de desarrollo compuesto por una única persona, además de tener que procesar componentes de otras disciplinas como el arte, animación y sonido, lo que implicó una mayor carga de trabajo y de conocimientos necesarios, además hubo limitaciones en cuanto a prestaciones del hardware utilizado.

1.3 OBJETIVOS

1.3.1 GENERAL

Desarrollar un videojuego del género *RPG* y modalidad *Indie,* a partir de un concepto original establecido.

1.3.2 ESPECÍFICOS

Levantar información y herramientas útiles para la creación de un videojuego con las características deseadas.

Conceptualizar el videojuego.

Elaborar el videojuego.

Evaluar un producto beta con un grupo de usuarios seleccionados, para la corrección de errores.

CAPÍTULO II. MARCO DE REFERENCIA

2.1 MARCO TEÓRICO

2.1.1 Antecedentes de la investigación

Existen ya una cantidad considerable de trabajos que proponen la creación de videojuegos y en algunos casos miden al mismo tiempo su impacto en los usuarios, la mayoría intentan explotar el potencial educativo que tienen y el grado de masividad que viene teniendo en los últimos años. Dentro de estos trabajos de investigación se encuentran:

Espinoza (2009), en su trabajo de investigación titulado Desarrollo de juego educativo RPG en teléfonos móviles, implementó un motor de juego propio hecho en Java que detectaba interacciones entre elementos, permitiendo compartir información entre esos objetos. Mas específicamente, cuando el personaje hacía contacto con alguna gráfica del juego, permitía el uso de métodos en las clases asociadas a cada gráfico. A partir de ese motor se construyó el juego RPG en cuestión, el cual tiene como objetivo apoyar los métodos de educación tradicional en la escuela. Utilizó varias herramientas conocidas en el desarrollo web, como el patrón de diseño modelo vista controlador (MVC) y diversas librerías de Java para el manejo de gráficos.

Pérez y Pérez (2014), tuvieron como objetivo en su trabajo de investigación llamado, Un conjunto de herramientas para Unity orientado al desarrollo de videojuegos de acción-aventura y estilo retro con gráficos isométricos 3D, apoyar a los desarrolladores indies y entusiastas de los videojuegos de estilo retro. Su motivación fue la gran cantidad de proyectos indies que fracasan debido a la complejidad de las ideas tan especificas que intentan llevarse acabo y conllevan una gran cantidad de tareas en diseño, arte y programación. Con este conjunto de herramientas, conocidas también como *assets*, se pone a la mano de los usuarios, una serie de *APIs* (*Aplication Programming Interface*) para propulsar la puesta en marcha y ejecución de estos proyectos, evitando la ardua tarea de tener que escribir código desde cero, así como también librerías gráficas para trabajar con

gráficos isométricos, pinturas y secuencias de cinemáticas. Este trabajo servirá de mucha ayuda ya que establece una estructura de datos limpia y reutilizable que funciona como guía a la hora de realizar el modelo de clases e interacciones.

Fuentes (2014), en su trabajo Desarrollo de un videojuego de aventuras en C# sobre Unity, realizó un producto desde cero para el sistema operativo android, el juego fue de genero RPG y se propuso como base para la realización a futuro de otros juegos en la Universidad Politécnica de Valencia. Proporcionando una guía de diseño que sirve como aprendizaje para los usuarios que se propusieran la creación de videojuegos como meta.

Palma y Alcobé (2009), usaron un motor mas especifico como *rpgmaker*, mas enfocado en los videojuegos del género rpg y con una fuerte influencia de estilo asiático. Su objetivo fue netamente de entretenimiento y trataba de sumergir al jugador en una compleja trama histórica que iba avanzando con el progreso dentro del juego. La programación fue hecha en el lenguaje Ruby, con ayuda de la librería RGSS2 (Ruby Game Scripting System). Se inspiraron en una de las franquicias mas conocidas del género, *Final Fantasy*. Debido a que la fortaleza de este trabajo es cumplir con todas las características de un juego *RPG*, este trabajo aportará una guía en cuanto a estilo de “jugabilidad” que debe poseer un juego del género.

2.1.2 Bases teóricas

El desarrollo de un videojuego es una actividad multidisciplinaria que involucra, entre otros, al desarrollo de software y a la creación audiovisual. El proceso de desarrollo consta de sus propias etapas diferentes a las del software tradicional y al tener objetivos difíciles de medir, como la diversión, construirlos constituye un gran desafío (Acerenza, Copes, Mesa y Viera., 2009).

Este trabajo, se enmarca en el área del desarrollo de videojuegos, tomando como herramientas, tecnologías bastante recientes, como los motores de juego y motores gráficos a alto nivel, a continuación, se describen una serie de conceptos relacionados con los mismos.

Un Motor de Videojuegos (en inglés *Game Engine*) es una aplicación de software que ofrece todas las herramientas necesarias para el diseño y desarrollo completo de un videojuego, disponiendo de un motor de renderizado para gráficos 2D y 3D, detector de colisiones, sonidos, *scripting*, animación, inteligencia artificial, redes, *streaming*, administración de memoria y mucho más (Arce., 2011).

Unity Engine fue el motor utilizado para el desarrollo del videojuego, “el motor de juegos Unity es desarrollado por Unity Technologies en Dinamarca, Unity integra un motor de renderizado en tiempo real personalizado, así como el motor de físicas PhysX de la compañía nVidia y Mono, la implementación de código abierto de la librería .NET de Microsoft” (Craighead, Burke y Murphy., 2007).

El renderizado en tiempo real comprende el proceso de crear imágenes rápidamente en el computador, esta es el área más dinámica en la computación gráfica. Una imagen es mostrada en la pantalla, el espectador actúa o reacciona, y esta acción afecta lo que es generado posteriormente. Este ciclo de reacción y renderizado ocurre a una tasa lo suficientemente rápida como para que el espectador no vea imágenes individuales, en lugar de eso, se ve sumergido en un proceso altamente dinámico. (Akenine-Moller, Haines y Hoffman., 2008).

El proceso de renderizado en tiempo real conlleva varias técnicas de procesamiento gráfico, entre ellas, la iluminación global. La iluminación global, es un término usado para describir un rango de técnicas y modelos matemáticos que pretenden simular el complejo comportamiento tanto de las luces, como sus rebotes y la interacción con el mundo. Simular la iluminación global de manera precisa representa un desafío y puede llegar a ser computacionalmente costoso, debido a esto, los videojuegos, utilizan un conjunto de aproximaciones para manejar esto de antemano, en lugar de hacerlo durante el *gameplay*. (Documentación oficial de Unity).

Por defecto, las luces en Unity, direccional, *spot* y punto, son de tiempo real. Esto significa que estas contribuyen directamente en la luz de la escena y se actualizan en cada fotograma. Mientras las luces y los *GameObjects* sean movidos dentro de

la escena, la iluminación será actualizada inmediatamente. Esto se puede observar tanto en la vista del escenario como en la vista del juego. Las luces en tiempo real son la manera más básica de iluminar objetos dentro del entorno y son muy útiles para iluminar personajes u otras geometrías dinámicas. Desafortunadamente, las rayos de luz producidos por el motor de Unity no realizan el proceso de rebote por si solas. Para obtener escenas más realistas usando técnicas como la iluminación global se necesita habilitar la iluminación de pre cómputo de Unity. (Documentación oficial de Unity).

Cuando se realiza un *lightmap*, los efectos de la luz en objetos estáticos en la escena son calculados y el resultado es escrito en texturas que son solapadas encima de la geometría de la escena para crear el efecto de iluminación. Estos llamados *lightmaps* pueden incluir tanto la luz directa que alcanza una superficie, y también la luz indirecta, que es la que rebota de otros objetos o superficies dentro de la escena. Esta textura de iluminación puede ser usada junto con la información de la superficie dentro de la escena, como el color (*albedo*) y relieve (*normal map*), por el *shader* asociado al material de un objeto. (Documentación oficial de Unity).

Los materiales son definiciones acerca de cómo la superficie debería ser renderizada, incluyendo referencias a texturas utilizadas, información del tiling (suelo de baldosas), tines de color y más. Las opciones disponibles para un material depende en qué shader del materia está utilizando. (Documentación oficial de Unity).

Los *shaders*, son pequeños códigos que contienen los cálculos de matemáticas y algoritmos para calcular el color de cada pixel renderizado, basándose en la entrada de iluminación y la configuración del material. (Documentación oficial de Unity).

En los videojuegos podemos encontrar conceptos difíciles de definir, como lo explica Bertín y colaboradores (2011), en términos de lenguaje, es de suma importancia para cualquier disciplina obtener una definición clara y abstracta del tema que le ocupa, es decir, poder llegar a tratar un tema de forma independiente

a la experiencia misma. En este sentido, los teóricos y diseñadores han hecho grandes esfuerzos durante años para poder definir tanto el *gameplay* como otros elementos de su oficio. Uno de los comienzos para muchos de ellos fue una definición básica y libre, entregada por el diseñador Sid Meier “el *gameplay* es una serie de decisiones interesantes”. A partir de esta definición, el concepto de *gameplay* ha empezado a ser construido, y aunque hasta la época contemporánea el término continúa siendo utilizado en contextos dispares y para denotar fenómenos diversos, existe cierto consenso al respecto entre los profesionales del medio. El diseñador Henry Rollins cita a Sid Meier, cambiando y expandiendo un poco la frase inicial: “Una serie de desafíos interconectados en un entorno simulado” En el avance del término, existen tres aspectos importantes que se definen más específicamente y que son prácticos para diferenciar la experiencia del *gameplay* de otras experiencias humanas. En primer lugar, el uso de la palabra desafío en lugar de decisión: Las personas toman muchas decisiones en sus vidas cotidianas, muchas de las cuales pueden ser interesantes, pero que también se encuentran a mucha distancia del ámbito del entretenimiento interactivo.

Durante el diseño, programación y ejecución del *gameplay* toman un importante papel varias características esenciales ofrecidas por los motores de juego, las cuales se definen a continuación.

Los GameObjects son los objetos fundamentals en Unity, estos pueden representar personajes, componentes, artefactos, entre otros. Por si solos no tienen mucho significado pero actúan como contenedores para otros componentes, que son los que implementan las verdaderas funcionalidades (Documentación oficial de Unity).

El componente *transform* determina la posición, rotación y escala de cada objeto en la escena. Cada GameObject posee un *transform*.

Un *rigidbody* es el componente principal que permite el comportamiento físico para un objeto. Con un *rigidbody* adjunto, el objeto inmediatamente responderá a la

gravedad. Si uno o más componentes *collider* son también agregados entonces el objeto será movido por colisiones entrantes (Documentación oficial de Unity).

Los componentes *Collider* definen la forma de un objeto para los propósitos de colisiones físicas. Un collider, el cual es invisible, necesita no estar con la misma forma exacta que la malla del objeto y de hecho, una aproximación a menudo es más eficiente e indistinguible en el juego (Documentación oficial de Unity).

El sistema de scripting puede detectar cuando colisiones suceden e instanciar acciones utilizando la función *OnCollisionEnter*. Sin embargo, usted puede también utilizar el motor de física simplemente para detectar cuando un *collider* entra al espacio de otro sin crear una colisión. Un *collider* configurado como *Trigger* (utilizando la propiedad *Is Trigger*) no se comporta como un objeto solido y simplemente le permitirá a otros colliders pasar a través de él. Cuando un *collider* entra su espacio, un *trigger* va a llamar la función *OnTriggerEnter* en los scripts del *trigger* del objeto (Documentación oficial de Unity).

Monobehaviour, es la clase base de la cual todos los scripts derivan. Usando Javascript todos los script automáticamente se derivan de *Monobehaviour*, cuando se usa C# hay que especificar que la clase deriva de *Monobehaviour*. (Documentación oficial de Unity).

La clase NavMesh es usada para obtener consultas espaciales, como *pathfinding* y pruebas de caminos, se puede estableces el *pathfinding* para áreas específicas, y para manejar comportamientos globales de búsqueda de caminos y evasión (Documentación oficial de Unity).

Pathfinding es el área de la inteligencia artificial que busca encontrar el mejor camino de un punto a otro en mapas representados digitalmente (Cuevas, 2013).

El *Character Controller* es un componente utilizado para controles de jugador de tercera o primera persona que no hace uso de la física del *rigidbody* (Documentación oficial de Unity).

C# es un lenguaje de programación simple, moderno, orientado a objetos y de tipado, que combina la alta productividad del desarrollo ágil con el poder que ofrecen los conocidos lenguajes C y C++. (Hejlsberg, Wiltamuth y Golde., 2003).

En el desarrollo de videojuegos 3D se utilizan una gran cantidad de herramientas y técnicas para el diseño de las figuras gráficas, estas generalmente son externas al motor de juego.

Acerenza, Copes, Mesa y Viera (2009) definen modelado como, la construcción de los modelos de personajes y objetos del videojuego (p.ej. un animal o un arbol). Para los personajes se requiere conocer de anatomía humana o animal para hacerlos creíbles y de animación para que se puedan mover naturalmente. Para los objetos se requiere entrenamiento en diseño industrial o mecánico para conocer el balance, los materiales y la física de cada elemento.

RIG es una cadena de huesos o controladores que permiten crear deformaciones o poses sobre un objeto 3D y *rigging* se conoce como el proceso de implementar *rigs* en una malla (Andagoya., 2016).

La cinemática inversa consiste en hallar el vector de ángulos de las articulaciones a partir de la orientación y posición del efector final, el cual es un problema de difícil solución debido a que incluye ecuaciones no lineales y múltiples soluciones (Giraldo, Delgado y Castellanos., 2006).

Los paquetes de Unity son una manera sencilla de compartir y reutilizar los proyectos de Unity, son conocidos como *assets*. Unity ofrece por defecto una serie de paquetes prediseñados para la iniciación de un proyecto.

Estos paquetes son colecciones de archivos y datos de proyectos de Unity, o elementos de estos, lo cuales están comprimidos y almacenados en un archivo, similar a los archivos de formato Zip. Un paquete mantiene su estructura de directorios original cuando es desempaquetado, así como también sus metadatos (Documentación oficial de Unity).

2.2 MARCO METODOLÓGICO

2.2.1 Metodología del área aplicada

Para la elaboración del videojuego se usó como proceso de desarrollo de software el método *SUM*, la cual es una metodología ágil concebida especialmente para la creación de videojuegos. Según sus autores, Acerenza, Coppes, Mesa, Viera, Fernández, Laurenzo y Vallespir (2009) “La metodología *SUM* para videojuegos tiene como objetivo desarrollar videojuegos de calidad en tiempo y costo, así como la mejora continua del proceso para incrementar su eficacia y eficiencia”.

SUM toma como base dos metodologías que han registrado éxitos en la industria de los videojuegos, *Scrum* y *XP*, que también son métodos ágiles que garantizan en gran medida el éxito de este tipo de proyectos, por su flexibilidad y adaptabilidad. *SUM* adapta los procesos de dichas metodologías a equipos de desarrollo multidisciplinarios y de reducido número de integrantes, por ende, se acopla más a la realidad de los equipos de trabajo característicos de un juego *Indie*.

Roles

Se toma una estructura de roles muy parecida a la de *Scrum*, definiendo cuatro roles diferentes: equipo de desarrollo, productor interno, cliente y verificador beta. El equipo de desarrollo a su vez se divide en subroles, estos son los que se utilizan habitualmente en un equipo de desarrollo de videojuegos, diseñador, programador, artista gráfico y artista sonoro.

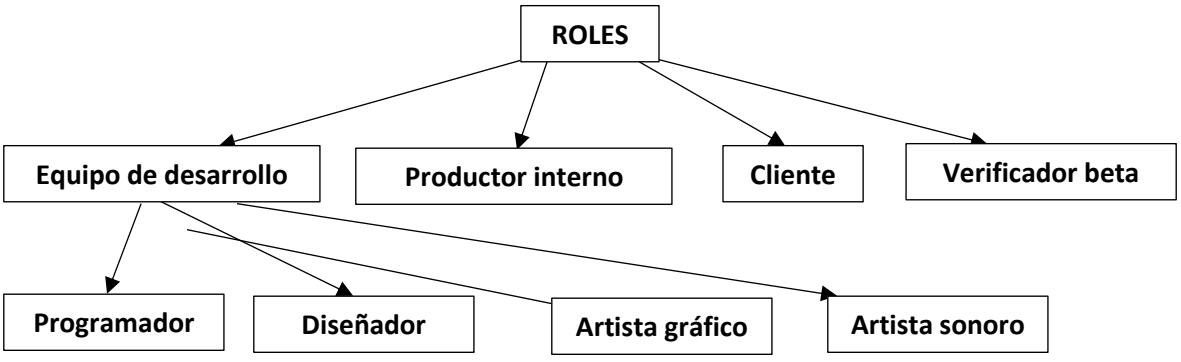

Figura 1: Roles de SUM

El productor interno y el cliente se refieren a los roles de *Scrum Master* y *Product Owner* que define la metodología *Scrum*, así entonces, el productor interno debería ser la persona conocedora de los procesos y encargada de orientar al resto del equipo, mientras que igual que en *Scrum*, se incluye al cliente o *Product Owner* dentro del proceso de desarrollo.

Dentro de los subroles del equipo de desarrollo se encuentran las distintas disciplinas que toman parte en la creación de todo el contenido del videojuego, según los autores mencionados anteriormente “es necesario esta definición ya que se requiere una alta especialización para satisfacer las distintas disciplinas que involucra del desarrollo de videojuegos, aspecto no contemplado en Scrum”.

El rol de verificador beta nace de la actividad común que existe en la industria del videojuego de exponer el producto a pruebas de funcionalidad dentro un grupo clasificado, con la responsabilidad de devolver el resultado.

Ciclo de vida

La metodología *SUM* se divide en cinco fases iterativas y secuenciales: concepto, planificación, elaboración, beta y cierre. Además de una fase llamada gestión de riesgos, que transcurre a lo largo de todo el proyecto.

En la fase de concepto, se filtran las ideas iniciales con las que ha sido concebido el proyecto, como el *Gameplay,* historia, personajes, características relevantes, entre otros. Además también se elaboran aspectos más técnicos como el modelo de negocios y los recursos para llevarlo a cabo, tanto humanos como tecnológicos. Todas estas propuestas son sometidas a discusión en reuniones y pruebas de concepto para analizar su factibilidad. Al tener un acuerdo común entre todas las partes se tiene un concepto validado y se da por cerrada esta fase.

En la fase de planificación se hace la estructuración de las fases subsiguientes, para lo que se define un cronograma del proyecto con los hitos que se deben alcanzar con cada iteración, además se dividen las tareas del equipo de trabajo y se establece un presupuesto para llevar adelante el proyecto. Esta fase tiene la

propiedad de ser flexible, ya que debe poder adaptarse a cambios que aparezcan en las consecuentes iteraciones y poder reflejar el estado actual del proyecto.

En la fase de elaboración se ejecuta la elaboración del videojuego, de una forma iterativa e incremental, obteniendo un producto ejecutable del videojuego al finalizar cada iteración. Esta fase se divide en tres etapas:

Se establece el hito a cumplir, cuales son las tareas necesarias para conseguirlo, las características a implementar y las métricas para el seguimiento del progreso.

Se desarrollan las características planificadas, con la ejecución de las distintas tareas previamente establecidas.

Se evalúa el resultado obtenido con respecto al esperado, dando así una base con la cual planificar la siguiente iteración.

La fase beta tiene como objetivo corregir errores que no se detectaron en la elaboración antes de que llegue la fase de cierre, además de evaluar aspectos importantes en un videojuego como la curva de aprendizaje, nivel de entretenimiento, dificultad, entre otros aspectos. La fase beta se da en distintas versiones del producto luego de que en la fase de elaboración ya se tiene un juego que es funcionalmente usable para un usuario final. Los resultados son evaluados y son tomados en cuenta para hacer ajustes al respecto y así comenzar un nuevo ciclo en la fase beta.

En la fase de cierre se hace la liberación del videojuego al cliente, se evalúa el éxito obtenido, los problemas que surgieron y los objetivos conseguidos.

La fase de gestión de riesgos se lleva a cabo en el transcurso de todo el proyecto, su objetivo es hacer un seguimiento de los posibles problemas que hagan presencia en un futuro, tratando de mitigar la posibilidad de que ocurran, establecer mecanismos de monitoreo y planes de contingencia.

CAPÍTULO III ELABORACIÓN

En este capítulo se describe el desarrollo del videojuego a través de cada una de las fases propuestas por la metodología Sum.

3.1 FASE 1 Concepto

A continuación se describe la presentación del concepto inicial del videojuego.

3.1.1 Definición de aspectos del juego

Mediante múltiples reuniones se propuso la creación de un videojuego en tercera persona, que posea las características de un juego rpg, de modo que el jugador pueda sumergirse en una aventura gráfica por un pequeño universo ficticio descrito por un guion o historia. Un título que se puede tomar como ejemplo a seguir o musa es la saga *The Legend of Zelda* (1986), guardando distancia, entre lo complejo y amplio que resulta la creación de Shigeru Miyamoto, en comparación a la meta que se desea conseguir con este proyecto.

3.1.2 Visión

Se desea que el juego de una experiencia al usuario entretenida y rápida, y principalmente que descubra cual es el desenlace de la trama del juego.

Para esto el jugador podrá desenvolverse en un mundo abierto, con distintas misiones que progresivamente irán narrando el desenlace de los hechos, con un grado de dificultad medianamente retador, para que el usuario deba explorar la mayor cantidad de opciones a la hora de resolver problemas y tomar decisiones.

El jugador debe sentir que se encuentra en un mundo con un alto grado de similitud al de la historia, así como darle un sentido a cada detalle del escenario para despertar la curiosidad del usuario en hallar el significado o causa de los elementos en su entorno.

3.1.3 Género

El género que mejor cumple con las metas propuestas es el de *rpg*, como se piensa incluir elementos de aventura, acción, exploración en casi todo momento, el género mas especifico en que se podría categorizar sería el de *action-rpg* de un solo jugador.

Un ejemplo conocido y exitoso de este género es la saga *The Witcher* (2007), que tiene un desarrollo conceptual fuerte, basado en las novelas escritas por el polaco Andrzej Sapkowski, en estos juegos el usuario toma el control del protagonista de la serie y recorre las aventuras narradas en los libros de una manera no tan lineal, tal como en un *rpg*.

3.1.4 Gameplay

El *Gameplay* es el aspecto mas retador y complejo debido a la gran cantidad de movilidad que debe tener el personaje principal, debe poder moverse en un entorno tridimensional libremente por el plano generado entre los ejes ZX, así como lograr saltos y caídas en el eje Y. La cámara que sigue al personaje debe cambiar de estados dependiendo de cuál sea la posición del jugador y las restricciones del entorno.

El jugador debe tener cambios de fase que le permitan moverse de una manera mas flexible de acuerdo a su situación, por ejemplo, si está explorando una ciudad debe moverse de una forma que le permita pasar calles, callejones, interactuar con otros personajes, entre otros; en cambio, en combate debe tener siempre en rango de visión a los personajes enemigos y moverse alrededor de su objetivo actual.

Mediante este esquema se pueden visualizar las distintas fases y estilo de movimiento, tanto del personaje como de la cámara que le hace seguimiento al mismo.

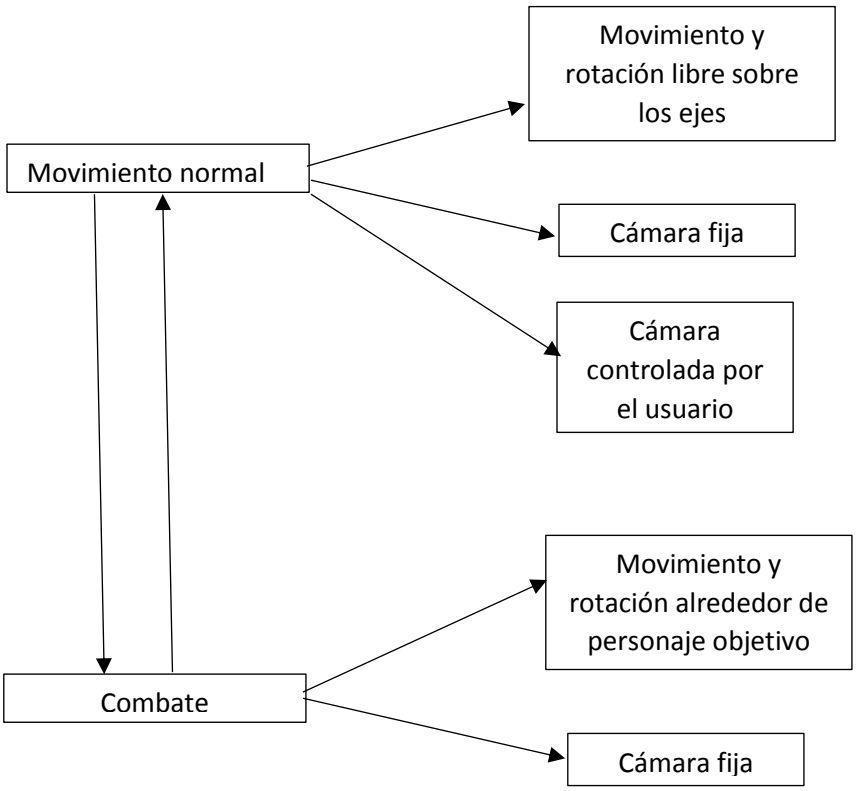

Figura 2: Esquema de movimiento del personaje

El jugador debe contar con una serie de habilidades para lograr sus objetivos, estas habilidades son en su mayoría de ataque, y el resto de movilidad. Con esta serie de posibilidades el jugador tendrá un considerable rango de acciones que puede tomar a la hora de encarar desafíos y tomar decisiones para conseguir logros.

La velocidad del gameplay debe ser alta, para que el usuario perciba mayor grado de acción en el transcurso del juego, no solo por parte del personaje principal sino también de los personajes antagonistas.

Nuevamente se puede citar los dos ejemplos dados anteriormente, *The Legend of Zelda* y *The Witcher* que también poseen fases similares en su *Gameplay*, básicamente ofrecen una jugabilidad similar con unas pocas variantes el uno del otro.

3.1.5 Características

Mundo abierto, el juego permitirá al usuario explorar libremente cualquier rincón del escenario en el momento que lo desee, por ello la mayoría de los hechos tendrán lugar en el escenario principal y todo el esfuerzo del equipo de desarrollo en el diseño y modelado del entorno estará enfocado en dicho escenario.

Narración continua, el jugador debe ir siguiendo el transcurso de los hechos a través de misiones, pistas encontradas dentro del mundo, diálogos con otros personajes, entre otros.

Curva de dificultad elevada, el juego debe significar un reto para los usuarios, de esa manera se garantiza más tiempo invertido en el mismo.

Mecánicas basadas en las habilidades del personaje, de esa manera el usuario debe pasar tiempo leyendo sus habilidades y realizando las combinaciones que crea resulten efectivas en la situación en que se encuentre.

3.1.6 Historia y ambientación

Algún tiempo el ser humano soñó con encontrar vida en algún planeta o estrella del vasto universo, o que alguna raza de humanoides llegaría a invadir la Tierra, único lugar habitado por millones de años; pero si la tierra llegara a ser visitada por criaturas de inteligencia superior en este momento, se darían cuenta de que alguien se adelantó. Como era de esperarse el día en que la Tierra se quedara sin recursos y la vida en este llegara a su ocaso aconteció, y encontró a la humanidad preparada para tal día. La ciencia avanzó lo suficiente como para permitir a la raza humana colonizar planetas lejanos, en donde podían establecerse una vida hostil, pero posible.

Nunca se hallaron rastros de otras formas de vida inteligente, por lo menos, nada mas complejo que microorganismos la mayoría de estos inofensivos. Como siempre imaginando que algún enemigo emergería de la oscura infinidad del universo, la humanidad nunca se dio cuenta de que la mayor amenaza estaba dentro de ella misma. Los hombres que llegaron al poder pronto se dieron cuenta que la recién nacida estructura política y social de la nueva civilización humana les

daba muchas libertades, y fue cuestión de tiempo para que las diferentes colonias empezaran a luchar entre ellas por igualdad de recursos y derechos, sin saber que solo eran marionetas de los grandes señores que gobernaban.

Una organización llamada "*The Automaton Theathre*" dedicada a la construcción de androides programados para servir como esclavos al hombre, decidió dar un paso adelante en la tecnología implementada para la creación de estas máquinas. Comenzó a usar cerebros humanos dentro de las máquinas con el fin de crear seres totalmente pensantes, tan inteligentes como el hombre mismo, resistentes gracias a sus placas de hierro, y ágiles como para recorrer mundos sin cansancio.

Las colonias humanas más grandes al darse cuenta del poder militar que podían conseguir con estos ciborg, planearon la adquisición de la organización T.A.T por cualquier vía posible. Al final en lugar de competir entre ellas, decidieron unirse en una sola gran nación, la "Gran unión de primeras colonias", y adoptar T.A.T como su fuerza armada. Las personas de los bajos estratos sociales, y los desactualizados primeros androides de estas mismas naciones fueron ofrecidos como materia prima para la creación del ejército más grande y poderoso que se haya visto.

Los ciborgs asolaron planetas y civilizaciones sin ninguna oposición que se pueda destacar, con cada asentamiento humano que caía el tamaño del ejército se elevaba, ya que usaban a los sobrevivientes para el desarrollo de nuevos miembros listos para incorporarse al combate, ahora del lado de sus anteriores enemigos.

Al planeta Atremedes, bautizado así por las primeras colonias, aunque sus habitantes solo lo conocían como "el mundo", uno de los más pobres e inhóspitos del universo conocido, le tocaba el turno de recibir la indeseada visita, en los días de crisis sus habitantes solo podían confiar en tal vez un puñado de hombres medianamente armados y un héroe de guerra que había impuesto su voluntad cuando la gente más lo necesitó. Nadie sabe dónde nació, de que familia viene ni cuál es su nombre, se le conoce como "Pein" y para algunos era mas un mito que

una realidad, se decía que fue el mismo mundo quien lo engendró y que no podía ser derrotado por ningún ser mortal, dándole así una connotación de semi-dios o figura divina semejante.

Pasados varios días del comienzo de la invasión ciborg, Pein apareció, y acabó con todas las unidades de la pequeña brigada con la que se esperaba conquistar el planeta. Esa sería la primera derrota conocida de la armada de las primeras colonias, pasando así a conocerse el nombre de Pein, fuera de Atremedes. Llegando esta información a los oídos del general Nalar, el ciborg mas temido por su crueldad y destreza en los campos de batalla, quien decidió ir personalmente a acabar con este héroe que se levantaba amenazante ante los planes de conquista de su poderosa nación.

El escenario aquí planteado es de un universo con dos contrastes, por un lado una mega organización con el monopolio de la tecnología y la fuerza militar, y por el otro la remanente población humana que lucha por sobrevivir en este ambiente post-apocalíptico.

Se busca que el escenario tenga influencia *vintage* y transmita la sensación de desolación que se busca, es necesario un nivel intermedio de detalles, tales como grietas, ruinas, entre otros.

El mapa debe ser un lugar árido en general, para ello la influencia del sol debe ser fuerte, con rayos solares en los terrenos abiertos y la iluminación bastante notable en todos los rincones.

Se establecen dos tipos de personajes en cuanto a su forma, por un lado figuras más humanas que pueden o no prótesis robóticas y por el otro lado robots con forma humanoide.

3.1.7 Definición de aspectos técnicos

Plataformas

El juego estará disponible para jugar en computadores personales con sistema operativo Windows.

Tecnologías y herramientas

Se eligieron las herramientas que el equipo de desarrollo estuviera en capacidad de utilizar y que, además, permitan ser usadas con el hardware que hay a disposición del equipo. Por lo tanto, las herramientas seleccionadas fueron las siguientes:

Unity 3D, como motor de juego. Permite crear productos con alta compatibilidad en las plataformas objetivos, además de contar con un editor relativamente ligero en comparación con otros motores.

Blender**,** como software de modelado 3D. Bastante ágil y ligero, y a la vez con gran potencia, además de ser gratuito.

SketchUp 2016, como software de modelado 3D alternativo, para las estructuras arquitectónicas que llevará el ambiente dentro del juego.

Adobe Photoshop CC, software de manejo de gráficos, con el que se crearan y editaran las texturas, además del diseño de todos los elementos necesarios para la interfaz de usuario.

Adobe Audition, para la creación y edición de los efectos de sonido que se emiten en todo el transcurso del juego.

iClone 6, software de diseño de personajes 3D y animaciones, para la creación de animaciones extras necesarias. Permite la creación de efectos de animaciones en avatares de una manera mas fácil y fluida que en otros programas como Blender.

Mixamo Fuse, software de diseño de personajes bípedos 3D, trae maquetas predefinidas de personajes con forma humanoide, que posteriormente pueden ser editadas y animadas en cualquier otro software.

Monodevelop, como IDE de programación, en lugar de Visual Studio, que son los dos IDE compatibles con el motor Unity. Se eligió monodevelop por ser más ligero y rápido que Visual Studio.

Adobe AfterEffect, para la creación y edición de efectos visuales, detalles importantes para la vistosidad del videojuego.

3.1.8 Modelo de negocios

Público objetivo:

El juego está orientado principalmente a jugadores asiduos de videojuegos (*gamers*), entusiastas de los juegos con perspectiva de tercera persona, que en determinado momento busquen una forma de entretenimiento rápida, sin dejar de considerar a cualquier otro tipo de jugador mas casual. Se estipula este objetivo debido a que este género es mas solicitado por personas que tienen el ocio electrónico como costumbre, por lo tanto, se debe aceptar este tipo de público como los que más curiosidad tendrán por probarlo.

Los desarrolladores *indies* siempre están interesados en conocer los proyectos homólogos, ya sea para evaluar competencia, medir capacidades, estudiar el mercado, simple entretenimiento, entre otros. Por lo tanto este proyecto al ser *indie*, por omisión será objetivo de este tipo de personas.

Los apasionados del género ciencia ficción también se esperaría sientan curiosidad de sumergirse en la narrativa que ofrece el videojuego, este género literario ha tenido gran éxito en la industria, y son apoyados por una numerosa comunidad.

Modelo de negocio a seguir:

El producto final estará enmarcado en el modelo de negocios *free to play*, el cual permite a los jugadores descargar y jugar sin necesidad de realizar algún tipo de pago. Este modelo de negocio es la contraparte de *pay to play* en donde se solicita un pago previo para poder obtener el videojuego.

El modelo *free to play* da la libertad al equipo de desarrollo implementar y explorar una gran variedad de métodos para recibir contribución económica, en este caso se proponen tres formas:

Ofreciendo a los usuarios conocer un concepto original, una historia profunda y un universo expandible con capacidad de ofrecer mucho más contenido en un futuro, se busca motivarlos a formar una comunidad de jugadores, apoyando el proyecto tanto en difusión como en el aspecto económico a través de donaciones voluntarias.

Ofrecer herramientas bajo cierto precio, que puedan utilizar otros desarrolladores *indies* para sus proyectos, empaquetados en *assets*, que pueden ir desde los modelos utilizados hasta librerías de código. De tal manera que puedan ser utilizadas en el desarrollo de proyectos similares o en la creación de *mods* (modificaciones del juego original por parte de la comunidad).

Publicar contenido extra al juego base de forma paga, los cuales pueden ir desde ítems hasta nuevos niveles.

3.2 FASE 2: PLANIFICACIÓN

3.2.1 Especificación del videojuego

A continuación se describen las características deseadas del producto final, teniendo en cuenta de que la proyecto es de un juego *rpg* en tercera persona, género que ya tiene propiedades básicas bien definidas.

3.2.2 Especificación de características

Características funcionales:

Libertad de movimiento para el jugador, de tal manera que sea más fácil y libre la exploración del escenario.

Continuación de la historia a través de misiones y objetivos que el jugador puede completar.

Ambientación del escenario conforme a la historia.

Interfaz gráfica relativa al concepto del juego

Mundo abierto.

Control de cámara automático.

Características no funcionales:

El videojuego debe ser en 3D y con modelos de bajos polígonos.

Interacción dinámica entre los personajes del juego.

Iluminación fuerte y detallada a lo largo del escenario.

Fluidez de la imagen considerable en equipos de gama media.

Criterios de evaluación:

Tabla 1: criterios de evaluación

N_o	Característica	Criterios de evaluación
1	Libertad de movimiento para el jugador, de tal manera que sea más fácil y libre la exploración del escenario.	Bajo costo de procesamiento Cubrir los estados básicos del jugador, tales como: inactivo, en movimiento, muerte, colisiones, entre otros. Tiempo de respuesta rápido a la entrada del usuario.
2	Continuación de la historia a través de misiones y objetivos que el jugador puede completar	Validación de todos los estados posibles: finalizada, fallida, en proceso. Verificación del cumplimiento de las condiciones de cada una.
3	Ambientación del escenario conforme a la historia	Uso de modelos de baja cantidad de polígonos Texturas semi-realistas Escala de modelos proporcional al tamaño del personaje principal.

Tabla 1. Continuación

N_o	Característica	Criterios de evaluación
4	Interfaz gráfica relativa al concepto del juego	Cubrir opciones de entrada del teclado. Diseño intuitivo. Pantallas de carga.
5	Mundo abierto	Colisiones sencillas Validar que no haya posibles atascamientos del personaje en alguna zona Bloquear salida del escenario.
6	Modelos en 3D de bajos polígonos	Medidas proporcionales
7	Interacción dinámica entre los personajes del juego	Rápidos tiempos de respuesta. Desplazamiento correcto a través del entorno. Evitar desplazamientos a largas distancias. Evitar aglomeración excesiva de entidades en algún punto.
8	Iluminación fuerte y detallada a lo largo del entorno	Sombreado correcto Sin zonas completamente oscurecidas
9	Fluidez de la imagen considerable en equipos de gama media	Al menos 30 fps en escenas con bajo movimiento y 15 fps en escenas más dinámicas.

3.2.3 Estimación de características

Estimación en tiempo de desarrollo de cada una de las características funcionales y no funcionales.

Tabla 2: Estimación de características

N_o	Característica	Tiempo
1	Libertad de movimiento para el jugador, de tal manera que sea más fácil y libre la exploración del escenario.	4 semanas
2	Continuación de la historia a través de misiones y objetivos que el jugador puede completar	1 semana
3	Ambientación del escenario conforme a la historia	4 semanas
4	Interfaz gráfica relativa al concepto del juego	2 semana
5	Mundo abierto	1 semana
6	Modelos en 3D de bajos polígonos	3 semanas
7	Interacción dinámica entre los personajes del juego	4 semanas
8	Iluminación fuerte y detallada a lo largo del entorno	2 semanas
9	Fluidez de la imagen considerable en equipos de gama media	1 semana

3.2.4 Priorización de características

Ponderación de cada una de las características según su peso e importancia en el desarrollo del videojuego. Como la misma metodología propone, se le da más importancia a los elementos del *gameplay*, que vendría siendo conformado por el control del personaje principal y su interacción con el medio, el siguiente grado de importancia se le asignó a la construcción del entorno, que es posible concretar de

mejor forma una vez que se hayan materializado los personajes que estarán dentro del mismo. Por último quedan los elementos más decorativos tales como la iluminación y ambientación.

Tabla 3: Ponderación de características

N_o	Característica	Ponderación
1	Libertad de movimiento para el jugador, de tal manera que sea más fácil y libre la exploración del escenario.	10
2	Continuación de la historia a través de misiones y objetivos que el jugador puede completar	6
3	Ambientación del escenario conforme a la historia	3
4	Interfaz gráfica relativa al concepto del juego	1
5	Mundo abierto	4
6	Modelos en 3D de bajos polígonos	5
7	Interacción dinámica entre los personajes del juego	8
8	Iluminación fuerte y detallada a lo largo del entorno	2
9	Fluidez de la imagen considerable en equipos de gama media	7

3.2.5 Objetivos del proyecto

Luego de determinar las características deseadas para el videojuego, se pueden obtener los objetivos a alcanzar para este proyecto.

1. Construir un controlador de personaje en tercera persona, para dar al personaje principal una buena movilidad y la capacidad de interactuar con el entorno.

2. Programar los controladores de personajes no jugadores (*npcs*) que les permita tomar acciones de acuerdo a su rol (hostil, neutral, aliado).

3. Elaborar una estructura de datos, que permita la ejecución simultanea de todos los elementos del entorno de una manera ordenada.

4. Diseñar el escenario y los elementos del ambiente, de acuerdo con el concepto artístico establecido.

5. Liberar una fase beta del juego.

Criterios de evaluación:

Obtener un producto jugable al llegar a la fase beta de desarrollo, y una aceptación notable en la descripción de la experiencia de juego de los testers.

Proporcionar a los jugadores una longitud de tiempo de entretenimiento aceptable.

Permitir a cada controlador administrar el comportamiento de las entidades en diferentes situaciones, y obtener resultados aceptables.

Cada clase controladora debe tener la capacidad de construir entidades diferentes en comportamiento.

3.2.6 Definición del equipo de desarrollo

Necesidades del proyecto:

Se cuenta con el equipo calificado para desarrollar el proyecto tanto en el aspecto de diseño y ambientación 3D como en el apartado dela programación de las funcionalidades que debe poseer el juego. El equipo de dos integrantes está en capacidad de concluir el proyecto en un plazo menor a un año.

Equipo de desarrollo:

Para este proyecto el equipo de desarrollo estuvo conformado por una única persona que tomó todos los roles y subroles establecidos por la metodología, exceptuando la fase beta donde por supuesto varias personas tomaron el papel de verificadores beta y enviaron su respectivo *feedback*.

También hubo consultas para verificar el concepto del juego, en la que varias personas voluntarias dieron su opinión acerca de la historia narrada.

Contratistas externos

Ya que no se cuenta con un personal dentro del equipo de desarrollo con conocimientos avanzados en la composición de audio, ni con el presupuesto para costear creaciones de algún tercero, se usarán *assets* de licencia libre con pequeñas ediciones, que cumplan, en la medida posible, con el concepto original del videojuego. Así como también se recurrirá al mismo recurso con las animaciones de los personajes, conforme a que no se tiene a disposición un estudio de animación, debido a la considerable cantidad de animaciones necesarias y tomando en cuenta de que no hay un gran número disponible de manera gratuita, se crearán algunas animaciones cortas con el software iClone 4.

3.2.7 Definición del cronograma de elaboración

Se definió el cronograma de desarrollo de la aplicación, la cantidad y tiempo de sprints, hitos y planificación de la fase beta.

Cronograma de elaboración

Cada sprint tendrá una duración de dos semanas, teniendo como resultado de cada uno, alguna característica en funcionamiento del videojuego, algunas características que se estiman lleven un tiempo de más de 2 semanas han sido divididas en dos módulos, tal es el caso de las características 1, 3 y 7.

En el caso de la característica 1: “libertad de movimiento para el jugador, de tal manera que sea más fácil y libre la exploración del escenario”, debido a su tiempo estimado y al considerable alcance que esta implica, se ha dividido en los siguientes módulos:

1. Manejo básico de desplazamiento y rotación a través de escenario tridimensional, esto implica, lectura de entradas por el teclado, dando al usuario la facultad de modificarlas a gusto, así como las respuestas del personaje a dichas entradas, pudiéndose desplazar en la dirección ordenada.
2. Control de los flujos alternativos que pueden ocurrir durante el desplazamiento, tales como, caídas, saltos, cambios de velocidad, inclinaciones, colisiones, entre otras.

La interacción con los personajes del juego puede ser de distintas formas, las cuales podemos clasificar en dos grupos, amistosa y hostil, en base a esta clasificación se dividirían los dos módulos de la característica 7 “interacción dinámica entre los personajes del juego”.

1. Elaboración de los personajes no controlados por el jugador (*npc*), que toman papel en el concepto histórico del videojuego.
2. Creación de los personajes enemigos, diseñando sus modelos y el algoritmo que administra su comportamiento, lo que implica también, el desarrollo de un sistema de enfrentamiento o combate.

En cuanto a la característica 3: “ambientación del escenario conforme a la historia”, ya que es costoso en cuanto a tiempo generar todos los modelos deseados, esta característica puede dividirse según dos procedimientos básicos: modelado y texturizado. Por lo tanto se extraen dos módulos bien definidos:

1. Modelado de las estructuras del escenario, con esto nos referimos a los objetos estáticos dentro del entorno tridimensional.
2. Texturizado de los modelos conseguidos anteriormente.

Al culminar ambos módulos también se estaría cumpliendo en gran medida con la característica 6 “modelos en 3D de bajos polígonos”. Y que estaría finalizada al concretar la característica 5 “mundo abierto”, por ende, no hará falta planificar un sprint dedicado a cumplir con esta propiedad.

Otra de las características no funcionales, como “fluidez de la imagen en equipos de gama media”, tampoco se le asignará un sprint adicional, debido a que debe ser un aspecto a tomar en cuenta a lo largo del desarrollo.

Las características 2 y 5, “continuación de la historia a través de misiones y objetivos que el jugador puede completar” y “mundo abierto” respectivamente, debido a su corta duración estimada (1 semana) pueden ser abordadas en un único sprint.

La cantidad de sprints resultante es de 9, con una duración de 2 semanas cada uno, lo que se traduce en un tiempo de desarrollo estimado de 18 semanas. Empezando con la elaboración de los dos módulos derivados de la característica 1, por ser la que posee un número más elevado en la priorización (10). Una vez finalizados dichos sprints, la siguiente característica en prioridad es la 7, con una ponderación de 8, que implica la construcción de las entidades no manejadas por el jugador.

Tabla 4: Cronograma de sprints

Sprint	**Fecha de inicio**	**Fecha de culminación**	**Descripción**
1	23-05-2016	04-06-2016	Controlador básico del personaje principal.
2	06-06-2016	18-06-2016	Finalización del personaje principal y su controlador.
3	20-06-2016	02-07-2016	Elaboración de personajes no controlados por el jugador (npc).
4	04-07-2016	16-07-2016	Desarrollo de los personajes enemigos y sistema de enfrentamientos.
5	18-07-2016	30-07-2016	Construcción de escenario básico y recorrido de misiones.
6	01-08-2016	13-08-2016	Modelado de la ambientación y decoración del entorno.
7	15-08-2016	26-08-2016	Diseño e implementación de texturas en los modelos del entorno.
8	28-08-2016	10-09-2016	Diseño y mapeado de iluminación.
9	12-09-2016	24-09-2016	Diseño e implementación de la interfaz de usuario.

3.2.8 Definir cronograma de beta

Se definió que la fase beta estará compuesta por 3 iteraciones de una semana respectivamente, que se estiman deben ser las suficientes para llevar a cabo una exitosa corrección de errores, y un posterior cierre del desarrollo. Cada sprint incluye tres días de prueba y recepción de reportes de los testers, y otros tres días de corrección de errores.

Se dará inicio a la siguiente semana luego de haber concluido el último sprint, dando como resultado el siguiente cronograma:

Tabla 5: Cronograma de beta

Iteración	Tipo de actividad	Fecha de inicio	Fecha de culminación
1	Prueba y recepción de reportes	26-09-2016	28-09-2016
	Corrección de errores	29-09-2016	01-10-2016
2	Prueba y recepción de reportes	03-10-2016	05-10-2016
	Corrección de errores	06-10-2016	08-10-2016
3	Prueba y recepción de reportes	10-10-2016	12-10-2016
	Corrección de errores	13-10-2016	15-10-2016

Al iniciar la tercera iteración, no debe haber errores en el videojuego que afecten la experiencia de juego del usuario, si concluyen los tres días de prueba y no se reportan nuevos errores de esta índole, se puede dar por concluida la fase beta, de lo contrario deben ser corregidos en los tres días subsiguientes y programar una nueva iteración para la siguiente semana.

Los errores menores, es decir, aquellos que no afectan de forma significativa la experiencia de juego de los usuarios, tales como, *bugs* visuales u objetos decorativos con comportamiento incorrecto, pueden o no, ser tomados en cuenta, esto quedará a criterio del equipo de desarrollo.

3.2.9 Definir cierre de proyecto

La finalización del proyecto se estimó para la semana posterior a la última iteración de la fase beta, específicamente el día 22 de octubre de 2016, luego de haber realizado las últimas correcciones según el reporte de los usuarios.

3.2.10 Definir hitos

A continuación, la serie de hitos que se definieron para demarcar la evolución del proyecto:

Tabla 6: Hitos

Hito	Descripción	Fecha
1	Personaje principal completamente funcional, con modelo y texturas finalizadas e implementadas.	18-06-2016
2	*Gameplay* totalmente funcional, interacción entre objetos en funcionamiento.	16-07-2016
3	Producto básico funcional, jugable y listo para ser sometido a pruebas.	30-07-2016
4	Producto funcional con un entorno virtual más definido al del hito anterior, sin detalles decorativos tan resaltantes.	26-08-2016
5	Escenario terminado con acabados decorativos y sin modificaciones pendientes.	10-09-2016
6	Producto listo para ser puesto en fase beta	26-09-2016
7	Videojuego listo para ser liberado, con todas las correcciones realizadas.	17-10-2016

Fase 3 Elaboración

3.3.1 Sprint 1

Descripción: Controlador básico del personaje principal.

Objetivos:

Crear una estructura de datos para el control del personaje por parte del usuario.

Poder comprobar la movilidad de un personaje prototipo usando el controlador elaborado.

Métricas:

Obtener un código flexible y reutilizable por las clases que se implementarán en las iteraciones subsiguientes.

Comportamiento correcto en un terreno con obstáculos, colisiones e inclinaciones.

Animaciones y transiciones suaves y con cortos tiempos de respuesta.

Seleccionar características:

Control del usuario por el teclado, con capacidad de ser modificado.

Movimiento direccional por los ejes X, Y, Z. Con restricciones de velocidad en el retroceso.

Cambios de velocidad.

Colisiones y obstáculos.

Cámara simple que siga el movimiento del personaje.

Desplazamiento del personaje por animación.

Capacidad de cambiar de estilo de rotación, rotación libre a rotación en torno a algún objeto y viceversa.

Truncar desplazamiento para evitar solapamiento con otros personajes.

Refinamiento de características:

Tabla 7: Refinamiento de características sprint uno

N_o	Descripción	Tareas
1	Control del usuario por el teclado, con capacidad de ser modificado.	Crear clase controladora de entradas del usuario. Crear clase controladora del personaje con conexión a la clase de entradas del usuario.
2	Movimiento direccional por los ejes X, Y, Z. Con restricciones de velocidad en el retroceso.	Crear máquina de estado con las animaciones pertinentes. Truncar áreas de desplazamiento y saltos.
3	Cámara simple que siga el movimiento del personaje.	Crear clase controladora de cámara del personaje.
4	Capacidad de cambiar de estilo de rotación, rotación libre a rotación en torno a algún objeto y viceversa.	Programar identificación de objetos adyacentes. Establecer modo de rotación entorno al eje de un objeto adyacente.
5	Truncar desplazamiento para evitar solapamiento con otros personajes.	Determinar distancia mínima entre personajes.

Desarrollo de características

A continuación, se describe el proceso de desarrollo de cada una de las características listadas anteriormente.

Selección de tarea:

1.1 “Crear clase controladora de entradas del usuario”. Debido a que esta será una clase que debe ser usada por las demás, se seleccionó esta en primer lugar, según el siguiente funcionamiento propuesto:

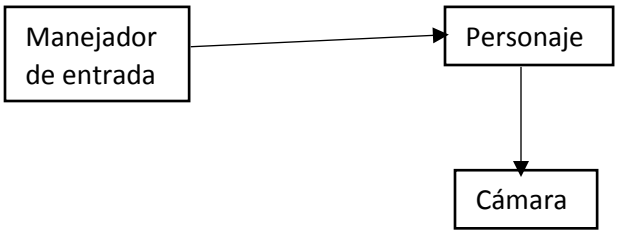

Figura 3: Funcionamiento de la clase controladora de entradas

Ejecución de tarea:

Básicamente el personaje debe responder al receptor, que en este caso es la clase controladora de entradas, que también tendrá la misma función para todos los elementos del juego que requieran de la orden del usuario. La cámara en tanto debe seguir el desplazamiento del personaje, aunque luego el usuario también podrá modificar su posicionamiento.

La clase resultante fue la siguiente:

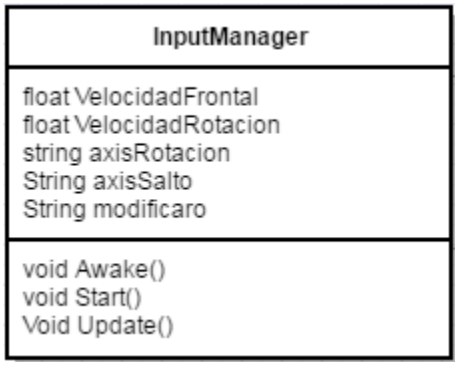

Figura 4: Clase controladora de entradas

La clase deriva de *Monobehaviour*, y además, es de tipo *singleton* lo cual permite que las entradas solo sean leídas por esta única clase, verifica constantemente la entrada de las teclas establecidas, mediante los métodos *get* se puede obtener si la tecla ha sido presionada en ese instante o no.

Verificación de tarea:

Los métodos de la clase están hechos para servir a cualquier otra clase, además, solo puede haber una única instancia de ella, por su particularidad de ser *singleton*, por ende, su código puede ser reutilizado cuantas veces sea necesario, cumpliendo así, con las métricas establecidas para esta iteración.

Selección de tarea:

1.2 “Crear clase controladora del personaje con conexión a la clase de entradas del usuario”. Continuando con el funcionamiento explicado anteriormente, el siguiente en abordar sería la clase que lleva el control directo del personaje.

Ejecución de tarea:

Se programaron las funciones de movimiento de manera que en las funciones *get* solicite a la instancia *singleton* de la clase InputManager el estado actual de la entrada por teclado, mientras la velocidad puede ser establecida y modificada de cualquier manera. La clase quedó de la siguiente manera:

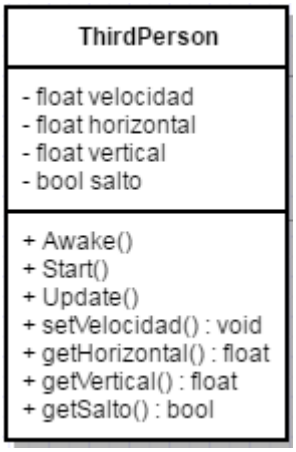

Figura 5: Clase controladora de personaje

Verificación de tarea:

Se creó un sistema de salida que muestre los resultados por pantalla a través de *Unity*, de tal forma que, permita monitorear el comportamiento de la clase, obteniendo el resultado esperado a todas las entradas por teclado posibles. Funcionando para un sistema genérico de monitoreo se asume que los métodos de la clase son reutilizables y funcionales para muchos casos, cumpliendo así con las métricas establecidas.

Selección de tarea:

2.2 “Crear máquina de estado con las animaciones pertinentes”, para posteriormente continuar con la elaboración de la clase del personaje, haciendo la conexión hacia la máquina de estados.

Ejecución de tarea:

Se implementó la capa base de la máquina de estados del personaje, solo con los estados de movimiento y sus transiciones.

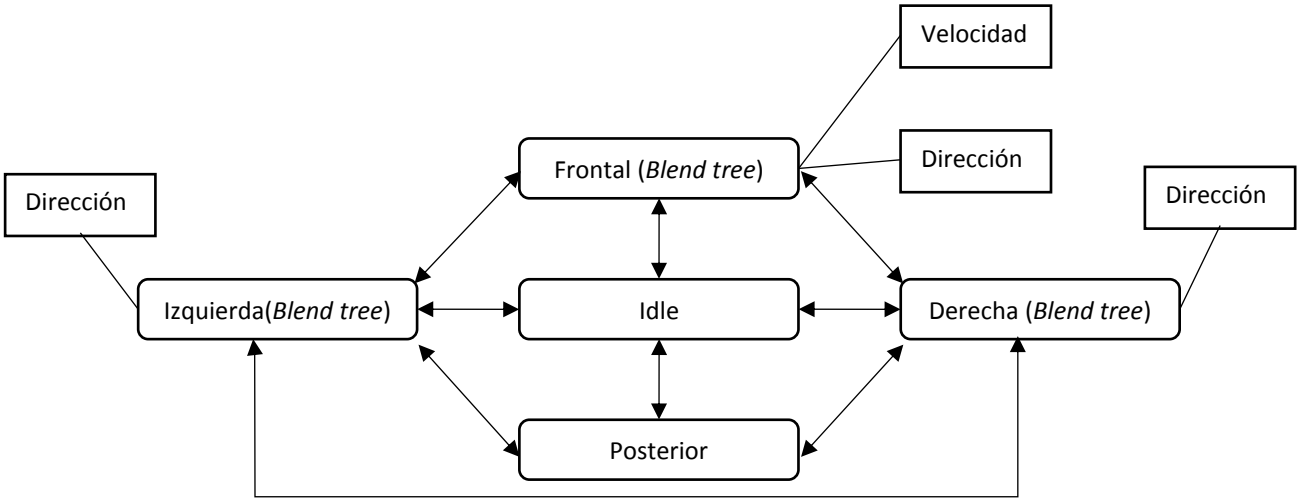

Figura 6: Máquina de estado

Todas las direcciones de movimientos están formadas por un *blend tree*, a excepción del posterior, debido a que se estableció un parámetro de velocidad fijo para este tipo de movimiento. El *blend tree* del movimiento frontal toma dos parámetros, velocidad y dirección, lo que le permite al personaje cambiar el valor de las mismas sin necesidad de detenerse, o que esta máquina de estado tenga que hacer alguna transición, consiguiendo así, un tiempo más rápido de respuesta. Las direcciones, derecha e izquierda, representan rotaciones del personaje cuando este se encuentra en estado de reposo, con un único parámetro en su *blend tree* que define el multiplicador de rotación.

Verificación de tarea:

Ya con la clase controladora y la máquina de estados lista, se puso a prueba el personaje, logrando movimientos ágiles, animaciones fluidas y transiciones suaves pero a la vez rápidas, cumpliendo así con las métricas ya determinadas.

Selección de tarea:

2.2 “Truncar áreas de desplazamiento y saltos”, para completar la configuración de movilidad del personaje y ser capaz de transitar por cualquier terreno.

Ejecución de tarea:

Se dio inicio con la detección de colisiones del personaje, para ello, se dio uso de los objetos de Unity para crear campos de colisión, que no son más que mallas sin un material que renderizar pero con restricciones en sus bordes. Para el personaje se utilizó un cilindro de radio 6.79 y de 91.9, valores proporcionales que usa el motor de Unity y que no representan realmente un patrón de medidas de uso cotidiano.

También se le adicionó al *gameobject* del personaje un objeto del tipo *Character Controller*, que proporciona una simulación no tan real de físicas como la gravedad y el cálculo de fuerzas de choque, se decidió por este tipo de objeto en lugar de un *rigidbody*, para evitar lidiar con características de la física como momentum, masa, rebotes, entre otros; que generan comportamientos incomodos en el movimiento del personaje.

Una vez listo el manejo de las colisiones, se procedió con los saltos y caídas, se planificaron tres estados diferentes para facilitar el manejo a nivel funcional del personaje, salto, caída y caída profunda. Cada uno de estos estados fue añadido a la capa base de máquina de estados. Para conocer cuando el personaje se encuentra en una situación de caída profunda se usó la función *raycast*, pasando como parámetros la posición en el eje Y del personaje y el vector dirección (0,-1,0), consiguiendo como resultado la distancia entre el personaje y el primer colisionador que encuentre en dicha dirección. La distancia para entrar en el estado de caída profunda se estableció como 5.0.

Verificación de tarea:

Se creó una escena de prueba, llamada “test”, con distintas mallas simples que permitieran probar el desplazamiento del personaje por una superficie bastante irregular, gracias a la serialización de las variables en el inspector de unity se hicieron rigurosas pruebas con distintas cifras, y se terminó por aumentar el valor de caída profunda de 5.0 a 8.0, ya que deja un margen más amplio de caídas y

una sensación mayor de versatilidad del personaje, que es una de las características que se desea conseguir.

Selección de tarea:

3.1: “Crear clase controladora de cámara del personaje”, se seleccionó esta tarea para poder tener un mejor seguimiento del personaje, además, de ser un elemento importante para poder conseguir un producto entregable al final de la iteración.

Ejecución de tarea:

Se creó una clase que además de seguir al personaje pueda cambiar de posición a criterio del usuario, para así poder visualizar otros ángulos que no son posibles si solo se anexa el *gameobject* cámara como hijo del *gameobject* personaje.

Se utilizó dentro de la función evento *LateUpdate* de la clase *monobehaviour* la función *vector3.Lerp*, la cual permite la interpolación lineal entre dos puntos en el espacio, enviando como parámetros los vectores de posición de la cámara y el personaje.

Para posibilitar al usuario de cambiar el ángulo de la cámara a su gusto, se estableció una entrada por teclado que solicitaría esta posición, la cual por defecto se estableció como el botón izquierdo del mouse. Capturando esta entrada por la clase controladora “input”, y llamando a una nueva función que cambia la posición de la cámara hacia los ejes de coordenadas Y y X sobre donde se encuentre el puntero, de ese modo el jugador puede mover la cámara con solo mover el mouse en la dirección deseada.

Verificación de tarea:

Se abrió la escena “test” y se hicieron las pruebas en el personaje principal, la cámara funcionó del modo deseada y es funcional para fines de prueba y control.

Selección de tarea:

“Programar identificación de objetos adyacentes”. Con el personaje principal ya funcionalmente avanzado, es posible crear nuevos métodos que facilitarán en el futuro cercano la implementación de otras características.

Ejecución de tarea:

Para identificar los objetos adyacentes se comenzó con la forma más sencilla que fue implementar un objeto *trigger* dentro del *gameobject* del personaje, con forma circular y un radio de 6 a escala de *Unity* de tal manera que sobresalga del personaje. Se programó la identificación de objetos que entren en ese radio de acción, pero además, ya que el objeto trigger es invisible y sobrepasa todos los componentes del espacio, se verifica que no haya ningún tipo de oclusión entre el personaje y el objeto dentro del trigger, a través de la técnica raycast.

Verificación de tarea:

Se abrió la escena “test” y se dio inicio a la prueba, se añadieron varios objetos que hicieran la función de personajes, agregándoles el *tag* “personaje”, en efecto, se logró exitosamente la verificación correcta de los objetos con este *tag* al cumplir las condiciones establecidas.

Selección de tarea:

“Establecer modo de rotación entorno al eje de un objeto adyacente”, continuando con los métodos que proveen la amplitud necesaria a la clase controladora del personaje para luego manejar otro tipo de funcionalidades a futuro.

Ejecución de tarea:

Fue posible lograr que el personaje rotara entorno a cualquier eje operable estableciendo en su controlador dos estados de rotación, libre y hacia otro objeto, respectivamente, cuando el estado de rotación hacia otro objeto pasa a ser activo, el estado libre queda inactivo, así, el personaje deja de cambiar de dirección por orden del jugador, y solo cambia apunta al objeto destino, como su ejecución sucede en el evento *Update*, si el objeto cambia de posición, la rotación del personaje cambia en función a este.

Verificación de tarea:

Como ya es posible la identificación de objetos externos en el entorno, se hizo la prueba en la escena “test” en donde se ubicaron diferentes *GameObjects* con el *tag* “personaje”, al identificar alguno de estos, el personaje cambió su tipo de rotación.

Selección de tarea:

“Determinar distancia mínima entre personajes”, ya con el comportamiento básico del personaje realizado, solo queda evitar que se solape con otros objetos que no sean estáticos en el entorno, ya que, estos no tendrán un colisionador igual que los demás objetos.

Ejecución de tarea:

Se añadió al controlador de personaje, las funciones de modificación de velocidad, y se añadió una restricción de 2 a escala Unity, en la distancia a la que el personaje puede acercarse a otro.

De esta manera, cuando alcanza una distancia de 2 hacia un objeto en específico, el controlador automáticamente reduce la velocidad a cero, en caso de que el usuario continúe solicitando por teclado seguir avanzando, el controlador traducirá esta solicitud en movimiento lateral, de modo que, el personaje no quede estático al alcanzar la distancia mínima y pueda seguir desplazándose con facilidad, la posibilidad de retroceder siempre está habilitada.

Verificación de tarea:

Se abrió la escena de prueba y se comprobó el correcto funcionamiento al establecer restricciones hacía objetos con un *tag* especifico, cumpliendo con las características deseadas.

SEGUIMIENTO DE LA ITERACIÓN

A continuación se describe como se realizó el monitoreo de la iteración en el transcurso de la misma.

Manejo de problemas:

La carencia de assets libres de animación, fue el primer problema encontrado al momento de construir la máquina de estado, se tomaron animaciones del controlador de personaje en tercera persona que dispone Unity, los cuales no son los ideales para el concepto de personaje con el que se manejó, pero sirvieron para efectos de prueba.

El manejo del inspector de Unity dejó “expuestos” muchos atributos de las clases, con el propósito de facilitar la verificación de las tareas, por lo que se deben corregir en una iteración posterior para evitar fallos de seguridad.

Control del estado del proyecto a través de cada tarea

Tabla 8: Estado de tareas sprint uno

Tarea	**Descripción**	**Estado**
1	Crear clase controladora de entradas del usuario.	Concluida en el tiempo estipulado.
2	Crear clase controladora del personaje con conexión a la clase de entradas del usuario	Concluida en el tiempo estipulado.
3	Crear máquina de estado con las animaciones pertinentes.	Concluida en el tiempo estipulado.
4	Truncar áreas de desplazamiento y saltos.	Concluida en el tiempo estipulado.
5	Crear clase controladora de cámara del personaje.	Concluida en el tiempo estipulado.
6	Programar identificación de objetos adyacentes.	Concluida en el tiempo estipulado.
7	Establecer modo de rotación entorno al eje de un objeto adyacente.	Concluida en el tiempo estipulado.
8	Determinar distancia mínima entre personajes.	Concluida en el tiempo estipulado.

Registro de medidas:

A través de todas las pruebas que se hicieron en la verificación de cada tarea, se corroboró que cada proceso cumplió con las métricas previamente establecidas, obteniendo resultados altamente aceptables.

Cierre de la iteración:

Se presenta el estado actual del juego después de culminar el proceso del primer sprint, en donde se programaron los controladores necesarios para el personaje principal.

Evaluación del estado del juego:

Se dispone ya como producto, la base del *gameplay*, que es un personaje con perspectiva en tercera persona, con libre movimiento a través de un entorno, lo cual, es ya un producto entregable y que puede ser sometido a pruebas.

Evaluación de la iteración

El transcurso de la primera iteración se desarrolló sin inconvenientes destacables, el flujo de tareas fue el correcto, por lo que no hubo necesidad en ningún momento de retroceder a una tarea previa, gracias a la creación de un escenario de prueba fue posible abarcar un amplio espectro de sucesos, lo que da cabida a la creación de un entorno sin preocuparse tanto por las limitaciones del *gameplay*. El enfoque de las métricas de este sprint estuvo orientado a desarrollar un punto de partida para los componentes subsiguientes, de tal forma que se pueda desarrollar un framework bastante flexible para modificar o crear un juego de este estilo, dando pie así, a conseguir el modelo de negocios propuesto, en el desarrollo de este sprint dichas metas son de vital importancia, y a través de las pruebas se comprobó que han sido alcanzadas.

Actualización del plan de proyecto

Las tareas se cumplieron en su totalidad a la fecha establecida (04-06-2016) en el plan de proyecto, el primer hito sigue intacto para la fecha (18-06-2016).

3.3.2 Sprint 2

Descripción: Finalización del personaje principal y su controlador.

Objetivos:

Construir el *asset* del personaje principal (concepto, modelo, texturas).

Implementar el *asset* con el controlador de tercera persona programado.

Métricas:

Todos los elementos deben estar acordes al concepto original del videojuego.

Malla y *rig* sin defectos visibles.

Funcionamiento correcto con las animaciones preestablecidas.

Seleccionar características:

Malla de bajos polígonos.

Movimientos del personaje flexibles y suaves.

Desplazamiento del personaje por animación.

Texturas con buen nivel de detalle.

Refinamiento de características:

Tabla 9: Refinamiento de características sprint dos

N_o	**Descripción**	**Tareas**
1	Malla de bajos polígonos	Modelar personaje. Diseñar concepto artístico del personaje.

Tabla 9. Continuación

N_o	Descripción	Tareas
2	Movimientos del personaje flexibles y suaves.	Elaborar *rig* del personaje.
3	Desplazamiento del personaje por animación.	Exportar *asset* a Unity.
4	Texturas con buen nivel de detalle.	Obtener mapeado UV de la malla. Crear e implementar texturas.

Selección de tarea:

“Diseñar concepto artístico del personaje”, para iniciar con el proceso de creación del personaje es necesario tener un patrón que seguir, que lo dará la visualización gráfica del personaje.

Ejecución de tarea:

Después de haber analizado la descripción del personaje principal, y haber tomado varias ideas de ilustraciones de terceros que siguen la misma corriente, se llevó a cabo la elaboración del *sketch*.

Verificación de tarea:

El sketch pasó por la aprobación de todo el equipo, se verificó que cumple con el concepto del juego y del personaje.

Selección de tarea:

“Modelar personaje”, Se procedió con la implementación en 3D del *sketch* que se hizo en la tarea anterior.

Ejecución de tarea:

Para la elaboración de la malla se usó el motor gráfico Blender, se usaron las técnicas de esculpir y retopología, luego de haber obtenido una forma básica geométrica. Se exportaron los detalles de vestuario con el software Mixamo Fuse, así como también se aprovecharon muchas texturas dadas por este mismo, por último, se modelaron detalles importantes como los brazos y el arma del personaje.

Verificación de tarea:

La malla contiene un total de 10000 polígonos, lo que se considera dentro del rango aceptable para ser el personaje principal, las texturas exportadas por el software Fuse cumplen con las exigencias planteadas, además, la malla se corresponde al *sketch* usado de guía.

Selección de tarea:

“Elaborar *rig* del personaje”, Una vez ya la malla a disposición, se pudo comenzar con la implementación de un *rig* que pueda capturar las animaciones.

Ejecución de tarea:

Para esta tarea se continuó haciendo uso de las herramientas que dispone blender, se elaboró una estructura ósea no tan compleja, se omitieron los componentes de la cara del actor y elementos de cinemática inversa.

Verificación de tarea:

Se probó el *rig* en el software Blender dando movilidad a todas las articulaciones, se pudo verificar que no suceden alteraciones indeseadas en la malla y los movimientos son los correctos, se omitieron detalles casi imperceptibles de solapamiento en la malla del vestuario.

Selección de tarea:

“Obtener mapeado UV de la malla.”, para proceder al texturizado del modelo es necesario obtener el mapeado UV, ya que el modelo está concluido, se dio inicio a esta tarea.

Ejecución de tarea:

Se marcaron las costuras de la malla que aún estaban sin texturas, se desplegó el mapa UV y fue exportado del software Blender a Photoshop para el tratamiento de las texturas.

Verificación de tarea:

Se rellenó el mapa UV con los colores del sketch que se realizó para el personaje, al probarlo en Blender la alineación de las texturas fue la correcta.

Selección de tarea:

“Crear e implementar texturas”, al tener el mapeado UV realizado, es posible comenzar con el texturizado de la malla.

Ejecución de tarea:

Con el software Photoshop, se realizaron las texturas correspondientes, con el fin de conseguir la calidad de detalles esperada, se diseñaron varios tipos de texturas, de modo que, se pueda sacar provecho del *standard shader* de Unity, estas fueron, albedo, normal y *occlusion*.

Verificación de tarea:

Tal como en la tarea anterior, se volvió a visualizar el modelo en el software Blender, en el cual se reflejaron correctamente las texturas realizadas, se pudieron detallar los normales y el *occlusion map*, los cuales aportan un agradable detalle gráfico.

Selección de tarea:

“Exportar *asset* a Unity”, al concluír con las tareas de modelado y texturizado, es posible exportar el *asset* a Unity, para ser implementado como actor principal.

Ejecución de tarea:

En Blender se exportó la figura en formato fbx, y seguidamente, importado al proyecto en Unity, se configuró el *rig* como de tipo humanoide, se crearon los materiales con las texturas hechas, utilizando el *standard shader* que dispone Unity, y finalmente, se creó un *gameobject* que representa al personaje, que contiene el modelo y el controlador programado en la primera iteración.

Verificación de tarea:

En el escenario de prueba se dio inicio a las pruebas de controlador realizadas en la primera iteración, esta vez con el nuevo modelo, comprobando el buen funcionamiento de las animaciones y el comportamiento de la malla, consiguiendo buenos resultados.

SEGUIMIENTO DE LA ITERACIÓN:

A continuación se describe como se realizó el monitoreo de la iteración en el transcurso de la misma.

Manejo de problemas:

Haber planificado la producción del concepto gráfico del personaje en esta iteración retrasó por un lapso de tiempo, el desarrollo de la misma, sin embargo, la totalidad de las tareas lograron ser concluidas al tiempo estipulado.

El producto conseguido (*asset* de personaje), fue de alta calidad, consiguiendo un buen detalle gráfico a un costo no tan alto de gpu, como lo confirmó la herramienta de *profiler* del motor Unity.

El personaje representa muy bien el planteamiento de concepto del videojuego, algo que se logró con la elaboración de prototipos en papel antes de proceder al modelado en 3D del mismo.

Control del estado del proyecto a través de cada tarea

Tabla 10: Estado de tareas sprint dos

Tarea	Descripción	Estado
1	Diseñar concepto artístico del personaje.	Concluida a dos días posteriores al tiempo establecido.

Tabla 10. Continuación

Tarea	Descripción	Estado
2	Modelar personaje.	Concluida en el tiempo estipulado.
3	Elaborar *rig* del personaje.	Concluida en el tiempo estipulado.
4	Obtener mapeado UV de la malla.	Concluida en el tiempo estipulado.
5	Crear e implementar texturas	Concluida en el tiempo estipulado.
6	Exportar *asset* a Unity.	Concluida en el tiempo estipulado.

Registro de medidas:

El personaje lleva en gran medida de su diseño el concepto del juego, en el escenario de pruebas no se notaron defectos en la malla o en el transcurso de las animaciones que fueran muy perceptibles, además, se utilizaron todas las animaciones disponibles para verificar el funcionamiento del rig y en todas hubo resultados positivos.

CIERRE DE LA ITERACIÓN

Se presenta el estado actual del juego después de culminar el proceso del segundo sprint, en donde se diseñó e implementó el *asset* del personaje principal.

Evaluación del estado del juego:

Con la culminación de las tareas correspondientes a la segunda iteración, se obtuvo un personaje completo con la capacidad de navegar por el escenario, identificar otros objetos y cambiar su modo de movimiento.

Evaluación de la iteración

Las métricas y objetivos planteados en la planificación fueron conseguidos exitosamente, la prueba final en Unity se realizó junto con la visualización del *profiler*, obteniendo bajo consumo de los recursos del computador, lo cual da espacio a los elementos que aún se están por desarrollar e integrar al proyecto.

Actualización del plan de proyecto

Las tareas se cumplieron en su totalidad a la fecha establecida (18-06-2016) en el plan de proyecto, y con él, se llegó al primer hito del proyecto, el cual era obtener un personaje totalmente funcional, acorde al concepto, que pueda ser controlado por el jugador y desplazarse libremente.

3.3.3 Sprint 3

Descripción: Elaboración de personajes no controlados por el jugador (npc).

Objetivos:

Implementar una estructura de datos que permita crear y manipular entidades que representan personajes en el videojuego.

Crear controlador de misiones y objetivos.

Métricas:

Posibilidad de diseñar personajes a través del inspector de Unity sin necesidad de reescribir código.

Tiempos de respuesta rápidos.

Seleccionar características:

Navegación del personaje

Identificación de otros objetos

Diálogos

Misiones

Refinamiento de características:

Tabla 11: Refinamiento de características sprint tres

N_o	**Descripción**	**Tareas**
1	Navegación del personaje	Crear controlador de personaje. Configurar navegación de personaje.
2	Identificación de otros objetos	Elaborar métodos de identificación en el controlador de personaje

Tabla 11. Continuación

N_o	Descripción	Tareas
3	Diálogos	Crear interfaz de diálogos
		Programar manejador de diálogos
4	Misiones	Crear controlador de misiones.

Selección de tarea:

“Crear controlador de personaje”, Se inició lógicamente con la clase controladora básica del personaje, para luego implementar sobre la misma, todas las funcionalidades.

Ejecución de tarea:

Se identificaron dos tipos de personajes no controlados por el jugador (*npc*), aliados y enemigos, ambos totalmente distintos en su comportamiento, por lo que se decidió construir dos clases controladoras diferentes, con métodos básicos de interacción, y una máquina de estado para cada uno, que habilite las acciones esenciales como, desplazamiento y giro.

Se trabajó primero con los personajes amistosos, se diseñó una clase con métodos y atributos básicos que debe llevar el objeto.

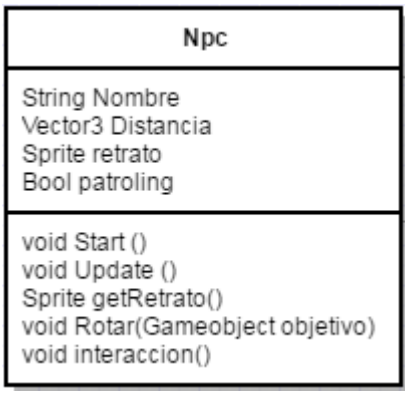

Figura 7: Clase controladora de npc

Toma los elementos de la clase base *monobehaviour* para hacer uso de las funciones eventos que servirán para identificar los objetos, cambiar de estado y desplazarse, además se añadió un sprite como atributo que servirá como retrato del personaje y puede tener múltiples usos, uno de ellos, el cuadro de diálogos, en donde se puede mostrar el nombre del personaje junto con su retrato y su respectivo discurso. Es capaz de almacenar el vector distancia entre dos objetos, con el fin de saber si otra entidad se encuentra cerca, y también, se le añadió la posibilidad de rotar su eje para colocarse frente a algún objeto determinado.

Se creó una máquina de estado con solo el estado *Idle*, debido a que aún el objeto no tiene la función de desplazarse.

Para el otro tipo de personaje, el cual debe tener una actitud totalmente distinta, como por ejemplo, no puede dialogar y obligatoriamente debe desplazarse.

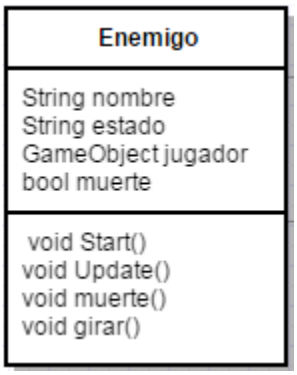

Figura 8: Clase controladora personaje enemigo.

Se añadió la función de muerte, que desactiva el *GameObject* asociado a la clase, función esencial para el manejo de este tipo de actores.

Verificación de tarea:

Debido a que la clases controladoras aún son muy abstractas y no tienen un funcionamiento claro dentro del juego, fue difícil al culminar esta tarea, realizar pruebas de funcionamiento.

Selección de tarea:

“Configurar navegación de personaje”, se tomó la movilidad de los personajes para iniciar con las funcionalidades que deben poseer los mismos.

Ejecución de tarea:

Se decidió trabajar con la movilidad de los personajes enemigos, ya que es una función esencial para su funcionamiento, por el contrario, para los personajes amistosos es más un detalle, por lo que se decidió postergar.

Dado que, Unity posee un componente llamado *NavMesh*, se utilizó este elemento para manejar el cálculo de caminos de los personajes (*Pathfinding*), se añadió y configuró el componente en un *GameObject* de prueba, la comunicación de los diferentes bloques (*NavMeshAgent*, controlador, máquina de estado), se planteó de la siguiente forma:

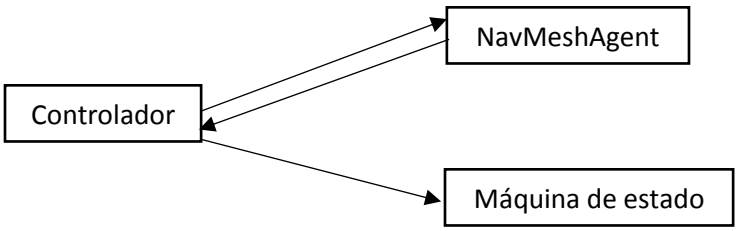

Figura 9: comunicación de componentes de personajes npc.

El controlador envía ordenes al *NavMeshAgent* de los caminos que debe buscar, este devuelve resultados como, si se encuentra en vía, si hay un camino disponible, distancia restante al destino, entre otros; en base a esta respuesta, el controlador modifica los parámetros de la máquina de estado, para que así, cuando el personaje se encuentre en movimiento, se lleve a cabo la animación de caminar o correr, dependiendo del resto de los parámetros.

Para que el controlador pueda ejecutar todas estas acciones, se implementaron las funciones necesarias para enviar y recibir información del *NavMeshAgent*. Se agregó la función opcional de patrullar un área determinada, asignándole un grupo de nodos que el agente debe recorrer, también es posible ejecutar un desplazamiento rápido o lento (correr o caminar), esto para cuando el actor deba decidir entre cual tipo de movimiento debe realizar. El actor será capaz de seguir cualquier objeto que le sea asignado, siempre y cuando encuentre previamente una ruta disponible desde su posición actual hasta la ubicación de dicho objeto. Es posible interrumpir su estado “patrullar” para realizar cualquier acción, y posteriormente retomar esta actividad en la misma posición donde se encontraba.

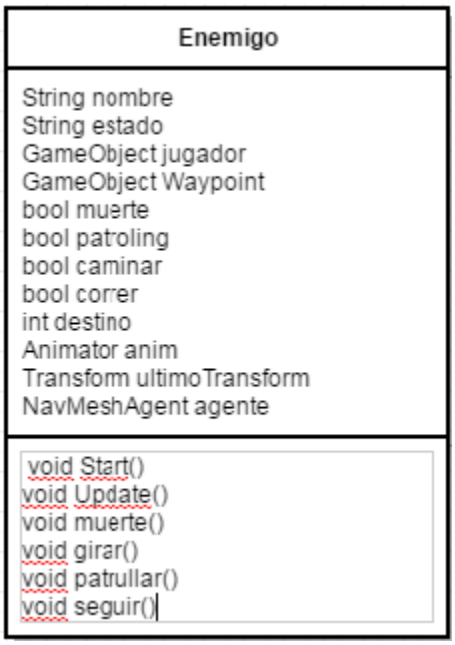

Figura 10: Clase controladora de personaje enemigo dos

Se diseñó una máquina de estado que pueda ejecutar las acciones básicas de movimiento que tendrá el personaje, además se adicionó el estado de muerte, para asegurar de que no haya ninguna transición hacia algún otro estado para cuando ocurra este evento.

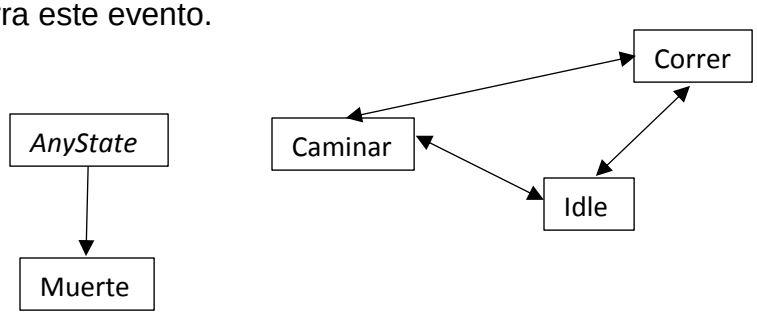

Figura 11: máquina de estado personaje enemigo

Verificación de tarea:

En el escenario de prueba se agregó un objeto con todos los elementos que conforman el funcionamiento de este tipo de personaje, se agregaron varios *GameObjects* vacíos que tomaran el papel de nodos para la navegación.

El objeto de prueba pudo navegar a los nodos correctamente, patrullar por ellos, dirigirse a un objeto con ruta disponible y permanecer estático en el estado idle si así lo ameritaba.

Selección de tarea:

“Elaborar métodos de identificación en el controlador de personaje”, se procedió con la capacidad del personaje de interactuar con el jugador, que es una propiedad importante en el producto final.

Elaboración de tarea:

El actor debe tener en su controlador un identificador de personaje, que puede ser cambiado en cualquier momento a través del editor, también es necesario conocer la posición actual del jugador, para posteriormente poder determinar la acción a tomar. El primer paso fue tomar la distancia entre el *npc* y el jugador, cálculo que se realizó de la siguiente manera:

Magnitud(Posición jugador – posición personaje)

El número resultante es comparado con una variable de proximidad, la cual se puede modificar solo por el inspector. Debido a que puede darse el hecho de que hayan objetos con colisión en medio de los personajes, se agregó otra condición, a través del método *raycast* para confirmar que no haya obstáculos en el vector dirección, esto es útil para evitar reacciones indeseadas cuando el personaje está en la distancia deseada pero hay un muro u obstáculo entre ambos personajes.

Verificación de tarea:

En el escenario de prueba se pusieron en marcha el personaje principal y un objeto de prueba que tiene el papel de *npc*, al acercarse al *npc* este lo detecta correctamente, se detectó un inconveniente, en el cual, al encontrarse en el medio un obstáculo con elevación mínima que no debe evitar que se divise el personaje, el método *raycast* no devolvía la posición del jugador.

Para resolver el problema se agregó un objeto vacío en la posición de los ojos del personaje, desde dicha posición, el método *raycast* actuaría, en lugar de hacerlo en el punto de origen del personaje como lo hacía previamente.

Selección de tarea:

“Crear interfaz de diálogos”, una de las acciones que puede tomar un personaje amistoso es dialogar con el jugador, antes de implementar esta función es necesario crear la interfaz para su visualización.

Ejecución de tarea:

En el software photoshop se creó un cuadro de diálogo en donde se mostrará el texto y se exportó como un *sprite* al proyecto en Unity.

Se creó un *canvas* en donde se mostrarán todos los elementos de la interfaz de usuario, y se ubicó el cuadro de diálogos en la parte inferior del mismo, por defecto se colocó como desactivado para que el controlador correspondiente solo lo active cuando sea necesario.

Dentro del cuadro de diálogo se realizó una división de tres secciones, una para mostrar el retrato del personaje, otra para el nombre del mismo y finalmente una con el resto del espacio para serializar el diálogo.

Verificación de tarea:

En la escena de pruebas se activó y desactivó el cuadro de diálogos para observar que se muestra correctamente, también se hicieron pruebas para verificar que se mostrara de forma correcta en distintas resoluciones de pantalla.

Selección de tarea:

“Programar manejador de diálogos”, Se procedió a crear la funcionalidad de diálogos, ya con la interfaz para hacer posible su visualización.

Ejecución de tarea:

Fue necesario crear una clase controladora de tipo *singleton* ya que no deben haber varios diálogos activos a la vez, por lo tanto la clase debe estar instanciada una única vez y ser usada solo cuando se vaya a realizar un diálogo. Funciona por solicitud de un personaje *npc*, este según sea necesario, envía el texto y la clase controladora de diálogos es encargada de activar el canvas y mostrarlo en pantalla.

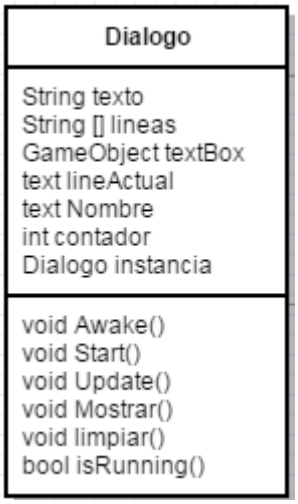

Figura 12: Clase controladora de diálogos

Todas las instancias de la clase controladora de personaje pueden acceder al sistema de diálogos, pero siempre se mostrará el resultado del último en acceder.

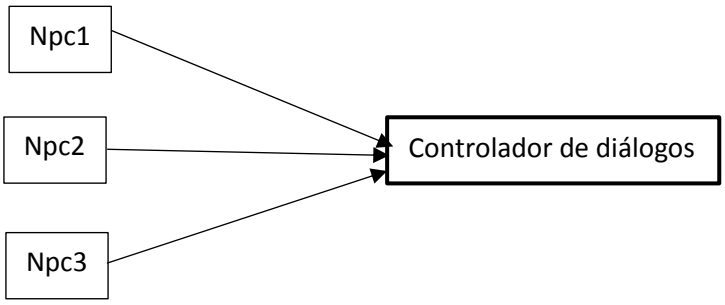

Figura 13: Mecanismos de conexión con clase controladora de diálogo

Como la clase es independiente de las cadenas de textos, los npc deben tener cargado el texto y enviarlo para así ser mostrado, esto conlleva a, que cada personaje deba administrar sus propias frases.

Se modificó la clase controladora de personajes amistosos, añadiéndole un atributo string que contiene el diálogo correspondiente, para evitar escribir el texto directamente sobre la clase, se implementó un mecanismo de carga sobre archivos en formato Json, en tiempo de carga cada instancia busca en este archivo si hay algún diálogo que corresponde al mismo.

Finalmente, se agregó la funcionalidad de conversación, en la cual, el jugador al alcanzar determinada distancia y hacer click en el personaje, este muestra el diálogo que tiene cargado, en caso de tener uno.

Verificación de tarea:

Se abrió el escenario y se hicieron pruebas con objetos que contengan las propiedades de un personaje amistoso, se creó un archivo Json con varias conversaciones que fueron asignadas a los objetos creados. Al cumplirse la condición, se mostró correctamente el cuadro de diálogo con las conversaciones que fueron escritas en el archivo.

Selección de tarea:

“Crear controlador de misiones”, otra de las acciones que puede realizar un *npc* amistoso es otorgar misiones al jugador.

Ejecución de tarea:

Debido a que para el seguimiento de una misión necesita que se lleve un registro estadístico de las acciones del jugador, se optó en primer lugar por crear una instancia que monitoree dichas acciones, y a través de esta, conseguir los puntos de partida y finalización de cualquier misión.

Se creó una clase del tipo *singleton* y con las funciones eventos que se extienden de la clase *monobehaviour* de la siguiente forma:

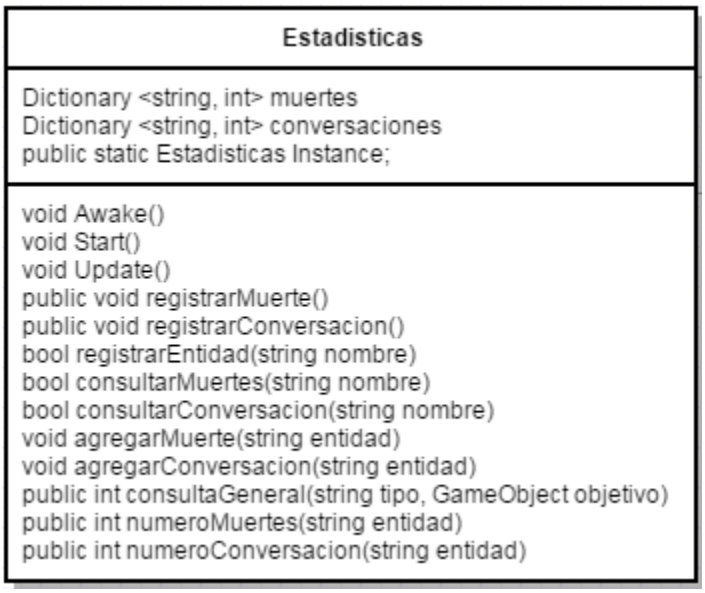

Figura 14: Clase controladora de estadísticas

Se trabajó con dos tipos de acciones, muertes para llevar registro de los enfrentamientos, y conversaciones con los npc. Mediante dos atributos de tipo *dictionary* se lleva el conteo de estas dos acciones.

Posteriormente se creó una clase que conforma de por si una misión, en la cual se establecen los atributos básicos que puede tener un objeto de este tipo y los cuales se establecerán por el contenido del controlador estadístico.

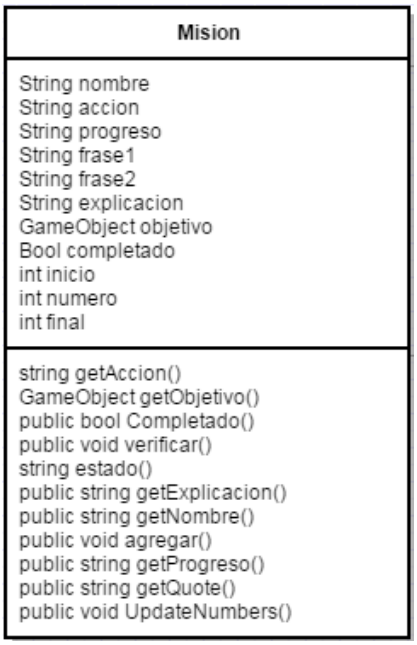

Figura 15: clase controladora de diálogos

Para manejar un array de misiones, fue necesario crear una clase de instanciación única que tomara los objetos de la clase anterior y los modifique según la información que obtenga de la instancia de estadísticas.

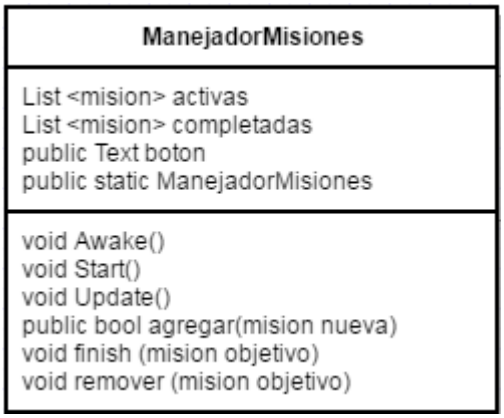

Figura 16: Clase controladora de diálogos

Esta clase es la encargada de agregar las misiones que tome el jugador, y en el evento *Update* se encarga de comparar los requisitos de cada uno de los elementos en la lista de misiones, con el recolector de estadísticas, si la condición

de finalización se cumple, la misión es transferida a otra lista con las misiones ya completadas.

Finalmente, se estableció un atributo en la clase controladora de los *npc* de tipo *mision*, la cual puede o no estar vacía, en caso de no estarlo, el jugador puede tomar dicha misión, y en ese instante se agrega a su array de misiones. Se agregaron los métodos para que el *npc* entregue la misión que tiene disponible, y además, se agregó la opción de mostrar la explicación de esta en sus métodos asociados a la conversación.

Verificación de tarea:

En el escenario de pruebas, se crearon varios objetos de prueba con la función de *npc*, cada uno con diálogos preestablecidos y uno en específico con una misión disponible, la cual consistió en hablar con el resto de los *npc*, fue posible tomar la misión y al realizar las tareas correspondientes fue marcada como completada.

Para el resto de las acciones posibles en una misión, no fue posible a este punto verificarlas, ya que, el sistema de combate aún está ausente del proyecto.

SEGUIMIENTO DE LA ITERACIÓN

A continuación se describe como se realizó el monitoreo de la iteración en el transcurso de la misma.

Manejo de problemas:

Algunos aspectos ya descritos anteriormente aún no fueron verificados al 100% hasta este punto, debido a la necesidad, de los componentes restantes que aún no estaban implementados, sin embargo, la mecánica de estos es similar a las que si pudieron ser testeadas.

Fue necesario invertir tiempo en solucionar algunos problemas de movilidad de los personajes, en los cuales se daban malas impresiones visuales, como por ejemplo, rotaciones irrealistas, movimiento no proporcional a las animaciones,

entre otras, detalles que no cubre el sistema de navegación con el que dispone Unity.

Control del estado del proyecto a través de cada tarea

Tabla 12: Estado de tareas sprint tres

Tarea	Descripción	Estado
1	Crear controlador de personaje	Concluida en el tiempo estipulado.
2	Configurar navegación de personaje.	Concluida en el tiempo estipulado.
3	Elaborar métodos de identificación en el controlador de personaje	Concluida en el tiempo estipulado.
4	Crear interfaz de diálogos	Concluida en el tiempo estipulado.
5	Programar manejador de diálogos	Concluida en el tiempo estipulado.
6	Crear controlador de misiones.	Concluida en el tiempo estipulado.

Registro de medidas:

A través de todas las pruebas que se hicieron en la verificación de cada tarea, se corroboró que cada proceso cumplió con las métricas previamente establecidas, obteniendo resultados altamente aceptables.

Cierre de la iteración:

Se presenta el estado actual del juego después de culminar el proceso del tercer sprint, en donde se programaron los controladores básicos de los personajes no controlados por el jugador.

Evaluación del estado del juego:

A este punto, se dispone de un producto funcional con capacidad de recorrer el entorno, interactuar con otros objetos y recibir misiones y completarlas, solo restando el sistema de enfrentamientos para concluir con el *gameplay*.

Evaluación de la iteración

El transcurso de la iteración ocurrió con normalidad, a pesar de algunos inconvenientes sorteados debido a las limitaciones del sistema de búsqueda de caminos de Unity, hubo que trabajar en optimizar la movilidad de los personajes, tanto en mecánicas como en rendimiento.

Se consiguió un código bastante flexible y reutilizable, que podría ser adaptado a mecánicas distintas a las que fueron planteadas, la comunicación entre las clases se logra de manera sencilla y logran un comportamiento consistente, gracias a esto, es posible manejar todos los objetos de *npc* a través del inspector de Unity, sin necesidad de volver a escribir código.

Actualización del plan de proyecto

Las tareas se cumplieron en su totalidad a la fecha establecida (02-07-2016) en el plan de proyecto, el segundo hito sigue intacto para la fecha (16-07-2016).

3.3.4 Sprint cuatro

Descripción: Desarrollo de los personajes enemigos y sistema de enfrentamientos.

Objetivos:

Agregar los métodos correspondientes a enfrentamientos para los controladores de personajes.

Implementar manejo de objetos y artefactos para el personaje principal.

Métricas:

Tiempos de respuestas cortos, que permitan visualizar enfrentamientos ágiles.

Ajuste de movilidad en los enfrentamientos, de tal manera que el usuario tenga la mayor facilidad de moverse y cambiar de objetivo

Seleccionar características:

Manipulación de armas y artefactos por parte del personaje principal

Manejo de habilidades para todos los personajes

Control de estados de enfrentamiento

Refinamiento de características:

Tabla 13: Refinamiento de características sprint cuatro

N_o	**Descripción**	**Tareas**
1	Manipulación de armas y artefactos por parte del personaje principal	Crear controlador de objetos Implementar los métodos de interacción con los objetos

Tabla 13. Continuación

N_o	Descripción	Tareas
2	Manejo de habilidades para todos los personajes	Programar manejador de habilidades para los personajes Implementar mecanismos de toma de decisiones para los personajes enemigos
3	Control de estados de enfrentamiento	Crear manejadores de salud Programar estados de combate

Selección de tarea:

“Crear controlador de objetos”, un elemento clave para el manejo de enfrentamientos será el control de las armas, por lo que cada una de estas deben manejar propiedades para ser identificadas por los personajes.

Ejecución de tarea:

Se creó una clase base para las armas, la cual contiene las coordenadas en las que se debe posicionar de acuerdo a la situación del personaje. Estas situaciones son identificadas como superior y reposo, superior se refiere al momento en que el personaje se encuentra empuñando el arma, y reposo lo contrario.

Además de esto, es necesario identificar la hoja del arma cuerpo a cuerpo, para poder conocer cuando la hoja impactó o no a un objetivo, para fines de agrado de *gameplay*, el tamaño de colisión de la hoja se hizo con un radio considerablemente mayor al del que realmente es, al punto de que sea más fácil alcanzar objetivos y que el jugador tenga aún la sensación de que fue impactado por colisión de la hoja.

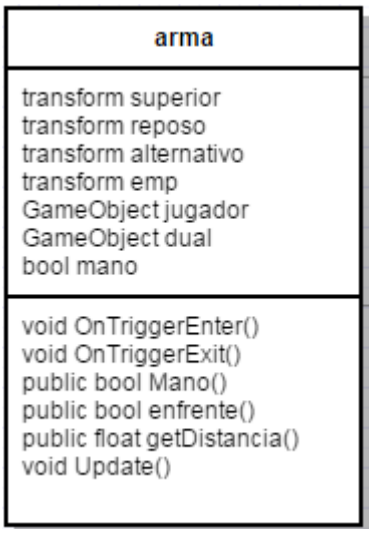

Figura 17: Clase controladora de artefactos

Verificación de tarea:

Se creo un objeto 3D de prueba y se le anexionó la clase programada, esta logró identificar al jugador y tomar su distancia, se calcularon los vectores de cada posición y fueron agregados como atributos.

Selección de tarea:

“Implementar los métodos de interacción con los objetos”, ya que el objeto tiene una clase y puede identificar al jugador, ahora es necesario que el jugador pueda manipular este objeto, este debe poder levantar y guardar el arma, cambiando la posición de esta según las coordenadas específicas que contiene su clase.

Ejecución de tarea:

Se decidió crear una clase dedicada al manejo de las armas, en lugar de modificar la clase controladora del jugador, esto con el fin de tener clases más modulares que se dediquen cada una a alguna labor en específico.

Mediante esta clase se permite al jugador tomar acciones como guardar el arma, equipar una diferente, o cambiarla, por lo tanto se agregó un nuevo método a la

clase controladora de entradas, en la cual el jugador puede empuñar y cambiar de arma.

Debido a que hay animaciones excluyentes de otras, como lo es la manipulación de las armas, ya que esta debe impedir que se realicen otras acciones, se utilizaron corutinas para manejar este aspecto, las cuales permitieron realizar un seguimiento de la animación establecida en la máquina de estados a través de código.

Para la máquina de estados se creó una nueva capa, con solo tres estados, que representan las animaciones posibles para el manejo del arma.

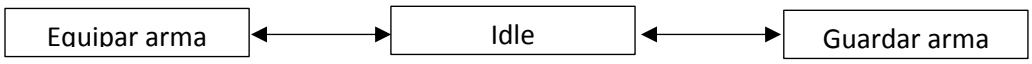

Figura 18: capa dos de máquina de estado

Esta nueva capa en la máquina de estado tiene enlazada una máscara de avatar que solo usa los brazos, de ese modo, la animación solo altera la malla del personaje en dichos puntos, mientras el resto se encuentran realizando otra animación que corresponda a la situación.

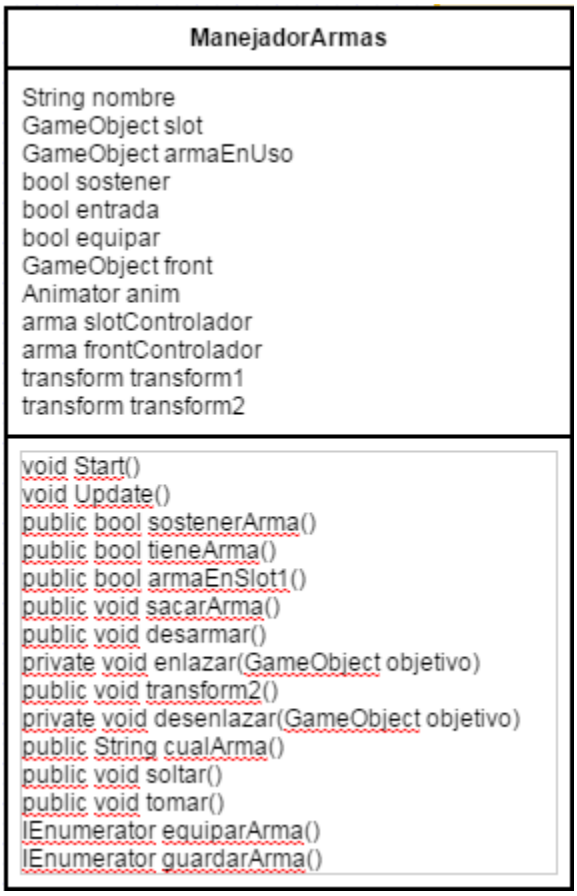

Figura 19: Clase manejadora de artefactos y armas.

Verificación de tarea:

Se anexó al personaje principal un objeto 3D que cumpliera el rol de arma, se agregaron las clases correspondientes, el personaje correctamente manipuló el arma según la orden del usuario, además se verificó que se pudiera hacer en todas las situaciones, como desplazamiento, salto, entre otras.

Selección de tarea:

“Programar manejador de habilidades para los personajes”, cada personaje debe manejar una serie de habilidades que conforman las mecánicas del juego, estas habilidades pueden ser usadas tanto por el personaje principal como por los personajes enemigos.

Ejecución de tarea:

Se buscó implementar un sistema de habilidades que permitiera crearlas por la interfaz del inspector de Unity, sin tener que volver a escribir código, para esto primero se introdujo una clase básica, llamada “habilidad”, con los atributos que tendría una entidad de este tipo en cualquier juego *rpg*, como lo son, daño, tiempo de reutilización, costo, entre otras. Debido a que esta clase, al no ser *monobehaviour*, no heredaría las funciones eventos del motor de Unity, y tampoco se conectaría con componentes necesarios como la máquina de estado del personaje, se diseñó otra clase que comunique la clase “habilidad” con la máquina de estados, esta de igual forma sin ser del tipo *monobehaviour*, pero heredando los atributos y métodos de la primera. De esta forma se logra que una clase se encargue de almacenar los atributos necesarios en el *gameplay* y otra clase heredada se encargue de enlazar la habilidad a los componentes de Unity para su funcionamiento y visualización, además, esto permite la creación de un objeto de tipo habilidad a nivel de interfaz de gráfica.

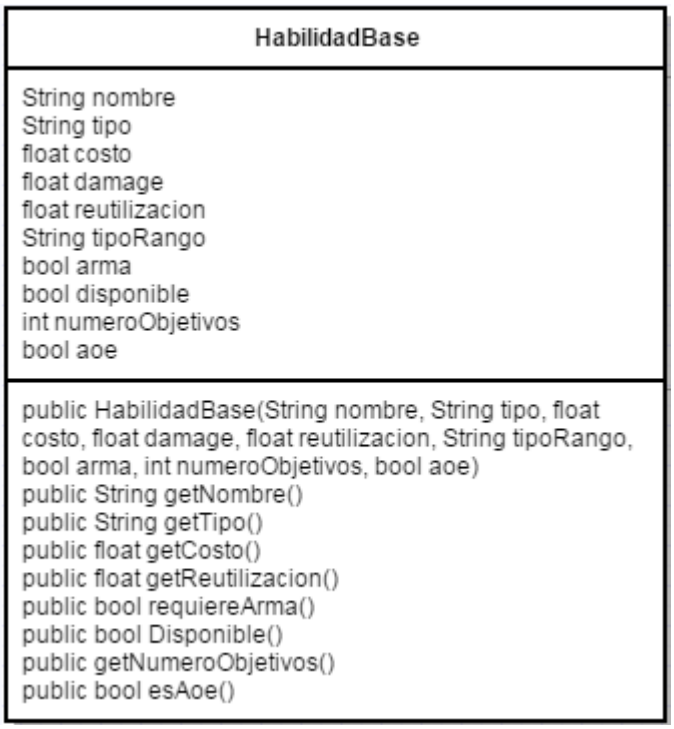

Figura 20: Clase base de habilidades

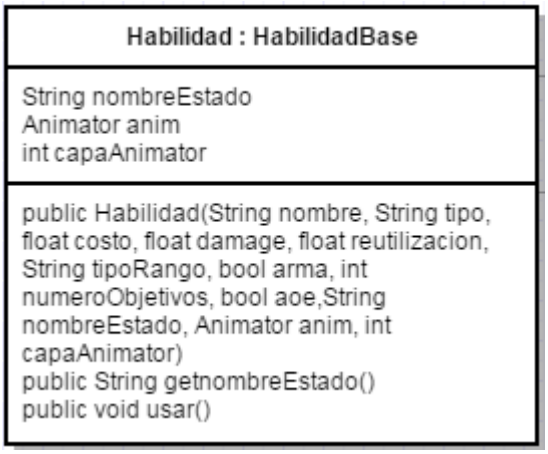

Figura 21: clase heredada de habilidades

Para almacenar las habilidades creadas por interfaz, se trabajó con un archivo Json que contenga todas las habilidades que sean escritas por el desarrollador, después de haber diagramado la estructura de dicho archivo, se programó una extensión de Unity para crear la interfaz de usuario, con una entrada de texto por

cada atributo de la clase habilidad, y al pulsar al botón aceptar, se agrega esta nueva habilidad al archivo Json.

Para asignar las habilidades al personaje, se diseñó una clase *monobehaviour*, que manejara un array de habilidades, esta clase se encarga de inicializar las habilidades, extrayendo las indicadas por el array desde el archivo Json a la memoria principal, disparar el efecto de las habilidades, verificar condiciones de uso, accionar la máquina de estado y manejar los tiempos de reutilización. Esta clase debe estar instanciada en cada personaje que pueda hacer uso de las distintas habilidades.

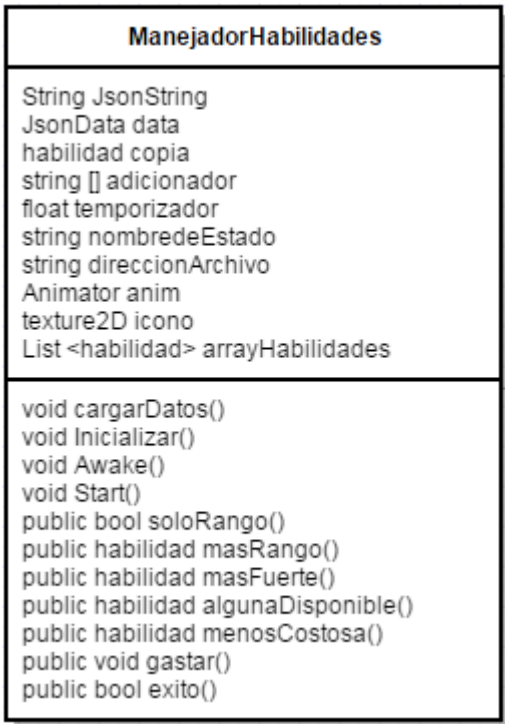

Figura 22: clase controladora de habilidades

La clase posee métodos de evaluación del conjunto de habilidades, como determinar cuál es la más fuerte, la de mayor rango, entre otras. Esto con la finalidad de servir de entrada de información a la inteligencia artificial de los personajes enemigos.

En la siguiente figura se puede detallar el funcionamiento del sistema de habilidades

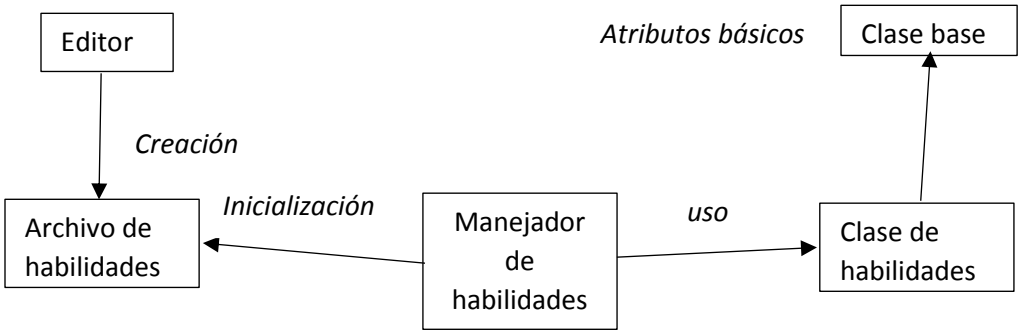

Figura 23: funcionamiento del sistema de habilidades

Verificación de tarea:

Se generaron diversas habilidades a través del editor, se creó una nueva capa en la máquina de estado del personaje principal en donde cada uno representa la animación de cada habilidad, se escogieron animaciones prediseñadas de Unity.

La clase controladora de entradas por teclado fue modificada para aceptar un nuevo grupo de entradas que estén enlazadas con el array de habilidades. Al realizar las pruebas, el personaje correctamente realizó las acciones que le correspondía, y la reutilización de las habilidades funcionó de manera correcta igualmente.

Selección de tarea:

“Implementar mecanismos de toma de decisiones para los personajes enemigos”, para continuar con la prueba del sistema de habilidades es necesario dotar a los personajes enemigos con inteligencia artificial que les permita tomar decisiones sobre su array de habilidades.

Ejecución de tarea:

Se procedió con la creación de un árbol de comportamiento que contenga las acciones deseadas para ser realizadas por los personajes o *bots*, este árbol quedó descrito de la siguiente manera:

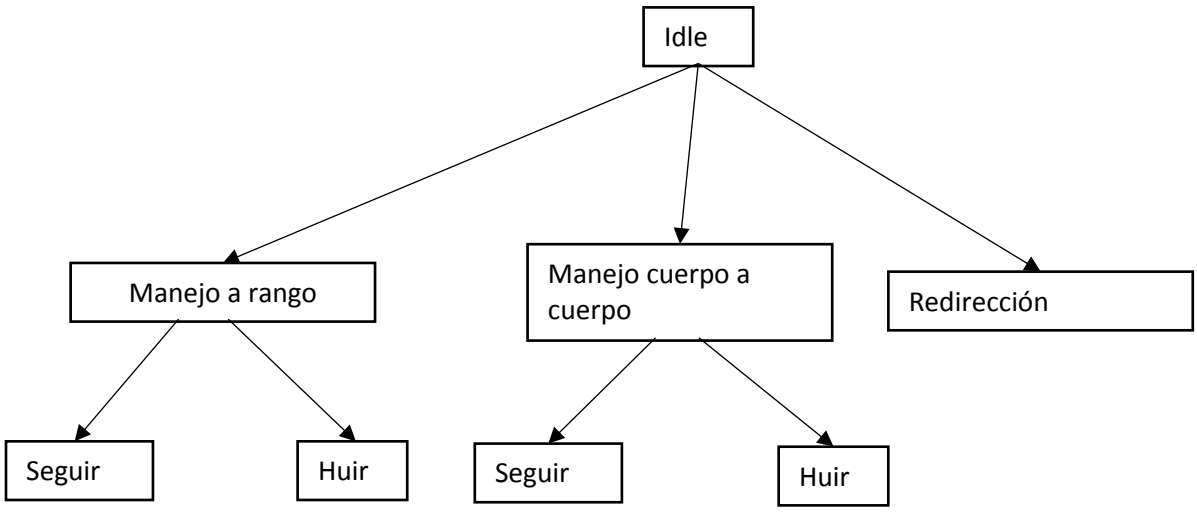

Figura 24: Árbol de comportamiento

En el nivel dos del árbol se lleva a cabo el cálculo de posicionamiento del bot, debido a que es la primera condición a evaluar cuando se enfrente al personaje principal, en el nivel tres se tiene como condición el estado de salud. Cada uno de estos nodos fue incluido como un método en el controlador de personajes enemigos.

Como es necesario tener entradas que sean verificables y accionen el árbol de comportamiento, se programó una nueva clase que sirva como portadora de información al personaje, esta se encarga de monitorear la posición del jugador y actuar como efector para el *bot*.

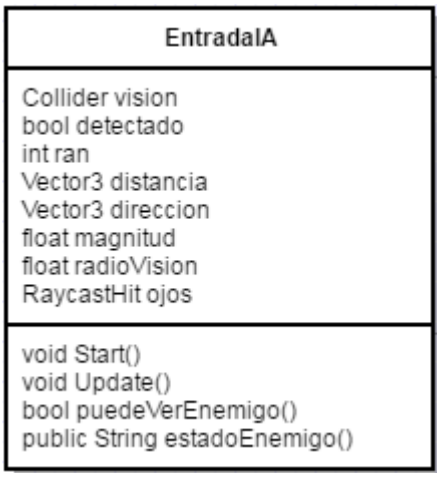

Figura 25: Clase de entradas para la IA

Para hacer más realista el comportamiento, se utilizó un vector posición como referencia de los ojos del personaje, y representan la visión del personaje, no pudiendo detectar al jugador en caso de estar oculto tras algún obstáculo.

La clase manejadora habilidades complementa a esta como procesadoras de entradas al árbol de comportamiento, evaluando tanto la ubicación de ambas entidades en el entorno como la capacidad de sus habilidades, dando como resultado un *bot* que puede interactuar de una manera adecuada según los recursos que tiene a su disposición.

Verificación de tarea:

Se realizó la misma prueba que en la tarea anterior, pero esta vez pudiendo corroborar las funciones del personaje enemigo gracias a la implementación de los métodos necesarios, el personaje efectivamente realizó las acciones correctamente, se crearon una serie de habilidades de prueba y se hizo una evaluación del orden de ejecución de estas según el algoritmo elaborado, el *bot* siguió correctamente este orden. El tercer nivel del árbol de comportamiento aún no pudo ser probado debido a la falta de los manejadores de salud de personajes.

Selección de tarea:

“Crear manejadores de salud”, Para concluir con el sistema de enfrentamientos solo queda incluir los controles de vida o salud para los personajes.

Ejecución de tarea:

Se elaboró una clase para el manejo en general de la salud de los personajes, de tal manera que en la siguiente tarea se pudieran usar los mismos métodos para cualquier tipo de personaje y facilitar así el trabajo. Esta clase se encarga también de trasladar información, sobre el estado actual de salud, al controlador de personajes enemigo, completando así el funcionamiento del árbol de comportamiento.

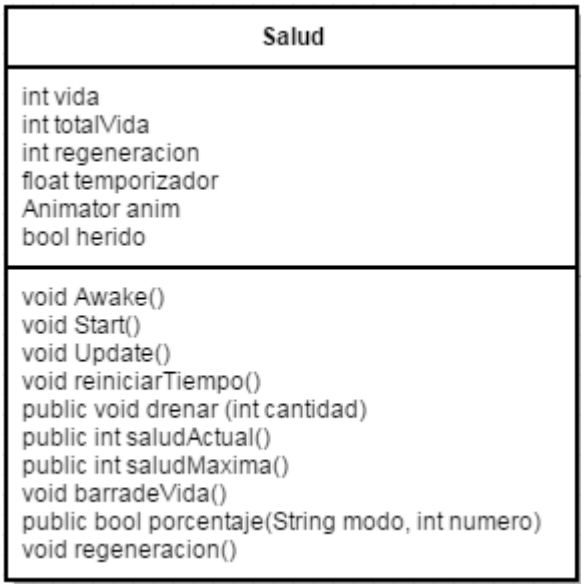

Figura 26: clase controladora de salud

Verificación de tarea:

Se inició una vez más la prueba con el personaje principal y un personaje enemigo, reduciendo y aumentando su salud a través del inspector de Unity, el personaje realizó correctamente las acciones correspondientes al nivel 3 del árbol de comportamiento.

Selección de tarea:

“Programar estados de combate”, para finalizar con las mecánicas del videojuego, solo resta programar las consecuencias de las acciones de los personajes en las situaciones de combate, que serían, reducción de vida y cambio de actitud.

Ejecución de tarea:

A este punto se cuenta con un manejador de vida que puede ser instanciado en cada personaje activo en el juego, se modificó el método de acción de las habilidades para que aplicaran el daño establecido a la salud del objetivo en caso de que sea exitoso, para conocer el éxito o fracaso de una habilidad se usó el radio del arma, en caso de que la habilidad marque el arma como necesaria, de lo contrario el daño se aplicaría automáticamente, para un tercer caso especial habría que escribir código para esta función en específico.

Para un mejor control del estado de combate del personaje principal se elaboró una nueva clase que pueda diferenciar entre el estado normal y el estado de combate, además de controlar las acciones de ataque, también se creó un método para que el jugador pueda salir del estado de combate, dando así más dinamismo a este aspecto del *gameplay*.

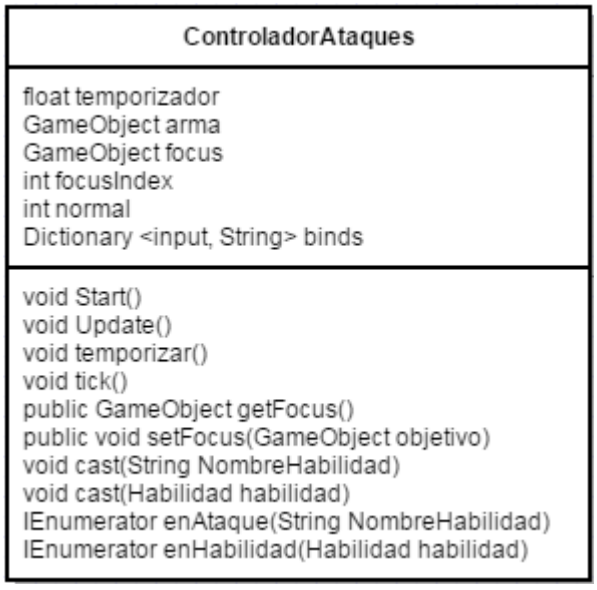

Figura 27: Clase controladora de ataques para el personaje principal.

Se crearon corutinas en esta clase que permiten tomar el espacio de tiempo de acción de las habilidades del personaje, en medio de este tiempo la clase puede verificar si esta habilidad tuvo éxito o fracaso, de esta manera, hay un mayor control al momento de la aplicación del daño.

Verificación de tarea:

Se agregaron instancias de la clase programada a cada personaje en el escenario del juego, se puso a prueba el estado de combate, con las diversas habilidades se aplicó el daño correcto a cada personaje.

SEGUIMIENTO DE LA ITERACIÓN

A continuación se describe como se realizó el monitoreo de la iteración en el transcurso de la misma.

Manejo de problemas:

Algunas tareas fueron concluidas sin haber sido testeadas al cien por ciento, ya que dependían de tareas posteriores, en las que las pruebas resultaron positivas, de igual forma, esto representó un riesgo en el control de las tareas para esta iteración en caso de que se hubiera encontrado un fallo.

Se realizaron varios casos de prueba para el control de la inteligencia artificial de los personajes, debido a la gran cantidad de posibilidades que hay al crear libremente distintas habilidades, proceso que tuvo que ser extendido a lo largo de la iteración debido a la influencia de las funcionalidades incorporadas en el transcurso de la misma.

Control del estado del proyecto a través de cada tarea

Tabla 14: estado de tareas sprint cuatro

Tarea	Descripción	Estado
1	Crear controlador de objetos	Concluida en el tiempo estipulado.
2	Implementar los métodos de interacción con los objetos	Concluida en el tiempo estipulado.
3	Programar manejador de habilidades para los personajes	Concluida en el tiempo estipulado.
4	Implementar mecanismos de toma de decisiones para los personajes enemigos	Concluida en el tiempo estipulado.
5	Crear manejadores de salud	Concluida en el tiempo estipulado.
6	Programar estados de combate	Concluida en el tiempo estipulado.

Registro de medidas:

A través de todas las pruebas que se hicieron en la verificación de cada tarea, se corroboró que cada proceso cumplió con las métricas previamente establecidas, obteniendo resultados altamente aceptables.

Cierre de la iteración:

Se presenta el estado actual del juego después de culminar el proceso del cuarto sprint, en donde se programó el sistema de enfrentamientos entre personajes.

Evaluación del estado del juego:

Se consiguió un producto con el *gameplay* completamente funcional, con la falta de la elaboración de un escenario y los modelos de los personajes no controlados por el jugador, a través de las pruebas se verificó que el juego cuenta con un

gameplay dinámico y entendible para los usuarios acostumbrados a jugar videojuegos de este género.

Evaluación de la iteración

En esta iteración había una carga de actividades con bastante complejidad, como lo es la elaboración de la inteligencia artificial de los personajes, estas actividades se realizaron en un buen orden, además de lograr crear pequeños módulos que se encargan de una actividad en específico, consiguiendo así un esquema bastante flexible y que puede estar sujeto a modificaciones posteriores de una manera fácil.

Actualización del plan de proyecto

Las tareas se cumplieron en su totalidad a la fecha establecida (30-07-2016) en el plan de proyecto, con la culminación de esta iteración se consigue un producto básico funcional, jugable y listo para ser sometido a pruebas, como se estableció en el tercer hito de la planificación.

3.3.5 Sprint cinco

Descripción: Construcción de escenario básico y recorrido de misiones.

Objetivos:

Conseguir un modelo 3D del entorno

Establecer los puntos clave para el recorrido de las misiones

Métricas:

Modelos 3D que estén conformes al concepto del videojuego

Mallas sin errores visibles

Al insertar los modelos en el escenario se debe preservar la fluidez de los gráficos

Seleccionar características:

Modelos de bajos polígonos

Recorrido de misiones de acuerdo a la historia planteada en el concepto del videojuego

Refinamiento de características:

Tabla 15: Refinamiento de características sprint cinco

N_o	**Descripción**	**Tareas**
1	Modelos de bajos polígonos	Elaborar idea conceptual del escenario Modelar el escenario
2	Recorrido de misiones de acuerdo a la historia planteada en el concepto del videojuego	Implementar diálogos y misiones en objetos de prueba.

DESARROLLO DE CARACTERÍSTICAS

Selección de tarea:

“Elaborar idea conceptual del escenario”, para poder trabajar con el modelado es necesario tener un concepto que sirva como especia de plano, para guiar el proceso de modelado 3D.

Ejecución de tarea:

A través de las ideas dadas por el concepto del juego, se elaboró un mapa que sirviera de guía para la elaboración de los modelos, se realizó a mano y posteriormente se llevó a digital.

Verificación de tarea:

Se comparó el mapa realizado con la idea original del juego, comprobando que todos los elementos añadidos en el son justificables a través del concepto del juego.

Selección de tarea:

“Modelar el escenario”, se procedió con la elaboración del escenario, usando como guía la idea elaborada anteriormente.

Ejecución de tarea:

Se utilizó el software Blender para el desarrollo de los modelos, se usaron técnicas sencillas que dieran resultados simples en cuanto al número de polígonos, además se omitieron detalles que se dejaron para que sean mostrados por las texturas, con el fin de lograr objetos más livianos y garantizar la fluidez de los gráficos

Verificación de tarea:

Una vez concluida la elaboración de los modelos, se exportó el modelo a Unity bajo el formato fbx, se insertó en un nuevo escenario y se le agregaron colisiones para evitar que los personajes traspasen la malla, para los objetos con una

cantidad considerable de polígonos se usaron colisiones más sencillas, tales como planos y cubos, con el fin de ahorrar el consumo de recursos.

Se inició la prueba en el escenario, consiguiendo altos *frames* por segundo, y además sin observar algún error en los modelos.

Selección de tarea:

“Implementar diálogos y misiones en objetos de prueba”, es posible ubicar las zonas donde se tomarán las cadenas de misiones, ya teniendo ubicaciones precisas dentro del entorno.

Ejecución de tarea:

Teniendo de antemano el transcurso de la historia del juego, se escribieron los diálogos de los *npc* y de las misiones, así como la ubicación de estos en el mapa, como aún no hay modelos de estos personajes se usaron elementos para referenciar la ubicación y poder probar el recorrido.

Verificación de tarea:

Se ejecutó el escenario y se cambiaron ciertas ubicaciones hasta coincidir con la experiencia de juego deseada, los diálogos parecen ser acordes pero estarán sujetos a cambios hasta la fase beta del juego.

SEGUIMIENTO DE LA ITERACIÓN

A continuación se describe como se realizó el monitoreo de la iteración en el transcurso de la misma.

Manejo de problemas:

En el transcurso del proceso de modelado ocurrieron varios errores en la malla, como era de esperarse, se trabajó con un estricto control de versiones para lograr afrontar estos problemas, evitando tener que recomenzar el proceso desde el inicio.

Algunos elementos importantes establecidos en el concepto del juego, como el desgaste de las estructuras, fueron dejados para el proceso de texturizado, previendo un posible exceso de polígonos en los modelos y garantizando siempre una tasa alta de *fps*.

Control del estado del proyecto a través de cada tarea

Tabla 16: Estado de tareas sprint cinco

Tarea	Descripción	Estado
1	Elaborar idea conceptual del escenario	Concluida en el tiempo estipulado.
2	Implementar los métodos de interacción con los objetos	Concluida en el tiempo estipulado.
3	Implementar diálogos y misiones en objetos de prueba.	Concluida en el tiempo estipulado.

Registro de medidas:

A través de todas las pruebas que se hicieron en la verificación de cada tarea, se corroboró que cada proceso cumplió con las métricas previamente establecidas, obteniendo resultados altamente aceptables.

Cierre de la iteración:

Se presenta el estado actual del juego después de culminar el proceso del quinto sprint, en donde se llevó a cabo el proceso de modelado del escenario.

Evaluación del estado del juego:

Se cuenta con un producto funcional, con mecánicas de juego completas y un escenario con texturas en blanco aún, además ya se puede vislumbrar el recorrido de los hechos a través de la línea de misiones.

Evaluación de la iteración

La quinta iteración se realizó sin algún inconveniente destacable, el videojuego se encuentra ya en una etapa avanzada. Los diálogos y misiones estarán sujetos a cambios hasta la finalización del juego.

Actualización del plan de proyecto

Las tareas se cumplieron en su totalidad a la fecha establecida (30-07-2016) en el plan de proyecto, se continúa sin alterar la estructura de este.

3.3.6 Sprint seis

Descripción: Modelado de la ambientación y decoración del entorno.

Objetivos:

Elaborar los modelos decorativos del entorno

Implementar los modelos elaborados en el escenario

Métricas:

Modelos 3D que estén conformes al concepto del videojuego

Mallas sin errores visibles

Al insertar los modelos en el escenario se debe preservar la fluidez de los gráficos

Seleccionar características:

Ambientación conforme al concepto del juego

Modelos de bajos polígonos

Refinamiento de características:

Tabla 17: Refinamiento de características sprint seis

N_o	Descripción	Tareas
1	Ambientación conforme al concepto del juego	Definir elementos necesarios y elaborar concepto.
2	Modelos de bajos polígonos	Crear modelos en 3D Implementar modelos en el escenario

DESARROLLO DE CARACTERÍSTICAS

Selección de tarea:

“Definir elementos necesarios y elaborar concepto”, es necesario definir cuáles son los pequeños elementos que estarán presentes en el escenario, dichos elementos no afectan de manera considerable el *gameplay*, pero ofrecen un gran atractivo visual y una sensación de inmersión al jugador

Ejecución de tarea:

Entre los objetos necesarios se propusieron, árboles, barriles, escombros, rocas, carteles, cercos, automóviles, antenas y objetos de la ciudad en general. Posteriormente se elaboraron ideas en papel del arte de estos objetos.

Verificación de tarea:

Cada una de las piezas de concepto fue evaluada en comparación con el concepto del videojuego, además se determinó que eran metas alcanzables en el trabajo de modelado y texturizado.

Selección de tarea:

“Crear modelos 3D”, Ya con el trazado de guía de las piezas decorativas fue posible comenzar con el proceso de modelado en Blender.

Ejecución de tarea:

Se realizaron los modelos en blender cuidando siempre la cantidad de polígonos, que en la mayor parte de los casos debieron ser muy pequeños en número de polígonos, puesto que ha mayor cantidad de decoración, mas carga en el proceso de renderizado en tiempo real a la hora de ejecutar el juego.

Verificación de tarea:

Se comparó cada modelo con las ideas conseguidas en la tarea anterior, entrando en el grado de aceptación deseado, los modelos son bastante ligeros y con caras bien definidas para el proceso de texturizado.

Selección de tarea:

“Implementar modelos en el escenario”, Ya teniendo todos los modelos se procedió a exportarlos al proyecto de Unity.

Ejecución de tarea:

Se exportaron los modelos en Blender en el formato FBX y fueron importados en el proyecto de Unity, se arrastraron en el escenario y fueron colocados de manera de que cada rincón de este estuviera cargado de una buena cantidad de detalles.

Verificación de tarea:

Se ejecutó el escenario y lo primero en evaluar fue la carga de gráficos en la herramienta *profiler* de Unity, se realizaron recorridos constantes para verificar que no hubiese picos muy altos en el uso de gpu.

SEGUIMIENTO DE LA ITERACIÓN

A continuación se describe como se realizó el monitoreo de la iteración en el transcurso de la misma.

Manejo de problemas:

A pesar de la variedad de elementos que fueron diseñados, al escenario ser tan extenso, ocasionaría una repetitividad notoria de los detalles del entorno, y como no se cuenta con un equipo numeroso para trabajar en lo que sería una gran cantidad de modelos, se decidió crear uno o dos modelos por objeto propuesto, y dejar la variedad para las texturas, consiguiendo así, objetos distintos con el mismo modelo.

No se encontraron problemas que afectaran el proceso de diseño y modelado, por lo que el flujo de trabajo fue rápido y exitoso.

Control del estado del proyecto a través de cada tarea

Tabla 18: Estado de tareas sprint seis

Tarea	Descripción	Estado
1	Definir elementos necesarios y elaborar concepto.	Concluida en el tiempo estipulado.
2	Crear modelos en 3D	Concluida en el tiempo estipulado.
3	Implementar modelos en el escenario	Concluida en el tiempo estipulado.

Registro de medidas:

A través de todas las pruebas que se hicieron en la verificación de cada tarea, se corroboró que cada proceso cumplió con las métricas previamente establecidas, obteniendo resultados altamente aceptables.

Cierre de la iteración:

Se presenta el estado actual del juego después de culminar el proceso del sexto sprint, en donde se llevó a cabo el proceso de modelado de los elementos decorativos del escenario.

Evaluación del estado del juego:

Se cuenta con un producto funcional, con mecánicas de juego completas y el escenario ya construido solo faltando las texturas del mismo.

Evaluación de la iteración

La sexta iteración se realizó sin algún inconveniente destacable, se han minimizado los costes de trabajo para el equipo de desarrollo con que se cuenta, en pro de alcanzar la meta establecida, el videojuego se encuentra en una etapa avanzada.

Actualización del plan de proyecto

Las tareas se cumplieron en su totalidad a la fecha establecida (13-08-2016) en el plan de proyecto, no se ha alterado su estructura y sigue en pie la consecución del cuarto hito para la fecha 26-08-2016.

3.3.7 Sprint siete

Descripción: Diseño e implementación de texturas en los modelos del entorno.

Objetivos:

Texturizar todo el escenario del videojuego

Implementar los *shaders* adecuados a cada material

Métricas:

Texturas sin defectos visibles.

Texturas de baja resolución para los elementos más pequeños y resolución media para los más grandes, hasta un máximo de 1024x1024.

Consistencia en el arte de todas las texturas.

Seleccionar características:

Elaboración y uso de texturas semi realistas.

Implementación de *shaders*.

Refinamiento de características:

Tabla 19: Refinamiento de características sprint siete

N_o	**Descripción**	**Tareas**
1	Elaboración y uso de texturas semi realistas.	Elaborar mapeado UV de los modelos. Diseñar e implementar texturas en los modelos.
2	Implementación de *shaders*.	Asignar *shaders* a los materiales creados en Unity

DESARROLLO DE CARACTERÍSTICAS

Selección de tarea:

“Elaborar mapeado UV de los modelos.”, para poder asignar las texturas correctamente a los modelos, es necesario generar los mapas UV de cada uno, requisito indispensable para un correcto renderizado en tiempo real

Ejecución de tarea:

Se crearon los mapas UV de los modelos del escenario en Blender, y posteriormente exportados como imagen a Photoshop para el posterior diseño de las texturas, se definió para cada uno la resolución que debería tener, para los detalles de entorno a la imagen más grande se le asignó una resolución de 512x512, y a la imagen más pequeña 128x128. Para los modelos más grandes como edificios y terrenos, se crearon de hasta un máximo de 1024x1024.

Verificación de tarea:

Se verificó el orden de los elementos en cada imagen de mapa UV, para evitar errores al momento de la implementación en el motor, además de la resolución en la que se exportó cada uno fue revisada.

Selección de tarea:

“Diseñar e implementar texturas en los modelos”, se inició el trabajo de texturizado en el software Photoshop.

Ejecución de tarea:

Se usaron algunas texturas libres encontradas, aplicadas encima de la capa de mapas UV, otras fueron creadas directamente en Photoshop, todas con un estilo semi realista.

Se generaron para todas las texturas el mapeado normal y para las mas importantes se crearon otros mapas adicionales como el de oclusión y el de desplazamiento, para dar tonos más realistas a algunos elementos, que ocuparan un gran espacio.

En este mismo proceso también se crearon *decals* para enriquecer el nivel de detalle del escenario.

Finalmente se exportaron las imágenes en formato PNG y fueron llevadas al proyecto de Unity.

Verificación de tarea:

Se crearon los materiales en el proyecto, todos con el standard *shader* por defecto, y se aplicaron a los modelos para verificar que no hubieran errores en la textura, como distorsiones o puntos blancos.

Selección de tarea:

“Asignar *shaders* a los materiales creados en Unity”, se cambiaron y configuraron los shaders para los materiales.

Ejecución de tarea:

Se cambiaron los parámetros de los materiales con *standard shader*, como suavidad y metal, hasta lograr el efecto deseado, para los materiales de los personajes se usaron otros *shaders* libres disponibles en Unity.

Se agregaron los mapas normales y los demás mapas secundarios a los materiales, y además se implementaron algunos *shaders* con animaciones en los vértices para materiales como el fuego y el vapor.

Se introdujeron los *decals* en distintas partes del escenario, agregando nuevos materiales y sobreponiéndolos a los materiales bases de los modelos, todos los *decals* usan *standard shader*.

Verificación de tarea:

Se ejecutó el escenario y se evaluó nuevamente el consumo de gpu, así como nuevas verificaciones en el mapeado de las texturas, consiguiendo resultados altamente aceptables.

SEGUIMIENTO DE LA ITERACIÓN

A continuación se describe como se realizó el monitoreo de la iteración en el transcurso de la misma.

Manejo de problemas:

Se redujo la carga de trabajo usando librerías de texturas libres, debido a la gran cantidad de materiales que debían generarse, siendo esta una labor muy extensa para el equipo de desarrollo con que se cuenta.

Se usaron algunos mapas normales en múltiples ocasiones sobre algunas texturas creadas con el fin de lograr consistencia en los diversos materiales.

Se monitoreó a través del *profiler* de Unity la carga en gpu, consiguiendo buenos resultados en un computador de gama media.

Control del estado del proyecto a través de cada tarea

Tabla 20: Estado de tareas sprint siete

Tarea	Descripción	Estado
1	Elaborar mapeado UV de los modelos	Concluida en el tiempo estipulado.
2	Diseñar e implementar texturas en los modelos	Concluida en el tiempo estipulado.
3	Asignar *shaders* a los materiales creados en Unity	Concluida en el tiempo estipulado.

Registro de medidas:

A través de todas las pruebas que se hicieron en la verificación de cada tarea, se corroboró que cada proceso cumplió con las métricas previamente establecidas, obteniendo resultados altamente aceptables.

CIERRE DE LA ITERACIÓN

Se presenta el estado actual del juego después de culminar el proceso del sexto sprint, en donde se implementaron las texturas de los modelos 3D.

Evaluación del estado del juego:

Se cuenta con un producto funcional, con mecánicas de juego completas, escenario modelado y texturizado y con gráficos ya casi definitivos.

Evaluación de la iteración

La septima iteración se realizó sin algún inconveniente destacable, se usaron herramientas libres para reducir la carga de trabajo, y se consiguieron gráficos bastante aceptables, tomando en cuenta de que se desarrolló un videojuego *indie*.

Actualización del plan de proyecto

Las tareas se cumplieron en su totalidad a la fecha establecida (26-08-2016) en el plan de proyecto, no se ha alterado su estructura y con la conclusión de esta iteración se logra el cuarto hito establecido en la planificación, en el cual se quería tener un escenario completamente desarrollado.

3.3.8 Sprint ocho

Descripción: Diseño y mapeado de iluminación.

Objetivos:

Realizar el proceso de cálculo de iluminación en el escenario

Métricas:

Sombras sin defectos o errores visibles.

Iluminación global fuerte.

Mantener bajo consumo de gpu.

Escenario sin rincones carentes de iluminación.

Seleccionar características:

Iluminación del escenario

Refinamiento de características:

Tabla 21: Refinamiento de características sprint ocho

Número	Descripción	Tareas
1	Iluminación del escenario	Insertar luces dentro del escenario y realizar *bake*

DESARROLLO DE CARACTERÍSTICAS

Selección de tarea:

“Insertar luces dentro del escenario y realizar *bake*”, con todos los modelos puestos en el escenario, ya solo queda realizar el cálculo de iluminación en escenario.

Ejecución de tarea:

Se agregaron las luces a la escena, empezando por la luz global, la cual debía ser fuerte y dejar sombras a lo largo del entorno, simulando el sol, la mayoría de los objetos del ambiente fueron marcados como iluminación estática, esto en favor del rendimiento gráfico del juego, otras luces menores fueron agregadas a lo largo de la escena, para exaltar elementos como el fuego o elementos con iluminación propia, como lámparas y postes de luz. Una vez ubicadas todas las luces deseadas se comenzó con el proceso de *bake*.

Verificación de tarea:

Luego de concluír el *bake*, que llevó aproximadamente siete horas, se observó el resultado y se repitió el proceso hasta conseguir el resultado esperado.

En el proceso de verificación se cambiaron varios elementos de estáticos a dinámicos y viceversa, tomando en cuenta el factor *gameplay* y de cómo convenía más establecer el cálculo de iluminación en dicha zona, se cambiaron los colores, la intensidad, y el rango de las luces hasta lograr el efecto deseado.

Luego de conseguir el efecto que más se adaptó al concepto, se ejecutó la escena y se monitoreó el rendimiento con la herramienta *profiler* de Unity, obteniendo resultados aceptables.

SEGUIMIENTO DE LA ITERACIÓN

A continuación se describe como se realizó el monitoreo de la iteración en el transcurso de la misma.

Manejo de problemas:

Algunos tipos de luces como la luz de punto dieron inconvenientes en algunos lugares, lo cual al final se determinó era un error en el motor, por lo que se tuvo que descargar una versión posterior en donde se corregía el problema.

Control del estado del proyecto a través de cada tarea

Tabla 22: Estado de tareas sprint ocho

Tarea	Descripción	Estado
1	Iluminación del escenario	Concluida en el tiempo estipulado.

Registro de medidas:

A través de todas las pruebas que se hicieron en la verificación de cada tarea, se corroboró que cada proceso cumplió con las métricas previamente establecidas, obteniendo resultados altamente aceptables.

CIERRE DE LA ITERACIÓN

Se presenta el estado actual del juego después de culminar el proceso del octavo sprint, en el cual se estableció la iluminación del escenario.

Evaluación del estado del juego:

Se cuenta con un juego ya funcional, a solo falta de la interfaz de usuario.

Evaluación de la iteración

La octava iteración se completó en el tiempo establecido, se finalizó el proceso de elaboración del escenario y ya es posible jugarlo y probarlo en su totalidad bajo el editor de Unity.

Actualización del plan de proyecto

Las tareas se cumplieron en su totalidad a la fecha establecida (10-09-2016) en el plan de proyecto, no se ha alterado su estructura y con la conclusión de esta iteración se logra el quinto hito dado en la planificación, teniendo una escena totalmente finalizada, solo sujeta a cambios en la fase beta.

3.3.9 Sprint nueve

Descripción: Diseño e implementación de la interfaz de usuario.

Objetivos:

Conseguir toda a interfaz de usuario del juego

Métricas:

Elementos de interfaz conforme al concepto del juego.

Interfaz rápida e intuitiva.

Seleccionar características:

Interfaz del menú principal del juego

Interfaz dentro del juego

Carga de partidas previas

Refinamiento de características:

Tabla 23: Refinamiento de características sprint nueve

N_o	Descripción	Tareas
1	Interfaz del menú principal del juego	Definir elementos del menú de juego Desarrollar menú principal
2	Interfaz dentro del juego	Definir elementos de interfaz necesarios dentro del juego Desarrollar interfaz dentro del juego
3	Carga de partidas previas	Función de guardado y carga de partidas.

DESARROLLO DE CARACTERÍSTICAS

Selección de tarea:

“Definir elementos del menú de juego”, para comenzar con el desarrollo del menú principal en necesaria definir cuáles son las funciones que podrá realizar el usuario en este punto.

Ejecución de tarea:

En el menú inicial el usuario debe poder realizar ciertas acciones, la principal sería iniciar con el juego, puede también ver alguna información de ayuda, opción útil en caso de que sea la primera vez que juegue, puede ver los créditos y cargar una partida guardada previamente, en el recorrido de las pantallas además, debe estar la opción de regresar al menú inicial. Siendo un total de cuatro opciones que puede realizar el usuario, por lo tanto el trabajo de diseño de interfaz queda esquematizado en cinco pantallas y seis botones, incluyendo el botón de salir.

Verificación de tarea:

Se revisó el esquema de interfaz varias veces, siendo este el más adecuado para llevar a cabo.

Selección de tarea:

“Desarrollar menú principal”, Se comenzó a elaborar los elementos del menú principal, así como su implementación en el motor Unity.

Ejecución de tarea:

Se realizaron las imágenes de los botones y las pantallas en el software Photoshop, y fueron exportados como imágenes a Unity, aquí se programó la funcionalidad de cada uno de estos elementos, exceptuando el de cargar partida que se realizará en una tarea posterior, se manejaron los controles de interfaz de Unity, como lo son los *canvas*, *buttons*, *rect*, entre otros.

Verificación de tarea:

Ya con el menú principal fue posible crear una versión ejecutable del juego y ponerse a prueba, el menú principal funcionó correctamente y su diseño se integra adecuadamente al concepto del juego.

Selección de tarea:

“Definir elementos de interfaz necesarios dentro del juego”, otros elementos de interfaz son necesarios dentro del juego, para ello primero se debió definir cuáles eran los necesarios para el usuario.

Ejecución de tarea:

Como el juego es de modalidad de un solo jugador, es posible usar la opción de pausa, esta misma opción puede llevar a un menú de opciones en el que el usuario puede salir del juego o continuar. Aparte de este menú el usuario debería disponer de una interfaz de ayuda para el seguimiento de las misiones que realiza, de esa manera poder visualizar la descripción y el progreso de la misma, en el momento que lo desee. Para ello se decidió incluir un botón dentro de la pantalla de juego, en el cual se puede acceder a un registro de misiones, donde se visualiza en forma textual la descripción y el progreso de cada misión disponible.

Verificación de tarea:

Se evaluó las posibles acciones que debería poder realizar el usuario a nivel de interfaz y se concluyó con que estas son todas las necesarias, puesto que no hay alguna otra funcionalidad que lo amerite.

Selección de tarea:

“Desarrollar interfaz dentro del juego”, se procedió con el diseño y la programación de las funciones de interfaz definidas en la tarea anterior.

Ejecución de tarea:

Se diseñó en Photoshop el botón de las misiones, y los botones correspondientes al menú de pausa, así como la pantalla de interfaz de las misiones.

Se exportaron las imágenes a Unity y fueron programadas sus funcionalidades dentro del controlador de entradas del usuario.

Verificación de tarea:

Se creó una nueva versión ejecutable del juego y se puso en marcha, comprobando que cada uno de los nuevos elementos de la interfaz funcionan correctamente.

Selección de tarea:

“Carga de partidas previas”, ya con la interfaz de juego completada, se pasó a la funcionalidad de guardar y cargar partidas.

Ejecución de tarea:

Se creó un nuevo archivo en formato Json destinado a los datos de la partida, estos datos serían posición actual del jugador, salud y progreso de misiones. Se agregó un nuevo botón dentro de la pantalla de juego con la opción de guardar y se programó su funcionalidad, sobrescribiendo los datos en el archivo.

Se programó la función de cargar partida y se asignó al botón de cargar partida del menú inicial del juego, esta función envía los datos cargados desde el archivo Json como parámetros al iniciar una nueva partida, ubicando al jugador en el punto donde se encontraba al momento de guardar y con el mismo progreso.

Verificación de tarea:

Se probó un nuevo ejecutable del juego, se realizaron misiones y se guardó la partida, posteriormente se cargó esta partida guardada y los datos fueron recuperados tal cual como se habían guardado.

SEGUIMIENTO DE LA ITERACIÓN

A continuación se describe como se realizó el monitoreo de la iteración en el transcurso de la misma.

Manejo de problemas:

Para la función de guardar partida se tomó un tiempo considerable en tomar los elementos necesarios para que el jugador pueda reiniciar el juego en el estado donde se encontraba, pero sin extenderse del periodo de tiempo planificado.

En el diseño de interfaz de usuario no se presentaron problemas que pudieran afectar el proceso de elaboración.

Control del estado del proyecto a través de cada tarea

Tabla 24: Estado de tareas sprint nueve

Tarea	**Descripción**	**Estado**
1	Definir elementos del menú de juego	Concluida en el tiempo estipulado.
2	Desarrollar menú principal	Concluida en el tiempo estipulado.
3	Definir elementos de interfaz necesarios dentro del juego	Concluida en el tiempo estipulado.
4	Desarrollar interfaz dentro del juego	Concluida en el tiempo estipulado.
5	Función de guardado y carga de partidas	Concluida en el tiempo estipulado.

Registro de medidas:

A través de todas las pruebas que se hicieron en la verificación de cada tarea, se corroboró que cada proceso cumplió con las métricas previamente establecidas, obteniendo resultados altamente aceptables.

CIERRE DE LA ITERACIÓN

Se presenta el estado actual del juego después de culminar el proceso del último sprint, en el cual se generó la interfaz de usuario y la opción de guardado.

Evaluación del estado del juego:

A este punto el juego ya puede pasar a fase beta para ser probado, todas las funcionalidades están desarrolladas.

Evaluación de la iteración

La novena iteración se desarrolló con normalidad, dejando el videojuego listo para entrar a fase beta.

Actualización del plan de proyecto

Las tareas se cumplieron en su totalidad a la fecha establecida (26-09-2016) en el plan de proyecto, no se ha alterado su estructura y con la conclusión de esta iteración se logra el sexto hito dado en la planificación, consiguiendo un producto listo para pasar a la fase beta.

3.4 FASE 4 BETA

3.4.1 Planificación de la primera iteración

A continuación se describe el proceso planificado para la primera iteración de la fase beta del proyecto.

Aspectos funcionales y no funcionales

El videojuego tiene todas las funcionalidades programadas, el usuario puede realizar todas las acciones que naturalmente podría realizar.

Medios de distribución

El medio de distribución es computadoras personales con Windows 7 o superior, aún no se determinaba cuáles eran los requisitos mínimos reales del computador para poder ejecutar el videojuego, sobre todo en gpu, cpu, y memoria ram, así que se inició con la siguiente configuración mínima estimada:

Tabla 25: Estimación de requisitos mínimos

Recurso	Mínimo
CPU	Intel Pentium dual core E214 o equivalente.
GPU	Con DirectX 9 y con 512MB de memoria RAM
RAM	2 GB

Mecanismos de reporte de errores

Como no se cuenta con una plataforma web dedicada al proyecto para manejar el *feedback* de los usuarios, el reporte se estableció a través de correo electrónico.

Equipo de verificadores beta

Se seleccionó un equipo de verificadores beta, personas conocidas con dedicación a este tipo de juegos, que dieran un punto de vista desde la perspectiva de un *gamer*. El equipo se conformó por tres personas.

3.4.1.1 Distribución de versión beta

Se describen a continuación los métodos de comunicación con el equipo verificador y la distribución del producto.

Medios de comunicación.

Se utilizó una plataforma de propósito general como los son las redes sociales, para mantener contacto directo con los verificadores, nuevamente por no tener una plataforma web dedicada al proyecto.

Aspectos a verificar.

Se establecieron los aspectos más importantes a verificar para que los *testers* se enfocaran más en estos puntos, dichos aspectos son, gameplay, gráficos y grado de entretenimiento.

3.4.1.2 Verificación

Evaluación y verificación.

Se evaluó el reporte de los *testers* recibiendo resultados satisfactorios en sus comentarios, en los tres aspectos mencionados anteriormente. El videojuego tuvo buen rendimiento en todas las computadoras que fue probado, las cuales eran superiores al mínimo establecido.

Se reportaron algunos errores que ocurrieron en repetidas ocasiones sobre el salto del personaje, en donde este saltaba varias veces cuando debía hacerlo solo una única vez.

3.4.1.3 Correcciones

Luego de recibir el reporte del error se realizaron varios casos de prueba en el proyecto de Unity y se determinó la causa del error, en el cual después de saltar algún obstáculo el controlador enviaba información errónea a la máquina de estado.

Se solucionó el inconveniente y se realizaron nuevos casos de prueba hasta comprobar que el error había desaparecido totalmente.

3.4.2 Planificación de la segunda iteración

A continuación se describe el proceso planificado para la segunda iteración de la fase beta del proyecto.

Aspectos funcionales y no funcionales

El videojuego tiene los mismos aspectos funcionales que en la iteración anterior, ahora con la corrección del error reportado por los jugadores.

Medios de distribución

Se utilizó el mismo medio de distribución que en la iteración anterior, sin modificaciones a los requerimientos mínimos en prestaciones de hardware.

Mecanismos de reporte de errores

Se utilizaron los mismos medios de distribución que en la iteración pasada.

3.4.2.1 Distribución de versión beta

Se describen a continuación los métodos de comunicación con el equipo verificador y la distribución de la segunda versión beta.

Medios de comunicación.

Se continuaron usando las redes sociales para mantener una comunicación constante con el equipo de verificadores.

Aspectos a verificar

Luego de haber despejado dudas sobre los puntos iniciales el principal objetivo fue verificar el problema ocasionado con los saltos del personaje, además de la búsqueda de algún otro error relevante.

3.4.2.2 Verificación

Evaluación y verificación.

Se evaluó el resultado de los reportes de los *testers*, aconsejaron varias mejoras en el estilo de juego y en la interfaz de usuario, sugiriendo una interfaz más acorde al concepto del juego y en cuanto a *gameplay* movimientos más rápidos, sobre todo el de retroceso. No se reportaron nuevos errores en las pruebas que realizaron.

3.4.2.3 Correcciones

Se decidieron que recomendaciones tomar en cuenta y cuáles no, se decidió retocar la estética de la interfaz de usuario, sin ningún cambio en su funcionalidad. El *gameplay* funciona según lo planeado, y se dejó tal y como está trabajando actualmente, por lo que se omitió esta sugerencia hecha por uno de los *testers.*

3.4.3 Planificación de la tercera iteración

A continuación se describe el proceso planificado para la tercera iteración de la fase beta del proyecto.

Aspectos funcionales y no funcionales

El videojuego se considera ya culminado con esta fase, y se decidió realizar solo muestras de la nueva interfaz para recibir la opinión de los usuarios.

Equipo de verificadores beta

Las imágenes fueron enviadas al mismo grupo de personas que conformaron desde el primer momento el equipo de verificadores.

3.4.3.1 Distribución de versión beta

Se describen a continuación los métodos de comunicación con el equipo verificador y la distribución del producto.

Medios de comunicación.

Se continuaron utilizando los mismos canales de comunicación de las iteraciones anteriores.

Aspectos a verificar.

Se buscó obtener la opinión de los verificadores sobre los cambios en la interfaz de usuario del videojuego, comparándola con la anterior, y con el objetivo de mejorar los aspectos que habían puntualizado.

3.4.3.2 Verificación

Evaluación y verificación.

Se evaluó la opinión dada por los usuarios, quienes dieron un visto positivo a los nuevos cambios. Concluyendo así el proceso de verificación beta, y terminando la elaboración del videojuego a este punto.

CONCLUSIONES

La metodología de desarrollo SUM basada en Scrum permitió un desarrollo bastante ágil y organizado de la aplicación, ahorrando el proceso de tener que crear una adaptación personal de Scrum para la iniciación del proyecto, además de poder adaptar el aprendizaje adquirido al final del proyecto para mejorar el proceso, adecuándolo a la capacidad del equipo de desarrollo. Esto se aprecia muy útil por el hecho de que el desarrollo de videojuegos conlleva características que no están presentes en la elaboración de software convencional, como lo es el arte en general, desde la pintura hasta la escritura y en las metodologías de desarrollo de software no hay espacio definido para estas otras disciplinas.

La consecución de productos al final de cada iteración permitió una elaboración sistemática del juego, organizando componentes y funciones, que facilitan el proceso de edición de la aplicación a la hora de tener que retroceder a realizar cambios en las funcionalidades. También facilita la integración de los componentes del juego, que puede volverse compleja debido a la interacción constante de todos los elementos, desde el mismo usuario hasta los personajes y decisiones que se toman dentro del *gameplay*.

La fase de concepto promueve el emprendimiento del proyecto a partir una idea, más que con una obligación de realizar un videojuego con una motivación vacía, lo que puede contribuir al éxito del mismo, en caso de terminar siendo un producto comercial. La misma etapa hace posible adaptar y organizar el flujo de trabajo a partir de la idea inicial.

Aunque el equipo de desarrollo estuvo conformado por una única persona y es difícil obtener una retroalimentación constante con otro grupo de personas durante el proceso de desarrollo, la fase beta fue muy útil para cubrir esta carencia, no funcionando solo para la depuración del código, sino también para cubrir aspectos mucho más subjetivos como el grado de diversión del usuario, o la capacidad de comprender la trama y el ambiente del videojuego.

El motor de juego Unity ofreció una curva de aprendizaje considerablemente empinada tomando en cuenta la cantidad de conocimiento adquirido a través del tiempo, algo que no se experimentó con tanto énfasis en otros softwares. Un motor de juego constituye una herramienta poderosa para el desarrollo de aplicaciones, que pueden ir más allá incluso de un videojuego.

A pesar de lo dicho sobre la facilidad de aprendizaje que ofrece Unity, se pudo experimentar cierto punto de quiebre a mitad del desarrollo, en donde la estructura del proyecto se va volviendo más extensa, y el control de la misma depende ya casi en totalidad del equipo de desarrollo, pudiendo representar un grave riesgo para equipos sin experiencia en el desarrollo de videojuegos, algo que no sucede cuando se usa un framework para estructurar el código de la aplicación, en este sentido, la adquisición de un framework dentro de Unity conlleva a la disposición de un presupuesto para el gasto en este tipo de recursos, disponibles en la *asset store* de Unity, que comprende el ancho del modelo de negocios de este motor. Es posible, que, para equipos sin experiencia en proyectos de complejidad media o más elevada, convenga usar otros motores de juego, que disponen de herramientas que ayudan más al usuario en la estructuración de funcionalidades, un ejemplo de estas herramientas son los frameworks de programación nodal, también disponibles dentro de Unity de forma paga.

Algunas herramientas a disposición de Unity contienen documentación vaga o defectos que a la larga pueden ocasionar inconvenientes, en el transcurso de la elaboración de este proyecto, uno de estos inconvenientes fue el manejo de la herramienta de *pathfinding* del motor, la cual puede escalar alto en el consumo de recursos cuando se ubica un número alto de agentes dentro del escenario.

El empaquetado final del videojuego en Unity es bastante ligero en comparación a la de otros motores, aún cuando es necesario conocer sobre optimización de memoria, funcionamiento del gpu, lectura de *profiler*, entre otros, para lograr un producto ligero.

El lenguaje C# permitió una escritura ágil del código, proporcionando librerías que facilitaron en gran medida la elaboración de los algoritmos necesarios para las funcionalidades del videojuego, en conjunto también, con la api de Unity, sin tener que especificar demasiado en manejo de estructuras de datos como las listas o en la manipulación de funciones matemáticas.

RECOMENDACIONES

El producto conseguido puede ser usado como framework para la elaboración de otros videojuegos de la categoría *rpg*, aventura, acción y a fines, gracias a que, cuenta con una api con funciones establecidas que se pueden reutilizar en otro proyecto.

Promover el desarrollo de videojuegos en la Universidad de Oriente como área de investigación.

BIBLIOGRAFÍA

Geymonat, n. (2014), *El videojuego en seis escuelas de tiempo completo: puente entre lo sociocultural y lo didáctico pedagógico*. Universidad ORT Uruguay, Montevideo, Uruguay.

Frasca, g. (2010), *Juego, videojuego y creación de sentido: una introducción*. Plurais, Revista multidisciplinaria. Bahia, Brasil. Extraído de: <http://www.revistas.uneb.br/index.php/plurais/article/view/879/623>

Benito, j. (2005), *El mercado del videojuego: unas cifras.*Revista ICONO 14, 7. Madrid, España.

Etxeberria, f. (2008), *Videojuegos, consumo y educación.* pedagogía social. revista interuniversitaria. Universidad del País Vasco, País Vasco, España.

De Aguilera, m. (2004), *La institucionalización de una industria cultural. Estructura y desafíos de la industria de los videojuegos*. Revista Telos. Maracaibo. Extraído:<https://telos.fundaciontelefonica.com/telos/articuloperspectiva.asp@idarticulo=3&rev=59.>

Thursten, c. (2015), *The international 2015: everything you need to know*. Revista PC Gamer, San Francisco. Extraído de: <http://www.pcgamer.com/the-international-2015-everything-you-need-to-know/>.

Espinoza, m. (2009), *Desarrollo de juego educativo rpg para teléfonos móviles* (tesis de pregrado). Universidad de Chile, Santiago de Chile, Chile.

Pérez, d; Pérez, d. (2014) Un conjunto de herramientas para Unity orientado al desarrollo de videojuegos de acción-aventura y estilo retro con gráficos isométricos 3D (tesis de pregrado). Universidad complutense de Madrid, Madrid, España.

Fuentes, a. (2014) *Desarrollo de un juego de aventuras en C# sobre Unity* (tesis de pregrado). Universidad politécnica de Valencia, Valencia, España.

Palma, s; Alcobé j. (2009) *Desarrollo de un juego de rol con RPGVX* (tesis de pregrado). Escuela politécnica universitaria de Mataró, Barcelona, España.

Arce, l. (2011) *Desarrollo de videojuegos (*tesis de pregrado*)*. Universidad del Aconcagua, Mendoza, Argentina.

Giraldo, i; Delgado, e; Castellanos, g. (2006) *Cinemática Inversa de un Brazo Robot Utilizando Algoritmos Genéticos* (tesis de pregrado). Universidad Nacional de Colombia, Medellín, Colombia.

Bertín, c; Bidart, s; Gorigoitía, p; Kierkegaard, s; Otero, e; Bidart, s; Cerda, g; Kierkegaard, p; Mendiburo, a; Uman, v; Valencia, n. (2011) *Guerra y paz en el mundo virtual*. Universidad Uniacc, Santiago de Chile, Chile.

Andagoya, s. (2016) *Desarrollo del modelado, animación y texturizado de los diferentes actores y escenarios que intervienen en el videojuego "LLUMPAK"* (tesis de pregrado). Universidad del Ecuador, Quito, Ecuador.

Craighead, j; Burke, j; Murphy, r. (2007) *Using the unity game engine to develop sarge: a case study*. Computer. Extraído de: <http://www.robot.uji.es/research/events/iros08/contributions/Craighead.pdf>.

Hejlsberg, a; Wiltamuth, s; Golde, p. (2003). *C# language specification*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

Akenine-Möller, t; Haines, e; Hoffman, n. (2008). *Real-time rendering*. CRC Press. Natick, MA, USA.

Cuevas, d. (2013) *Análisis de Algoritmos Pathfinding* (tesis de pregrado). Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile.

APÉNDICES

Plan de Proyecto

<versión 1>

Proyecto

<Videojuego “Drones” >

Introducción

A continuación se describe los aspectos de la planificación del desarrollo del videojuego, según lo obtenido al finalizar la segunda fase de la metodología.

Plan de Personal

Para este proyecto el equipo de desarrollo estuvo conformado por una única persona que tomó todos los roles y subroles establecidos por la metodología, exceptuando la fase beta donde por supuesto varias personas tomaron el papel de verificadores beta y enviaron su respectivo *feedback*.

También hubo consultas para verificar el concepto del juego, en la que varias personas voluntarias dieron su opinión acerca de la historia narrada.

Modelo de negocios

Público objetivo:

El juego está orientado principalmente a jugadores asiduos de videojuegos (*gamers*), entusiastas de los juegos con perspectiva de tercera persona, que en determinado momento busquen una forma de entretenimiento rápida, sin dejar de considerar a cualquier otro tipo de jugador mas casual. Se estipula este objetivo debido a que este género es mas solicitado por personas que tienen el ocio electrónico como costumbre, por lo tanto, se debe aceptar este tipo de público como los que más curiosidad tendrán por probarlo.

Los desarrolladores *indies* siempre están interesados en conocer los proyectos homólogos, ya sea para evaluar competencia, medir capacidades, estudiar el mercado, simple entretenimiento, entre otros. Por lo tanto este proyecto al ser *indie*, por omisión será objetivo de este tipo de personas.

Los apasionados del género ciencia ficción también se esperaría sientan curiosidad de sumergirse en la narrativa que ofrece el videojuego, este género literario ha tenido gran éxito en la industria, y son apoyados por una numerosa comunidad.

Modelo de negocio a seguir:

El producto final estará enmarcado en el modelo de negocios *free to play*, el cual permite a los jugadores descargar y jugar sin necesidad de realizar algún tipo de pago. Este modelo de negocio es la contraparte de *pay to play* en donde se solicita un pago previo para poder obtener el videojuego.

El modelo *free to play* da la libertad al equipo de desarrollo implementar y explorar una gran variedad de métodos para recibir contribución económica, en este caso se proponen tres formas:

- Ofreciendo a los usuarios conocer un concepto original, una historia profunda y un universo expandible con capacidad de ofrecer mucho más contenido en un futuro, se busca motivarlos a formar una comunidad de jugadores, apoyando el proyecto tanto en difusión como en el aspecto económico a través de donaciones voluntarias.
- Ofrecer herramientas bajo cierto precio, que puedan utilizar otros desarrolladores *indies* para sus proyectos, empaquetados en *assets*, que pueden ir desde los modelos utilizados hasta librerías de código. De tal manera que puedan ser utilizadas en el desarrollo de proyectos similares o en la creación de *mods* (modificaciones del juego original por parte de la comunidad).
- Publicar contenido extra al juego base de forma paga, los cuales pueden ir desde ítems hasta nuevos niveles.

Presupuesto y equipo

El proyecto no cuenta con un presupuesto para el desarrollo del mismo, se realiza en una máquina con las siguientes especificaciones:

- Intel(R) Core(TM) i3-2330M CPU @ 2.20GHz
- 4,00 GB de RAM
- Gráficos integrados Intel(R) HD Graphics 3000
- Resolución del monitor principal: 1366x768

Cronograma e Hitos

Hito	Descripción	Fecha
1	Personaje principal completamente funcional, con modelo y texturas finalizadas e implementadas.	18-06-2016
2	*Gameplay* totalmente funcional, interacción entre objetos en funcionamiento.	16-07-2016
3	Producto básico funcional, jugable y listo para ser sometido a pruebas.	30-07-2016
4	Producto funcional con un entorno virtual más definido al del hito anterior, sin detalles decorativos tan resaltantes.	26-08-2016
5	Escenario terminado con acabados decorativos y sin modificaciones pendientes.	10-09-2016
6	Producto listo para ser puesto en fase beta	26-09-2016
7	Videojuego listo para ser liberado, con todas las correcciones realizadas.	17-10-2016

Riesgos

Para el desarrollo de videojuegos es necesario un equipo multidisciplinario para llevar a cabo las tareas necesarias, por lo tanto, el principal riesgo de este proyecto es la consecución de la meta por una única persona en el equipo de desarrollo.

HOJAS DE METADATOS

Hoja de Metadatos para Tesis y Trabajos de Ascenso – 1/6

Título	Desarrollo De Un Videojuego Indie, 3d, Del Género Role Player
Subtítulo	

Autor(es)

Apellidos y Nombres	**Código CVLAC / e-mail**	
Galindo González Juan David	**CVLAC**	**20992475**
	e-mail	**Juandg91@gmail.com**
	e-mail	
	CVLAC	
	e-mail	
	e-mail	
	CVLAC	
	e-mail	
	e-mail	

Palabras o frases claves:

Videojuego, 3D, Unity, Role Player.

Hoja de Metadatos para Tesis y Trabajos de Ascenso – 2/6

Líneas y sublíneas de investigación:

Área	Subárea
Ciencias	**Informática**

Resumen (abstract):

Se desarrolló un videojuego indie del género *role player* en el motor de juegos Unity, usando herramientas como Blender, Photoshop, iClone, para la creación de gráficos en 3D y el lenguaje de programación C# para la programación de las funcionalidades del mismo. Se tomó como referencia la metodología SUM (Acerenza, Copes, Mesa y Viera., 2009) para videojuegos, una modificación de Scrum para acoplarse mejor a este propósito. Mediante un ciclo inicial de planificación se diagramaron las tareas y actividades necesarias para la elaboración del juego, logrando en un siguiente ciclo iterativo diversas versiones del producto con funcionalidades nuevas y/o más refinadas. Se consiguió un producto beta luego de cuatro fases de testeo en usuarios, en donde, de realizaron mejoras y ediciones en base al *feedback* de los *testers* en cada una de las iteraciones.

Hoja de Metadatos para Tesis y Trabajos de Ascenso – 3/6

Contribuidores:

Apellidos y Nombres	ROL / Código CVLAC / e-mail	
Romero Bottini Carmen Victoria	**ROL**	**CA [ ] AS [x] TU [ ] JU [ ]**
	CVLAC	**10947403**
	e-mail	**cvromerob@gmail.com**
	e-mail	
Galantón Alejandra	**ROL**	**CA [ ] AS [ ] TU [ ] JU [x]**
	CVLAC	**11383261**
	e-mail	**agalanto@gmail.com**
	e-mail	
Suárez Morantes Mariluz	**ROL**	**CA [ ] AS [ ] TU [ ] JU [x]**
	CVLAC	**8642900**
	e-mail	**Mariluz1968@gmail.com**
	e-mail	
Fuentes Marquez Ana Teresa	**ROL**	**CA [x] AS [ ] TU [ ] JU [ ]**
	CVLAC	**12666425**
	e-mail	**Afuentes_marquez@hotmail.com**
	e-mail	

Fecha de discusión y aprobación:

Año	Mes	Día
2017	**02**	**24**

Lenguaje: **SP**

Hoja de Metadatos para Tesis y Trabajos de Ascenso – 4/6

Archivo(s):

Nombre de archivo	Tipo MIME
tesis-galindoj.doc	**Aplication/word**

Alcance:

Espacial: _____

Temporal: _____

Título o Grado asociado con el trabajo: Licenciado en Informática

Nivel Asociado con el Trabajo: Licenciado

Área de Estudio: Informática

Institución(es) que garantiza(n) el Título o grado: Universidad de Oriente

Hoja de Metadatos para Tesis y Trabajos de Ascenso – 5/6

UNIVERSIDAD DE ORIENTE
CONSEJO UNIVERSITARIO
RECTORADO

CU Nº 0975

Cumaná, 04 AGO 2009

Ciudadano
Prof. JESÚS MARTÍNEZ YÉPEZ
Vicerrector Académico
Universidad de Oriente
Su Despacho

Estimado Profesor Martínez:

Cumplo en notificarle que el Consejo Universitario, en Reunión Ordinaria celebrada en Centro de Convenciones de Cantaura, los días 28 y 29 de julio de 2009, conoció el punto de agenda "SOLICITUD DE AUTORIZACIÓN PARA PUBLICAR TODA LA PRODUCCIÓN INTELECTUAL DE LA UNIVERSIDAD DE ORIENTE EN EL REPOSITORIO INSTITUCIONAL DE LA UDO, SEGÚN VRAC Nº 696/2009".

Leído el oficio SIBI – 139/2009 de fecha 09-07-2009, suscrita por el Dr. Abul K. Bashirullah, Director de Bibliotecas, este Cuerpo Colegiado decidió, por unanimidad, autorizar la publicación de toda la producción intelectual de la Universidad de Oriente en el Repositorio en cuestión.

Comunicación que hago a usted a los fines consiguientes.

UNIVERSIDAD DE ORIENTE
SISTEMA DE BIBLIOTECA
RECIBIDO POR
FECHA 5/8/09 HORA 5:30

Cordialmente,

JUAN A. BOLAÑOS CURIEL
Secretario

REPÚBLICA BOLIVARIANA DE VENEZUELA * UNIVERSIDAD DE ORIENTE * SECRETARÍA CONSEJO UNIVERSITARIO

C.C: Rectora, Vicerrectora Administrativa, Decanos de los Núcleos, Coordinador General de Administración, Director de Personal, Dirección de Finanzas, Dirección de Presupuesto, Contraloría Interna, Consultoría Jurídica, Director de Bibliotecas, Dirección de Publicaciones, Dirección de Computación, Coordinación de Teleinformática, Coordinación General de Postgrado.

JABC/YGC/maruja

Apartado Correos 094 / Telfs: 4008042 - 4008044 / 8008045 Telefax: 4008043 / Cumaná - Venezuela

Hoja de Metadatos para Tesis y Trabajos de Ascenso- 6/6

Artículo 41 del REGLAMENTO DE TRABAJO DE PREGRADO (vigente a partir del II Semestre 2009, según comunicación CU-034-2009) : "los Trabajos de Grado son de la exclusiva propiedad de la Universidad de Oriente, y sólo podrán ser utilizados para otros fines con el consentimiento del Consejo de Núcleo respectivo, quien deberá participarlo previamente al Consejo Universitario para su autorización".

Galindo Juan
Autor

Prof: Romero Carmen
Asesor