ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERIA DE SISTEMAS

DESARROLLO DE UN JUEGO DE VIDEO BIDIMENSIONAL POR MEDIO DE LA CREACIÓN DE UN MOTOR GRÁFICO REUTILIZABLE

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

LUIS FERNANDO ALVAREZ ARTEAGA

lalvarez@pronaca.com

DIRECTOR: ING. XAVIER ARMENDARIZ

xarmendariz@uamericas.com.ec

Quito, Junio 2008

DECLARACIÓN

Yo, Luis Fernando Alvarez Arteaga, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Luis Fernando Alvarez Arteaga

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Luis Fernando Alvarez Arteaga, bajo mi supervisión.

Xavier Armendáriz

DIRECTOR DE PROYECTO

AGRADECIMIENTOS

Primeramente quiero dar gracias a Dios, ya que por medio de una gran cantidad de acontecimientos a lo largo de mi carrera y de mi vida me demostró que él es la única fuerza que me ha sostenido y colmado de bendiciones a lo largo de estos años.

En segundo lugar quiero agradecer a mi familia, quienes han sido un apoyo incondicional en el desarrollo de este proyecto con sus oraciones y palabras de aliento, más aún en los momentos de mayor dificultad.

También agradezco a mi director de tesis, el Ingeniero Xavier Armendáriz, quien con su experiencia profesional en el campo del desarrollo de software contribuyó a que este proyecto tome forma y sea encaminado de la mejor manera.

Finalmente quiero agradecer a la gente con la que compartí tantos buenos momentos en esta facultad, y que de una manera u otra me enseñaron lecciones importantes. Las chicas Lynx, con las que compartí tantas alegrías y desvelos; los muchachos Nemesis, quienes con su espontaneidad siempre supieron arrancarme una o dos sonrisas; los profesores de la facultad, que con su conocimiento pudieron guiarme a lo largo de mi carrera.

A todas estas personas, y a las que compartieron su tiempo conmigo de una u otra forma en el desarrollo de este proyecto les extiendo mis más sinceros agradecimientos

Luis Fernando Alvarez A.

DEDICATORIA

Dedico esta tesis a Dios por mostrarme día a día su gloria. A mis padres Fernando y Sylvia por ser un apoyo constante e incondicional. A Juan Pablo y Esteban por ser mis mejores amigos, y quienes me hacen caer en cuenta de mis errores.

Finalmente dedico este trabajo a todas las personas cuyo niño interior todavía no ha crecido, para los cuales los videojuegos son una gran parte de sus vidas.

CONTENIDO

RESUMEN

El presente trabajo contiene la documentación del proceso de desarrollo del videojuego Piratas: La Tumba del Corsario, y su respectivo motor reutilizable. Este documento está dividido en cinco capítulos:

En el primer capítulo se planteará el tema básico del proyecto, al describir tanto el proceso de desarrollo de un videojuego, como un resumen y explicación del uso de motores en la industria.

Luego del análisis del problema, en el segundo capítulo se propone el diseño de concepto para el videojuego Piratas: La Tumba del Corsario, mediante el modelado básico de las reglas del juego.

En el tercer capítulo se realiza la selección de la metodología para el desarrollo del producto, y en base a ella se hace un análisis y diseño de la arquitectura del sistema.

El cuarto capítulo detalla el desarrollo del motor y del videojuego que hará uso del mismo. En éste se incluyen los documentos generados al realizar las pruebas de sistema y la evaluación del producto.

Finalmente, en el quinto capítulo se plantean las conclusiones obtenidas al realizar este proyecto y las recomendaciones pertinentes al tema tratado.

PRESENTACION

Desde la aparición de las interfaces gráficas en los computadores personales, el desarrollo de software de entretenimiento ha sido un campo poco explotado por la dificultad de implementación que tiene un producto de este tipo. Más aún, con las nuevas tecnologías emergentes en el sector de los videojuegos, la implementación de este tipo de software ha llevado a los equipos de desarrollo a invertir una gran cantidad de tiempo y de recursos económicos para obtener un producto de alta calidad que sea competitivo en el mercado.

En nuestro país no ha existido una incursión en el campo del desarrollo de videojuegos, ya que como se mencionó, es un negocio que requiere de una inversión muy alta, tanto en recursos de hardware como de software, y personal especializado en este tipo de productos, el cual no tiene la formación adecuada para la tarea.

Mediante este trabajo se pretende implementar una herramienta que ayude a los desarrolladores de videojuegos a crear productos de una forma rápida, flexible, y con un costo y una curva de aprendizaje muy baja. Así mismo se contempla la reusabilidad de esta herramienta en distintos proyectos de este tipo, lo que ayudará al desarrollo de este campo en nuestro país.

Adicionalmente, considerando que el desarrollo de videojuegos se ha caracterizado por ser un proceso totalmente informal, en este trabajo se plantea la selección de una metodología que se ajuste a un proyecto de este tipo de manera adecuada.

INTRODUCCIÓN

A lo largo de los años, el campo del desarrollo de software ha evolucionado de manera acelerada. Mientras el mercado requiere de más productos, es tarea de las compañías desarrolladoras producir software de una manera ágil y flexible, pero sin dejar de lado la calidad y una correcta implementación de los requerimientos del usuario final. Para este fin se han creado varias metodologías de desarrollo de software, las cuales se adaptan a los diferentes tipos de producto y a los requerimientos propuestos dentro de un proyecto.

El software de entretenimiento (mejor conocido como videojuegos), ha tomado un gran impulso desde la creación de las interfaces gráficas. Se han ido desarrollando juegos cada vez más inmersivos y realistas a medida que el hardware ha evolucionado. Es así que en la actualidad el software puede simular un entorno totalmente digital creado en base a millones de polígonos que representan un objeto del mundo real.

Este tipo de software, a lo largo de los años no ha podido utilizar estas metodologías de desarrollo de una forma adecuada. Esto se debe a varios aspectos, tales como la alta dependencia hacia los dispositivos de hardware, la demanda del mercado, y la alta cantidad de cambios en los requerimientos. Por estas razones el desarrollo de videojuegos ha tomado un enfoque más informal al momento de su creación.

Pese a tener estos factores que hacen al proceso de desarrollo de videojuegos tan difícil de acoplar a una metodología tradicional, el análisis hecho a lo largo de este documento demuestra que existen metodologías nuevas que pueden romper esta brecha existente. Se plantea como una solución el uso de la metodología Scrum, con lo cual un proyecto de desarrollo de un videojuego puede tener un poco más de formalidad sin dejar de lado la agilidad y flexibilidad que caracterizan a este tipo de proyectos.

En conjunto con esta metodología ágil, se plantea la reutilización de código por medio del uso de un motor gráfico. Esta herramienta permite al programador concentrarse en la implementación de las reglas de juego, dejando de lado la interacción entre los dispositivos externos y el sistema operativo, lo cual suele ser el problema primordial dentro de cualquier implementación.

De esta manera, los diseños de videojuegos pueden tener características innovadoras y creativas, ya que el ahorro de tiempo que trae el uso de un motor puede ser enfocado a la implementación de estas ideas.

A lo largo de este estudio veremos la importancia de reutilizar estos componentes previamente creados en conjunto con una metodología de desarrollo flexible, lo que será el punto de partida para la evolución de los procesos de desarrollo de videojuegos en nuestro país.

1. PLANTEAMIENTO DEL PROBLEMA

1.1. PROCESO DE DESARROLLO DE VIDEOJUEGOS

El proceso de desarrollo de videojuegos tiene fases muy similares a las de una aplicación tradicional, con cambios muy ligeros. En esta sección podremos ver una breve descripción de cada una de estas fases.

1.1.1. ETAPA DE PRE-PRODUCCIÓN

La etapa de pre-producción puede ser asociada con la fase de levantamiento de requerimientos en una aplicación tradicional. En ella se diseña la idea básica del juego, junto con las reglas y objetivos que lo acompañan.

Concepto inicial:

En esta fase la idea de la temática del juego es visualizada por un diseñador. Aquí se seleccionará entre otros aspectos, el género del juego que va a ser desarrollado, tomando en cuenta las expectativas y gustos del mercado al que va dirigido. En esta fase se genera el denominado “Arte Conceptual”, el cual consta de dibujos y bosquejos sobre la idea principal. Aquí se generan el *tratamiento inicial*, el cual es un documento que a grandes rasgos describe la temática del juego, y el *álbum de arte conceptual*, en el cual se incluyen piezas artísticas sobre el concepto básico a ser desarrollado.

Figura 1: Arte conceptual para el juego Warcraft III: Reign of Chaos[1]

Diseño de concepto inicial:

Una vez seleccionada la idea principal se la refina, se tratan aspectos generales del juego tales como: reglas básicas, look and feel, descripción de interfaz, objetivos del juego, entre otros. En esta fase se genera un *documento de diseño inicial*, en el cual se plantean aspectos más profundos sobre el concepto del juego.

Diseño de concepto detallado:

Finalmente, se plantean cada uno de los aspectos del juego en lenguaje natural, tales como: reglas avanzadas del juego y detalle de los elementos y objetivos de cada uno de los niveles. Cabe resaltar que este diseño no es definitivo, ya que puede ser cambiado al hacer un posterior análisis de viabilidades técnicas. En esta fase, se genera el documento de diseño detallado.

1.1.2. ETAPA DE PRODUCCIÓN

Luego de que la idea ha sido desarrollada y aprobada, se procede con el diseño de la misma. En esta fase se comparan los elementos diseñados previamente con las características técnicas a ser

[1] GREEN, Jeff; FARKAS, Bart; *The Art of Warcraft*

utilizadas, para obtener un producto final muy cercano al visualizado por el diseñador.

Diseño de arquitectura:

En esta fase se diseña la arquitectura del sistema, es decir, se detalla la implementación de cada una de las partes del producto. Esta fase incluye el análisis de las clases del producto, junto con otros modelos y diagramas que reflejarán el diseño de concepto previamente obtenido en la fase de pre-producción. Así mismo, en esta fase se diseñará el **motor gráfico** (engine) del juego, cuyas funciones serán detalladas posteriormente. En esta fase se obtienen diferentes diagramas y modelos, descritos en las figuras 2 y 3.

Matrices de interacción

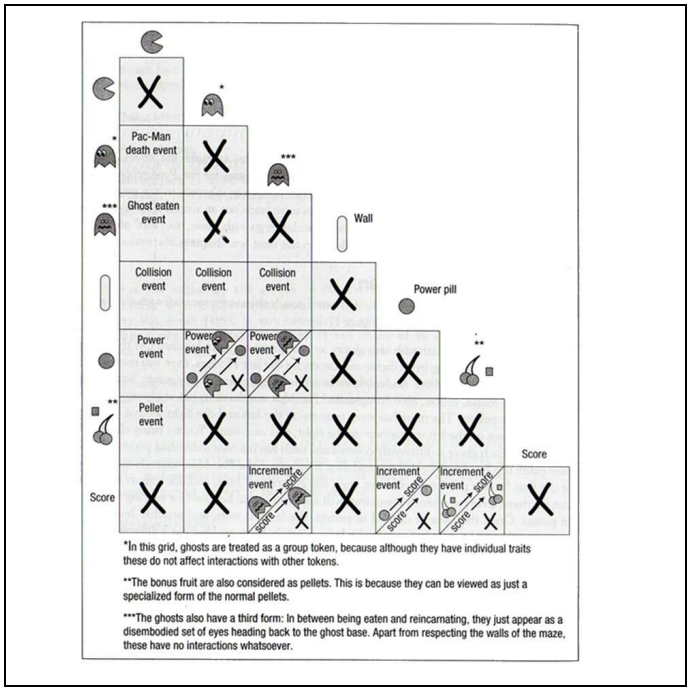

Figura 2: Matriz de interacción para el juego Pac-Man. En este diagrama se definen los eventos a ejecutarse de acuerdo a la interacción de los diferentes elementos de juego.[2]

[2] ROLLINGS, Andrew; MORRIS, Dave; *Game Architecture and Design*

Diagrama de clases

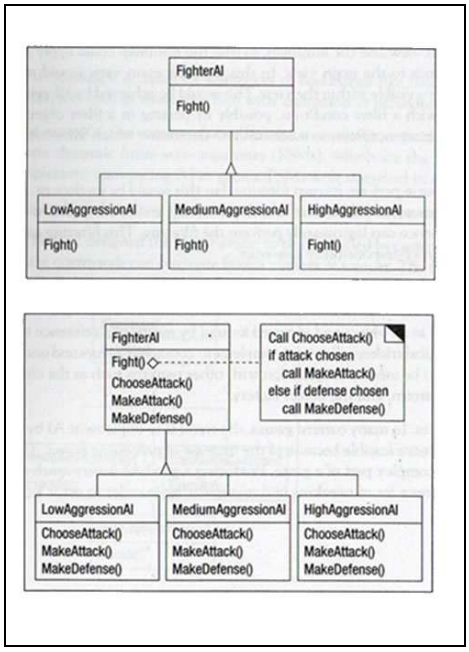

Figura 3: Diagrama genérico de clases para un videojuego de[3] peleas

Desarrollo del producto:

A partir del diseño de la arquitectura del producto, se procederá a la implementación del motor gráfico y luego a la creación del videojuego mediante la utilización de mencionado motor. Cabe resaltar que en la fase de desarrollo se usa un equipo multidisciplinario que consta de:

- Un *productor*, que será el líder del equipo de desarrollo, y se encargará de verificar la calidad y consistencia de todos los elementos en el producto final.
- Uno o más *programadores*, que se encargarán de implementar las reglas del diseño conceptual en el producto final.
- Uno o más *artistas gráficos*, que trabajarán en el aspecto visual del juego, mediante la generación de animaciones, imágenes, videos, etc.

[3] ROLLINGS, Andrew; MORRIS, Dave; *Game Architecture and Design*

- Uno o más *productores de sonido*, los cuales serán responsables de generar sonidos y pistas musicales para su uso en el juego.
- Uno o más *diseñadores de niveles*, que se encargarán de crear cada uno de los niveles del juego final.

Pruebas:

En esta sección se hará un análisis extensivo de los errores existentes en el producto. Las pruebas serán realizadas por un equipo de probadores o *testers*, quienes deberán documentar todas las fallas y errores encontrados en el producto, y tener una estrecha relación con el equipo de desarrollo para que mencionadas fallas sean corregidas. En esta fase se generan documentos de reporte de errores, los cuales servirán para que el programador pueda rectificar los diversos errores que existan en el programa.

Evaluación

En esta fase, el juego ya se habrá lanzado al público en general, y mediante ciertos estudios de mercado se determinará si es conveniente desarrollar una nueva versión, o un parche que aumente funcionalidad a la actual.

1.1.3. MANTENIMIENTO

Muchos juegos son considerados como completos al momento de su lanzamiento, sin embargo con la introducción de los videojuegos en línea, una gran cantidad de ellos reciben modificaciones luego de que el juego ha sido distribuido. Así mismo, se liberan parches de compatibilidad con el nuevo hardware que pueda salir al mercado, tales como tarjetas de video, tarjetas de sonido, periféricos de manejo, etc.

Como se puede apreciar, el proceso de desarrollo de un videojuego es similar en ciertos aspectos al desarrollo de una aplicación

tradicional. Podemos apreciar una comparación entre los enfoques de desarrollo de software en la tabla 1.

Desarrollo de videojuegos		**Desarrollo de una aplicación tradicional**
Pre-producción	- Concepto inicial - Diseño de concepto inicial - Diseño de concepto detallado	Análisis de requerimientos
Producción	- Diseño de arquitectura	Diseño del producto
	- Desarrollo del producto	Implementación del producto
	- Pruebas	Pruebas
	- Evaluación	*(No asociado)*
Mantenimiento		Mantenimiento

Tabla 1: Comparación entre fases desarrollo de videojuegos y de aplicaciones tradicionales

1.2. USO DE MOTORES GRÁFICOS EN LA CREACIÓN DE VIDEOJUEGOS

1.2.1. DEFINICIÓN DE MOTOR

Un motor es el componente básico y primordial de un videojuego. Provee al desarrollador con tecnologías subyacentes, simplifica el desarrollo y permite que el juego corra en múltiples plataformas y consolas. Un motor típicamente incluye las siguientes funcionalidades.

- *Motor de rendering:* Componente de manejo de elementos gráficos del juego. Puede ser enfocado a gráficos bidimensionales (2D) o tridimensionales (3D)
- *Motor de física:* Componente que simula modelos de física Newtoniana por medio del uso de variables y fórmulas para la generación de movimiento real en un videojuego.
- *Motor de detección de colisiones:* El cual incluye algoritmos para comprobar colisiones entre sólidos, calcular trayectorias y puntos de impacto en una simulación.

- *Motor de sonido:* Componente que maneja elementos de sonido de un videojuego, tales como manejo de canales múltiples, volúmenes, tonos, etc.
- *Motor de inteligencia artificial:* Componente que simula inteligencia en el comportamiento de los personajes no jugables de un videojuego por medio de algoritmos.
- *Motor de red:* Componente que controla la comunicación entre dos computadores por medio de dispositivos de red, para la implementación de modos de juego multijugador.

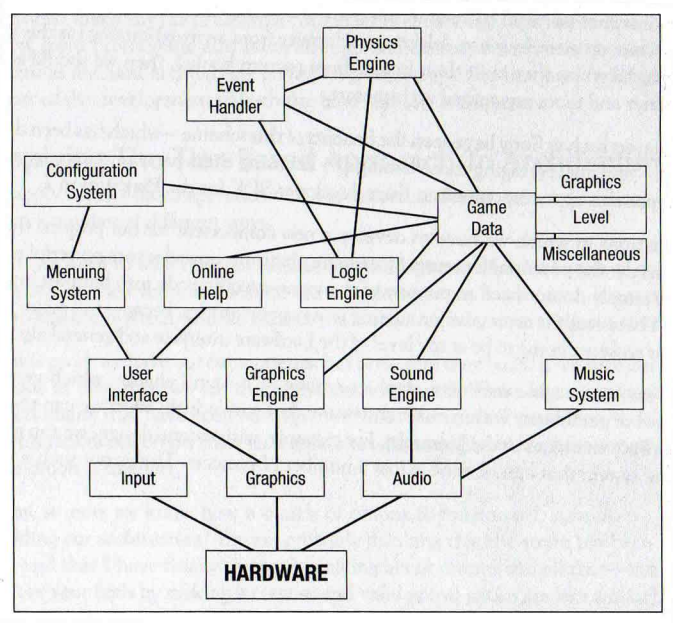

Figura 4: Estructura completa de un motor[4]

1.2.2. OBJETIVOS DE UN MOTOR

- Ahorrar tiempo a los desarrolladores, por medio de la implementación de funciones comunes a todos los videojuegos.
- Obtener nuevas versiones de juegos ya existentes, por medio del concepto de reutilización de las clases del motor.
- Proveer la capacidad de crear un kit de desarrollo de software (SDK) sobre dicho motor para facilitar las tareas relacionadas al desarrollo del videojuego.

[4] ROLLINGS, Andrew; MORRIS, Dave; *Game Architecture and Design*

1.2.3. MOTORES COMERCIALES

Nombre	Juegos	Compañía
Dark Engine	Thief 1 Thief 2 System Shock 2	Looking Glass Studios
Serious Engine 2	Serious Sam 2	Croteam
Glacier Engine	Timan	IO Interactive
Lionhead Engine	Fable 2	Lionhead Studios
Vengeance 2 Engine	Bioshock	Irrational Games
GoldSrc	Half-Life	Valve
SAGE Engine	Command & Conquer: Generals	EA
Source Engine	Half Life 2	Valve
CryEngine	Far cry	Crytek
Unreal Engine 3	Unreal Tournament 3	Epic Games
Doom 3 Engine	Doom 3 Quake 4 Prey Enemy Territory Quake Wars	ID Software

Tabla 2: Motores comerciales, juegos que lo utilizan y compañía desarrolladora[5]

[5] List of Game Engines – http://en.wikipedia.org/wiki/List_of_game_engines

2. ANÁLISIS Y DISEÑO CONCEPTUAL

2.1. CONCEPTUALIZACIÓN DEL VIDEOJUEGO

2.1.1. TRATAMIENTO INICIAL

El videojuego a ser desarrollado será del género de plataformas[6] (asociado a videojuegos tales como la saga de Mario, de Nintendo, o la saga de Sonic, de SEGA). Este es un género el cual era muy explotado en el pasado, sin embargo en la actualidad, con la aparición de los gráficos tridimensionales ha sido dejado de lado.

Pese a este enfoque actual, la innovación en el modo de juego es un factor fundamental, ya que es un elemento que podría resucitar este género.

La temática fundamental del juego (historia o argumento) trata sobre la piratería en los océanos en el siglo 18. Se ha decidido trabajar sobre este tema por dos motivos principales:

- El tema es bastante extenso, y el desarrollador conoce a fondo la temática, por lo que las ideas serán claras en el desarrollo del producto, y se podrá aportar con nuevas ideas o cambios de manera constante.

- La temática no ha sido explotada de manera adecuada en juegos existentes, por lo que la correcta implementación de la misma en el

[6] Juego de Plataformas: Género de videojuegos en los que se debe saltar, andar o escalar sobre una serie de obstáculos, entre ellos los enemigos, los que hay que superar para completar el juego. Suele usar desplazamiento horizontal para simular el movimiento del personaje principal a lo largo de un nivel.

juego a desarrollarse brindaría a la audiencia un tema nuevo y original.

La historia se basará en la temática ya mencionada, y será una parte muy importante del producto, sin embargo estará sujeta a cambios en caso de existir nuevas ideas al momento del desarrollo.

El videojuego constará de varias fases, cuyo aspecto gráfico será muy elaborado. Esto se ha planeado para ser una competencia con los juegos actuales, que hacen uso de tecnologías tridimensionales para la creación de gráficos. Sin embargo, para este proyecto de titulación, tan solo se elaborará una versión demo del videojuego, la cual demostrará el funcionamiento básico del motor y la temática principal del juego.

El nombre que se le ha dado al producto es Piratas: La Tumba del Corsario. El nombre se basa principalmente en la trama que tendrá el juego, donde se contará la historia de un pirata el cual decide desafiar los secretos de una antigua tumba llena de tesoros, y las consecuencias que le trae esta acción. Se ha pensado en crear dos juegos luego de éste, los cuales complementen la historia del producto actual, y traerán una mayor expectativa a los jugadores luego de finalizada la primera entrega.

Finalmente, cabe resaltar que el juego, si bien no contendrá aspectos técnicos de última generación, tendrá un diseño de idea muy elaborado, por lo que existirá una compensación en este aspecto. Uno de los mayores debates en el desarrollo de videojuegos ha sido la implementación de una idea totalmente original y nueva, con prestaciones gráficas que no son de última generación, contra un enfoque de productos técnicamente superiores, de última tecnología con diseños conceptuales deficientes, y que tienen mucho parecido

con juegos actuales. Se ha decidido optar por la primera opción, tomando en cuenta el tiempo de desarrollo que se le dará al producto.

2.1.2. ALBUM DE ARTE CONCEPTUAL

En esta sección se detallan algunas de las imágenes de concepto del juego. Todas ellas fueron realizadas por el equipo de desarrollo expresamente para este proyecto.

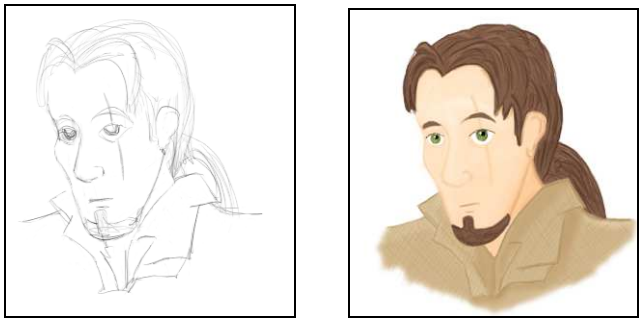

Figura 5: *Marius – Primer Bosquejo*

Figura 6: *Marius – Segundo Bosquejo*

Figura 7: *Navarro - Primer Bosquejo*

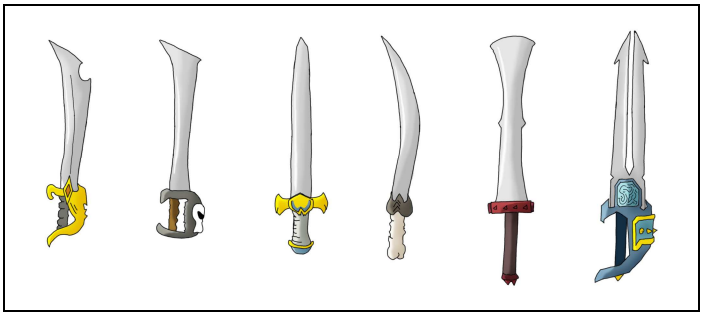

Figura 8: *Armas Nº 1*

Figura 9: *Armas Nº 2*

Figura 10: *Armas Nº 3*

Figura 11: *Marius – Bosquejo Final*

Figura 12: *La calma antes de la tormenta*

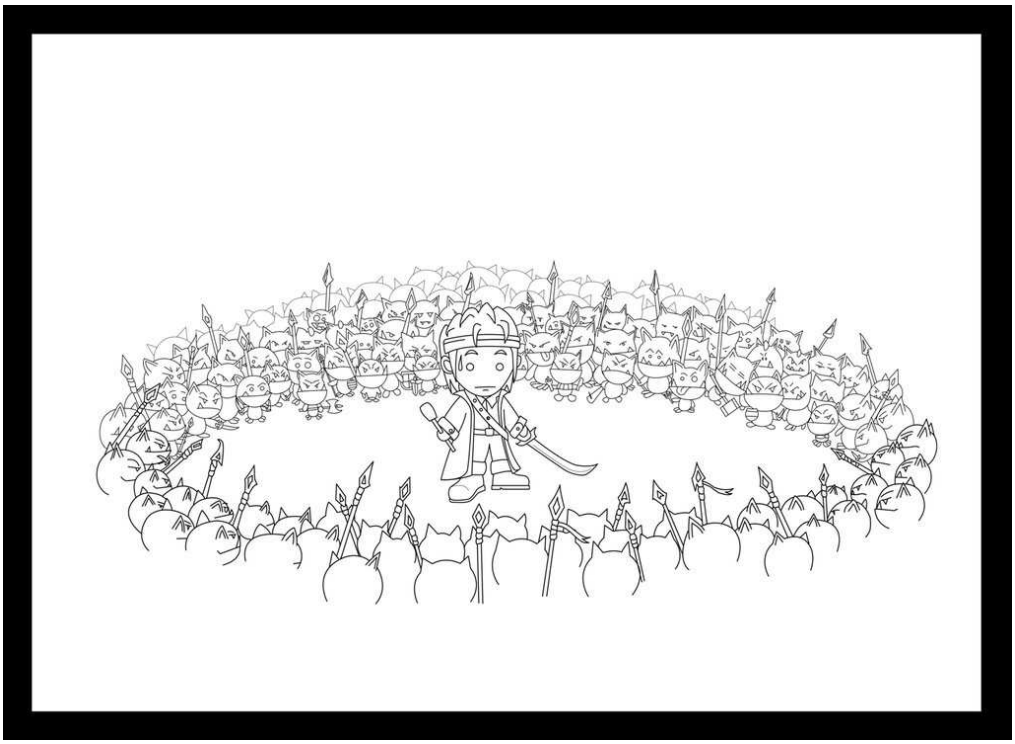

Figura 13: *Bosquejo: Te tenemos rodeado*

Figura 14: *Te tenemos rodeado*

2.2.ANÁLISIS Y DISEÑO DE CONCEPTO INICIAL

2.2.1. DOCUMENTO DE DISEÑO INICIAL

Introducción:

En este documento se tratan aspectos generales sobre el juego Piratas: La Tumba del Corsario. Mediante el mismo se pretende dar una visión general sobre el concepto principal del videojuego. Así mismo, este documento servirá de base para posteriores entregables del proyecto, tales como documentos técnicos de arquitectura y diseños detallados.

Generalidades:

Piratas: La Tumba del Corsario es un juego de plataformas ambientado en el siglo XVII, en los mares del Caribe. No está basado en eventos históricos, a pesar de contener ciertas referencias a los mismos. Los hechos descritos en el juego son totalmente ficticios.

Los jugadores deben atravesar y superar un número determinado de fases o niveles para finalizar el juego, cada una de las cuales tendrá sus objetivos determinados. Se estima que un jugador promedio podrá finalizar el juego en un período de aproximadamente 6 horas.

Jugador:

El jugador controlará a un solo personaje del juego, guiándolo por cada una de las fases. La apariencia y habilidades iniciales del mismo estarán predeterminadas, sin embargo irán cambiando a lo largo de las fases. Dicho personaje principal será definido posteriormente en el diseño de la historia que acompaña al juego.

Piratas: La Tumba del Corsario no tendrá un modo multijugador, por las limitaciones del género de plataformas, el cual no permite una integración adecuada con este modo de juego.

Look and Feel:

La pantalla principal será una representación 2D[7] de escenarios del Caribe, junto con animaciones aparentes al tema principal del juego. La cámara estará fija siempre en el personaje principal, como es usual en los juegos de plataformas, sin embargo se desplazará de

[7] 2D: Bidimensional, es decir generado por medio de gráficos e imágenes planas. En estos casos, para simular un efecto de profundidad se usan capas de estas imágenes.

manera horizontal y vertical según sea necesario para encuadrar al personaje en todo momento.

El estilo artístico está basado en diferentes caricaturas americanas con temática de piratas. Los personajes del juego serán sprites8, no figuras poligonales, basados en personajes estereotípicos de películas, libros y cultura general.

Las películas de base utilizadas para generar el concepto del juego son Piratas del Caribe: La Maldición del Perla Negra, Piratas del Caribe: El Cofre del Hombre Muerto, Cutthroat Island, Las Aventuras de Simbad el Marino, Las Aventuras de la Familia Robinson, La Isla del Tesoro.

Las fuentes musicales son Canciones Irlandesas antiguas y bandas sonoras de películas de piratas y navegantes.

Los libros de base son La Isla del Tesoro, Robinson Crusoe, El Corsario Negro, Barbanegra, Las Mil y una Noches (Simbad el Marino).

Interfaz:

En la interfaz general se utilizarán medidores para la salud o número de vidas del personaje, así como elementos informativos (nombre del nivel, información adicional, etc.) que ayudarán al jugador. Esto es un estándar en todos los juegos del género.

El juego será manejado por medio del teclado y/o de un dispositivo de juegos, tal como un Gamepad.

[8] Sprite: Figura bidimensional que representa a un objeto dentro de un juego. Puede o no puede ser controlada por el jugador.

Inicio del juego:

Al inicio de cada fase la energía y poder mágico del personaje estarán al máximo, e irán disminuyendo conforme el personaje vaya avanzando dentro de cada uno de los niveles.

Objetivos:

Los objetivos serán similares a los de otros juegos del mismo género, con ligeras modificaciones, es decir que la tarea principal será llegar hasta el final de cada una de las fases sin que las vidas del personaje se agoten.

Sin embargo, estos cambios se verán dentro de cada una de las fases, lo que dará una mayor diversidad y atractivo al juego, para que no se vuelva repetitivo.

Entidades:

Las entidades en el juego serán el personaje principal, enemigos y personajes no jugables. Solo el personaje principal será manejado por el jugador, mientras que el comportamiento de los otros se hará por medio de reglas de inteligencia artificial.

Cabe resaltar que estas entidades son visualizadas en pantalla por medio de sprites 2D.

Jugabilidad:

El jugador podrá utilizar una espada y un arma de fuego para poder avanzar por los niveles. Entre las acciones básicas que se pueden realizar están: saltar, agacharse, correr de lado a lado, atacar con la

espada, disparar, entre otros. Esto ayuda a sortear cada uno de los obstáculos en los niveles.

Las vidas y las balas para las armas de fuego pueden ser restituidos por medio de la obtención de diferentes artículos o power-ups en cada nivel (simbolizados por alimentos, íconos de vidas o balas, etc.). Algunos de estos objetos ayudarán a mejorar las características del personaje, y soportarán la personalización de sus habilidades.

En cuanto a dicha personalización, existirá un sistema de categorías para selección de habilidades, donde cada una de estas se dividirá entre 3 grupos: Alquimia, Piromanía y Magia Vudú.

Diseño de niveles:

Existirán 9 niveles en el juego, los cuales tendrán amplias diferencias uno del otro, para no resultar tediosos. Estarán todos ambientados en localidades del siglo XVII y tendrán diferentes objetivos, sin descuidar el objetivo general de un juego de plataformas. Sin embargo como ya se ha mencionado, el alcance de este proyecto de titulación se reduce a un solo nivel, que será suficiente para demostrar las características del motor desarrollado.

Requerimientos técnicos:

Piratas: La Tumba del Corsario estará diseñado para ser ejecutado en un computador Pentium 4 o superior, con al menos 512 MB en memoria RAM.

2.3.DISEÑO DE CONCEPTO DETALLADO

2.3.1. DOCUMENTO DE DISEÑO DETALLADO

Introducción:

Piratas: La Tumba del Corsario es un juego de plataformas 2D, basado en las aventuras de un pirata en diferentes localidades. El juego ha sido diseñado teniendo en cuenta el mercado de los videojuegos en la actualidad y los vacíos generados por los mismos, así como los temas que no se han abordado.

Estas especificaciones de diseño están sujetas a cambios posteriores, en caso de existir nuevas ideas y conceptos que puedan ser agregados al proyecto, o en su defecto de comprobarse la imposibilidad de implementación de uno de estos aspectos por limitaciones técnicas.

Historia:

Nuestra historia está situada a mediados del siglo 18 en las cálidas aguas del caribe, las cuales estaban infestadas de piratas. Se hablaba de antiguas maldiciones, barcos fantasma y grandes fortunas escondidas en misteriosas islas.

Existía una leyenda antigua, contada en los mares del caribe, la cual trataba acerca de la existencia de la legendaria “Tumba del Corsario”. Según contaba la historia, dicha tumba estaba repleta de oro. Sin embargo se hablaba también de una maldición sobre el tesoro, que acosaría a cualquiera que se atreviere a profanar la tumba.

Marius, un pirata despiadado y avaro, parte hacia la mencionada tumba y reclama el tesoro para sí. Esto desata la maldición, la cual haría que su alma se condenara por su avaricia. Sin embargo, descubre la forma de salvar su alma, y eso lo envía a lo largo de los siete mares a buscar los 5 fragmentos de un amuleto caribeño, para finalmente enfrentarse al mismo causante de su maldición.

Visión general de la jugabilidad:

Se conservará un estilo de jugabilidad similar a los videojuegos tradicionales de plataformas, con ligeros cambios en movimientos del personaje. Estos movimientos incluirán acciones básicas como saltar y desplazarse de derecha a izquierda, las cuales son comunes para todo el género.

En cada fase se colocarán elementos de mejora del personaje (también conocidos como power-ups), los cuales ayudarán al jugador a superar cada uno de los niveles. Estos power-ups incluirán elementos que añadan vidas al personaje, o incrementen su munición. Estos artículos serán descritos a fondo en una de las siguientes secciones. Así mismo, se hablará posteriormente de la personalización de habilidades en detalle, lo cual será un punto único del videojuego.

No existirá un modo de juego multijugador, ya que las limitaciones del género no permitirían una implementación adecuada de esta modalidad. Existirá tan solo el modo de un jugador.

En cuanto a aspectos de inicio del juego, el personaje comenzará con un número de 3 vidas y 5 balas, lo que establece un nivel de dificultad estándar para el juego.

La estructura general del programa se define en la figura 15

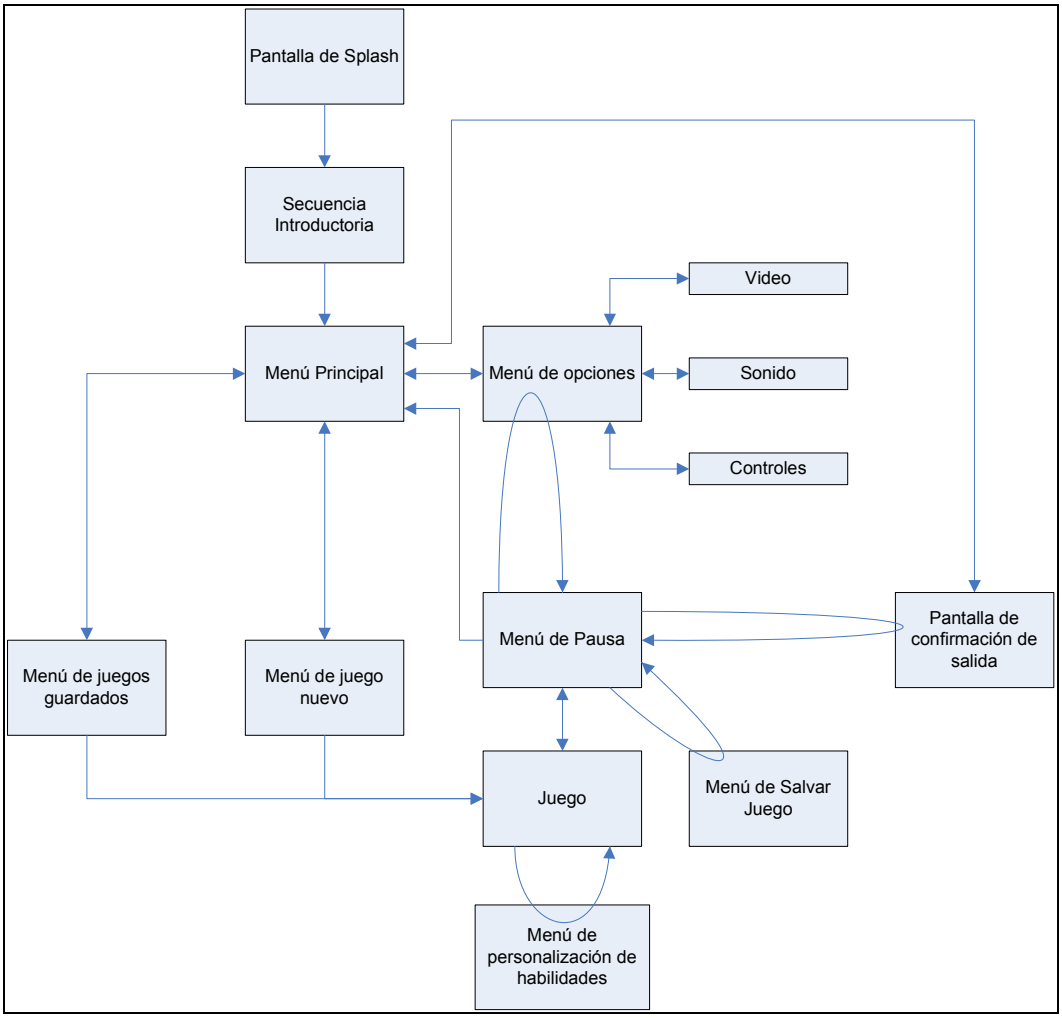

Figura 15: Estructura de menús y pantallas de la aplicación final

Descripción de acciones del personaje principal:

Los movimientos posibles para el personaje principal del juego son

- Movimiento horizontal (izquierda o derecha)
- Saltar
- Atacar con espada
- Atacar con habilidad secundaria o arma de fuego

Las habilidades secundarias estarán definidas en secciones posteriores de este documento en detalle.

Ítems y power-ups:

En el juego existirán ítems que ayudarán al personaje en cada uno de los niveles. Estos pueden ser apreciados en la tabla 3. Cabe resaltar que las imágenes mostradas en esta sección no son definitivas, sino tan solo bosquejos.

Icono	Nombre	Descripción
	Moneda de oro	Las monedas de oro estarán ubicadas a lo largo de los niveles. Al acumular 100 de ellas se restablecerá una de las vidas del personaje.
	Vida	Al obtener uno de estos ítems, el contador de vidas aumentará en 1 automáticamente.
	Balas	Las balas servirán para aumentar el número de munición disponible para el arma secundaria del personaje principal.
	Medallón	Los medallones tienen como finalidad otorgar al jugador con un punto de habilidad, el cual puede ser utilizado para aprender nuevas habilidades o hacer más fuertes a las habilidades existentes.

Tabla 3: Ítems y power-ups

Descripción de habilidades:

Como se mencionó anteriormente, mediante la obtención de medallones a lo largo de los niveles se podrán aumentar los puntos de habilidad del personaje. Existirán un total de 15 medallones en todo el juego, y se otorgará un punto de habilidad al jugador al inicio del juego, lo cual dará un total de 16 puntos.

Estos puntos podrán ser gastados en diferentes habilidades, situadas en tres diferentes categorías. Estas se verán detalladas en las tablas 4, 5 y 6.

Esta descripción incluirá el costo de energía que tiene cada habilidad, y la secuencia con la que se puede obtener las habilidades. Una habilidad puede tener diferentes rangos, y por cada punto gastado en la misma se la podrá mejorar.

- **Alquimia:** Las habilidades en esta categoría permitirán mejorar las características del personaje, tales como la energía o la salud.

Icono	Nombre	Notas
	Elixir de la vitalidad	**Requerimiento:** Ninguno **Descripción:** Aumenta el número máximo de vidas en X permanentemente **Número de rangos:** 5 **Costo de energía:** N/A
	Daga venenosa	**Requerimiento:** 5 puntos en *Elixir de la Vitalidad* **Descripción:** Lanza una daga envenenada, causando X daño inicial y Y daño a un enemigo a lo largo de 12 segundos. **Número de rangos:** 3 **Costo de energía:** 2
	Poción de vida eterna	**Requerimiento:** 5 puntos en *Elixir de la Vitalidad* **Descripción:** Te vuelves invulnerable por X segundos **Número de rangos:** 3 **Costo de energía:** 4
	Dardo de lentitud	**Requerimiento:** 3 puntos en *Daga Venenosa* **Descripción:** Lanza un dardo que hace lento al enemigo al momento del impacto. **Número de rangos:** 3 **Costo de energía:** 2
	Elixir levanta muertos	**Requerimiento:** 3 puntos en *Poción de vida eterna* **Descripción:** Tu daño se duplica, pero tu velocidad se reduce a la mitad por 10 segundos. **Número de rangos:** 1 **Costo de energía:** 3
	Sabiduría pirata	**Requerimiento:** 3 puntos en *Dardo de lentitud,* 1 punto en *Elixir levanta muertos* **Descripción:** Tu agilidad incrementa permanentemente, permitiéndote moverte más rápido y esquivar ataques enemigos con mayor facilidad. **Número de rangos:** 1 **Costo de energía:** N/A

Tabla 4: Descripción de habilidades - Alquimia

- **Piromanía:** Las habilidades en esta categoría permitirán usar habilidades de ataque con una fuerza superior a la normal.

Icono	Nombre	Notas
	Frenesí de ron	**Requerimiento:** Ninguno **Descripción:** Tienes un X% de probabilidad de duplicar el daño de tu ataque normal. **Número de rangos:** 5 **Costo de energía:** N/A
	Tiro despiadado	**Requerimiento:** 5 puntos en *Frenesí de ron* **Descripción:** Realizas un disparo a cada lado de tu personaje. **Número de rangos:** 3 **Costo de energía:** 2
	Ron volátil	**Requerimiento:** 5 puntos en *Frenesí de ron* **Descripción:** Escupe un cono de fuego que daña a los enemigos por X puntos. **Número de rangos:** 3 **Costo de energía:** 3
	Bala perforadora	**Requerimiento:** 3 puntos en *Tiro despiadado* **Descripción:** Dispara una bala que golpea hasta un máximo de 3 enemigos por X daño. **Número de rangos:** 1 **Costo de energía:** 3
	Lanzamiento de antorcha	**Requerimiento:** 3 puntos en *Ron volátil* **Descripción:** Lanza una antorcha, que quema al enemigo por X daño a lo largo de 12 segundos. **Número de rangos:** 3 **Costo de energía:** 2
	Cóctel corsario	**Requerimiento:** 3 puntos en *Lanzamiento de antorcha,* 1 punto en *Bala perforadora* **Descripción:** Lanza un cóctel molotov, que causa X daño al contacto y genera una ráfaga de balas por Y daño. **Número de rangos:** 1 **Costo de energía:** N/A

Tabla 5: Descripción de habilidades - Piromanía

- **Magia vudú:** Las habilidades en esta categoría permitirán usar habilidades de ataque y restitución de salud.

Icono	Nombre	Notas
	Reacción de energía	**Requerimiento:** Ninguno **Descripción:** Al eliminar a un enemigo te da un X% de probabilidad de aumentar tu velocidad de disparo por 5 segundos. **Número de rangos:** 5 **Costo de energía:** N/A
	Drenaje de salud	**Requerimiento:** 5 puntos en *Reacción de energía* **Descripción:** Maldice al enemigo, matándolo inmediatamente y robándole su energía vital. **Número de rangos:** 3 **Costo de energía:** 2
	Cuervo fantasma	**Requerimiento:** 5 puntos en *Reacción de energía* **Descripción:** Invoca un cuervo fantasma por 10 segundos que ataca al enemigo por X daño **Número de rangos:** 3 **Costo de energía:** 2
	Sello del destino	**Requerimiento:** 3 puntos en *Drenaje de salud* **Descripción:** Hace X puntos de daño a un enemigo luego de 5 segundos. **Número de rangos:** 3 **Costo de energía:** 3
	Danza de los espíritus	**Requerimiento:** 3 puntos en *Cuervo fantasma* **Descripción:** Invoca un espíritu por 10 segundos que ataca a un máximo de 3 enemigos. **Número de rangos:** 1 **Costo de energía:** 3
	Escudo macabro	**Requerimiento:** 3 puntos en *Sello del destino,* 1 punto en *Danza de los espíritus* **Descripción:** Te cubre con un escudo que refleja cualquier daño recibido por 10 segundos. **Número de rangos:** 1 **Costo de energía:** 4

Tabla 6: Descripción de habilidades – Magia Vudú

Diseño de niveles:

En esta sección se detallará el diseño de niveles, incluyendo a los enemigos que existirán en cada uno de ellos, la ambientación y el tipo de música que sonará a lo largo del nivel. Como se habló en el

documento de diseño inicial, el juego tendrá 9 niveles, que irán de acuerdo a la historia.

Nivel 1: El asedio de New Providence

Descripción: Este nivel se desarrolla durante un ataque de la tripulación de nuestro personaje a un puerto del Caribe. Este nivel servirá como un tutorial para enseñar la mecánica del juego y los movimientos básicos del personaje.

Ambientación: Ya que el nivel se desarrolla en un puerto bajo asedio, existirán detalles en el ambiente, tales como casas en llamas, explosiones, disparos de cañón, etc. La música será una tonada rápida, con muchos golpes de orquesta, para aumentar el dramatismo. El sonido ambiental será muy importante en este nivel, y estará compuesto de explosiones, golpes de espada, etc.

Enemigos:

- Guardia de la ciudad - Lancero
- Guardia de la ciudad – Tirador
- Milicia de la ciudad – Espadachín
- Milicia de la ciudad – Macero

Nivel 2: Isla Tortuga

Descripción: Isla Tortuga ha sido por siempre un legendario refugio de piratas. En este nivel, las cosas parecerán tranquilas al inicio, sin embargo una pelea se desatará. En este nivel nuestro personaje descubre la existencia de la tumba del Corsario.

Ambientación: En este nivel la ambientación será de un pueblo antiguo, repleto de piratas. En el ambiente existirán piratas, animales merodeando, escenas de pelea en el fondo. La música de fondo tendrá un corte irlandés, como era usual en los pueblos piratas. El sonido ambiental estará compuesto de gritos, cantos, risas, cristales rotos, etc.

Enemigos:

- Pirata - Espadachín
- Pirata – Bárbaro
- Pirata – Maestro de bestias
- Lobo entrenado
- Oso entrenado

Nivel 3: La maldición del Corsario

Descripción: En este nivel nuestro personaje entrará a la tumba del corsario, el cual es un sitio infestado de fantasmas, a causa de la maldición impuesta ahí.

Ambientación: El nivel estará ambientado en una tumba antigua, y tendrá detalles tales como telarañas, murciélagos, marcas de humedad en las murallas, etc. La música tendrá toques de suspenso y será una tonada lenta. Como sonidos ambientales existirán ruidos de ratones, murciélagos, gotas de agua, etc.

Enemigos:

- Esqueleto - Arquero
- Esqueleto – Espadachín
- Araña
- Cuervo Fantasma

Nivel 4: El templo Azteca

Descripción: En este nivel la misión es derrotar al espíritu del jaguar que habita en el templo, para obtener el medallón de la tierra.

Ambientación: El nivel estará ambientado en un templo azteca, por lo que los detalles serán murallas con runas y símbolos, caídas de agua y aves. La música será una danza tribal antigua. Los sonidos serán de animales salvajes, al estar el templo ubicado en la jungla, flujos de agua, etc.

Enemigos:

- Guerrero – Soldado Jaguar
- Guerrero – Soldado Águila
- Guerrero – Arquero
- Pantera entrenada

Nivel 5: La puerta del infierno

Descripción: Este nivel se desarrolla en un volcán activo denominado “La Puerta del Infierno”. El objetivo será derrotar al demonio de fuego que habita en él para obtener el medallón del fuego.

Ambientación: Los detalles visuales serán flujos de lava, geysers, nubes de humo, etc. La música será una tonada rápida, llena de adrenalina. El sonido ambiental estará compuesto de explosiones, ráfagas de aire, etc.

Enemigos:

- Elemental de fuego
- Elemental de roca
- Salamandra de espinas
- Demonios miniatura

Nivel 6: El cadáver de la Reina Ana

Descripción: Este nivel se desarrolla en un barco hundido, llamado “Reina Ana”. El objetivo será derrotar al líder de las serpientes marinas, Tidax, para obtener el medallón del agua.

Ambientación: El nivel está ambientado en un barco hundido, por lo que los detalles ambientales serán peces, fugas de agua, animales marinos, etc. La música será lenta, y cargada de suspenso. El sonido ambiental estará compuesto de chirridos, sonidos de animales marinos, etc.

Enemigos:

- Serpiente Marina - Lancero
- Serpiente Marina – Maestro de los venenos
- Serpiente Marina – Espiritista

- Cangrejo gigante

Nivel 7: La Montaña del Roc

Descripción: Existe un ave mitológica llamada Roc, la cual es la guardiana del medallón del viento. Para obtenerlo, nuestro personaje deberá sortear los obstáculos en la montaña donde ésta criatura habita, y derrotarla.

Ambientación: Los detalles de este nivel serán aves, rocas cayendo, nubes, etc. La música será una tonada de toque espiritual y lento, con ciertos toques de dramatismo para resaltar eventos en el nivel. Entre los sonidos ambientales se encuentran el viento, sonidos de águilas y otras aves, lobos, etc.

Enemigos:

- Tornado Viviente
- Jabalí Salvaje
- Águila real
- Guerrero de roca

Nivel 8: En el Ojo del Huracán

Descripción: Para obtener el último medallón, nuestro personaje debe vencer a la bestia marina conocida como Rigaxo. En este nivel se dará la pelea con esta mítica criatura, a bordo del barco de nuestro personaje.

Ambientación: En este nivel lucharemos sobre el barco de nuestro personaje, durante una tormenta. Los detalles principales serán la lluvia, los rayos las olas. La música será rápida y llena de suspenso. Los sonidos principales constarán de truenos, el sonido del agua golpeando contra el bote y la incesante lluvia.

Enemigos:

- Elemental de rayos
- Anguila

- Murciélago de rayos

Nivel 9: El Destino de un Pirata

Descripción: En este nivel nuestro personaje regresará a la tumba del corsario, para regenerar el amuleto de los elementos y pelear contra el temido Corsario.

Ambientación: El nivel estará ambientado en la cámara del tesoro, dentro de la tumba del Corsario, donde se realizará la pelea final con el mismo. La música para este nivel será muy rápida, con toques de cantos gregorianos para dar emoción a la batalla.

3. ANÁLISIS Y DISEÑO DEL PRODUCTO

3.1. SELECCIÓN DE LA METODOLOGÍA DE DESARROLLO

Como se mencionó anteriormente, no existe una metodología asociada al desarrollo de videojuegos, al ser éstos una rama muy especializada del desarrollo de software. Sin embargo, se ha decidido tomar como modelo de referencia a la metodología SCRUM, la cual se ajusta en la mayoría de sus aspectos al desarrollo de este tipo de productos.

3.1.1. SCRUM

Scrum es una metodología para la gestión de proyectos, basada en dos principios fundamentales:

- El mercado actual es altamente competitivo y con tecnología muy cambiante. En el desarrollo de software se pide rapidez, calidad y reducción de costos, pero para asumir estos retos es necesario tener agilidad y flexibilidad para la toma de requerimientos contínua, durante cada una de las iteraciones.
- Los ciclos de desarrollo de software tienden a ser largos, y lo que se exige por otra parte es que los ciclos sean lo más cortos posibles.

Scrum es una metodología que obedece a las necesidades anteriormente planteadas, y responde a una necesidad real en el desarrollo de software.

Esta metodología evita la alta generación documental. No es que en Scrum no se debe documentar, sino que se plantea que la documentación no sea una exigencia previa para comenzar un proyecto de desarrollo. La idea principal es generar un producto

visible, con el cual el cliente pueda evaluar los avances del proyecto, y satisfacer sus requerimientos completamente.

La metodología describe principalmente dos elementos: Los *actores*, que son las personas encargadas del proceso completo de desarrollo, y las *acciones*, que son cada una de las tareas a realizarse, ya sea repetitivamente en cada iteración (conocida en esta metodología como *Sprint*) o una sola vez en todo el proyecto.

Podemos definir cuatro actores:

- Un Propietario del producto (Product Owner), el cual se encarga de marcar las prioridades del proyecto o producto.
- Un Administrador de Scrum (Scrum Manager o Scrum Master), que asegura el seguimiento de la metodología por parte del equipo
- El Equipo de Scrum (Scrum Team), que tiene como tarea implementar la funcionalidad o funcionalidades elegidas por el propietario del producto.
- Los usuarios o clientes, que son los beneficiarios finales del producto, y que viendo el progreso del proyecto pueden aportar con ideas, sugerencias o necesidades.

Las acciones básicas a seguir dentro de la metodología son:

- La pila de producto (Product Backlog), que es la lista de tareas, funcionalidades o requerimientos a ser realizado. El propietario del producto es la persona encargada de mantener esta lista y marcar las prioridades del proyecto.
- La pila del sprint (Sprint Backlog), que es un listado de tareas obtenido de la pila de producto. Estas tareas deben ser realizadas en un período de 2 a 4 semanas. Una de las normas fundamentales de la metodología es que una vez iniciado el desarrollo de una pila de sprint, éste no puede ser

alterado o modificado. En caso de ser necesaria una modificación se debe esperar al final del desarrollo de la pila de sprint actual, y luego modificar la tarea, que pasaría a ser parte de una nueva pila.

- La reunión diaria de Scrum (Daily Scrum Meeting), la cual es una tarea iterativa que se realiza a diario, mientras dure el desarrollo de una pila de sprint. Es una reunión operativa, informal y ágil, de un máximo de 30 minutos, donde se hacen 3 preguntas principales a cada integrante del equipo.
    - Qué tareas ha realizado desde la última reunión
    - Sobre qué va a trabajar el día actual
    - Cuáles son los obstáculos que pueden impedir el avance normal de sus tareas

Adicionalmente, al inicio de cada sprint se tiene una reunión de planificación, la cual tiene como objetivo determinar las tareas del sprint actual de acuerdo a la pila de producto. De esta reunión surge un *objetivo de sprint*, el cual es un documento con una breve descripción de lo que el sprint intentará alcanzar.

Al final de cada sprint se tiene una reunión de revisión, en la cual se analiza el cumplimiento de los objetivos para el sprint terminado. En este punto se debe tener un entregable que el cliente pueda evaluar.

Finalmente, al terminar el desarrollo de una pila de sprint y la reunión de revisión, se inicia la *retrospectiva del sprint*, donde el propietario del producto revisará con el equipo los objetivos marcados inicialmente, se aplicarán los cambios y ajustes necesarios, y se marcarán los aspectos positivos y negativos del sprint.

Cabe señalar que la terminación del proyecto se da cuando en la pila del producto ya no hay tareas ni requerimientos pendientes.

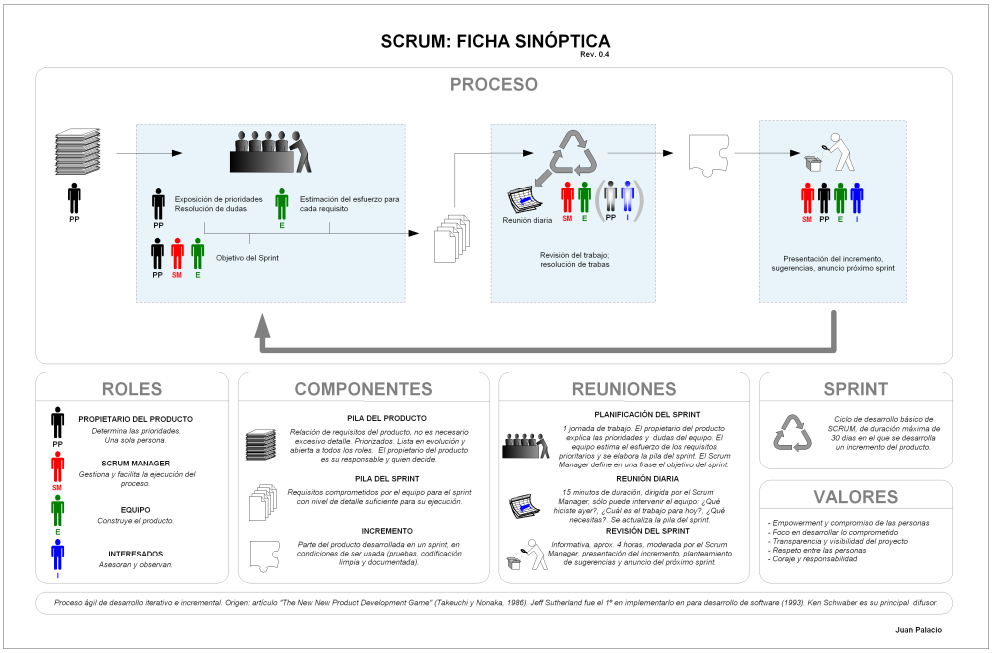

Figura 16: Diagrama de descripción de Scrum[9]

3.1.2. RAZONES PARA USAR SCRUM

Dentro del presente proyecto, se han analizado varias metodologías, sin embargo Scrum es la que más se ajusta al desarrollo del mismo. Se han planteado varias razones, las cuales son:

- El desarrollo de un videojuego debe ser realizado de manera rápida, ya que la aparición de nuevas tecnologías de hardware ocasionan que un software de este tipo (altamente dependiente de la infraestructura física) pueda quedar en la obsolescencia incluso antes de su liberación al público. La alta velocidad de desarrollo planteada por la metodología soluciona este inconveniente.
- La alta informalidad que ha existido en los proyectos de desarrollo de videojuegos ocasiona que muchas veces la documentación exigida por el proyecto quede incompleta o inconsistente. La metodología no obliga al equipo a utilizar

[9] PALACIO, Juan; *Flexibilidad con Scrum: Principios de Diseño e implantación de campos de Scrum*

documentación de ningún tipo, sin embargo, deja abierta la posibilidad para hacerlo, lo cual es beneficioso para un proyecto de alta velocidad.

- Al tener una alta interacción con el cliente, los requerimientos del mismo se plasman de manera adecuada en el producto final. Este es uno de los puntos planteados por Scrum, ya que luego de cada una de las iteraciones de desarrollo se produce una reunión con el cliente.
- Un equipo de desarrollo de videojuegos es altamente heterogéneo, ya que sus miembros son de diferentes áreas (programadores, diseñadores, ingenieros de sonido, etc.). La alta cantidad de reuniones planteadas por la metodología mejoran el control y seguimiento de los diferentes miembros del equipo, y mejoran la comunicación entre los mismos.

3.1.3. ENTREGABLES Y DOCUMENTOS DEL PROYECTO

Una vez definida la metodología que se utilizará, y en base a la misma, se han definido los entregables y documentos del proyecto, los cuales serán enumerados en la tabla 7. Cabe resaltar que éstos serán los documentos finales del proyecto, y no se incluirán versiones intermedias de los mismos por el alto número de cambios que pueden producirse.

Fase del proyecto	Documento (s)	Modelo (s)
Concepto Inicial	- Tratamiento inicial - Álbum de arte conceptual	- Dibujos - Esquemas - Bosquejos - Gráficos
Diseño de concepto inicial	- Documento de diseño inicial	----------
Diseño de concepto detallado	- Documento de diseño detallado	----------
Análisis del producto	- Pila del producto	----------
Diseño de arquitectura	- Documento de diseño de arquitectura - Pilas de sprint - Diccionario de clases	- Matriz o matrices de interacción - Diagramas de clases
Desarrollo	----------	- Imágenes - Sonidos - Música
Pruebas	- Documentos de reporte de errores	----------

Tabla 7: Documentos y modelos generados en el desarrollo del videojuego *Piratas*

3.2. ANÁLISIS DEL PRODUCTO

De acuerdo con la metodología y con los documentos de concepto planteados anteriormente, el análisis de tareas y requerimientos se resume en la *pila del producto*, cuya última versión está descrita en este apartado. Se incluyen tanto la pila de producto del motor como del juego, en conjunto con las prioridades asignadas por el propietario del producto que servirán para planear cada uno de los sprints.

Motor	
Tarea	**Prioridad**
Creación del motor	1
Dibujar imágenes estáticas en pantalla	1
Soporte para transparencia en imágenes	1
Dibujar animaciones en pantalla	1
Creación del motor	1
Dibujar fondos con desplazamiento	2
Dibujar fondos en capas	2
Manejar sprites	3
Manejar acciones de límite de pantalla	3
Manejar colisiones entre sprites	3
Manejar pulsaciones del teclado	4
Manejar dispositivos de juego (joysticks)	4
Manejar métodos generales de dibujo de sprites	5
Manejar métodos generales de actualización de sprites	5
Manejar sonidos y música de fondo	6

Tabla 8: Pila de producto para el desarrollo del motor

Juego	
Tarea	**Prioridad**
Creación de animaciones - Personaje principal	1
Creación de animaciones – Enemigos	1
Creación de animaciones – Objetos	1
Creación de fondos	2
Creación de logotipos y pantallas splash	2
Creación de clase de sprites de juego	3
Creación de subclases de sprites de juego	3
Programación de movimientos de personaje principal	4
Programación de movimientos de enemigos	4
Programación de colisiones entre sprites del juego	4
Programación de esquemas de IA	5
Vector manejador de IA	5
Vector manejador de física	5
Creación de controlador de nivel	5
Inserción de sonidos	6
Creación de panel de estado	6
Programación de flujo de pantallas	6

Tabla 9: Pila de producto para el desarrollo del videojuego

Según el análisis realizado, se ha determinado que los procesos de desarrollo del juego y del motor deben ser realizados en paralelo, ya que

existe una relación muy grande entre los dos productos. Así mismo, el desarrollo del juego servirá para realizar las pruebas de cada uno de los avances que se vayan suscitando en el desarrollo del motor.

3.3.DISEÑO DE ARQUITECTURA DEL SISTEMA

3.3.1. DOCUMENTO DE DISEÑO DE ARQUITECTURA

Introducción:

El presente documento tiene como finalidad describir la arquitectura general del sistema a ser desarrollado. Contiene documentos que describen de manera acertada el comportamiento de los distintos componentes del sistema y se lo ha dividido en 5 secciones principales: En la primera sección se encontrará la visión general del sistema, y las herramientas utilizadas para desarrollarlo. En la segunda sección se detalla el diagrama de clases con las respectivas relaciones que existen entre las clases del sistema, así como la estructura final de los archivos del sistema. En la tercera sección se describen cada una de estas clases por medio del diccionario. En la cuarta parte se detalla la matriz de interacción entre los diferentes objetos del juego. Finalmente, en la quinta sección se detallan cada uno de los sprints del proyecto, con sus respectivos documentos de objetivos de sprint.

Visión general del sistema:

El sistema a desarrollarse consta de dos partes principales:

- El motor de juego, que será utilizado para manejar todas las interacciones de dispositivos externos con el sistema operativo.

- El videojuego, que hará uso del motor previamente desarrollado, adaptando las reglas de juego definidas en el diseño de concepto.

Para el desarrollo de estos componentes se ha visto la necesidad de utilizar un lenguaje de bajo nivel, el cual permita controlar la interacción avanzada de dispositivos de hardware con el sistema operativo, así como un manejo a profundidad de los recursos del sistema. Es por esto que el lenguaje de programación escogido para este proyecto es C++. Adicionalmente, las herramientas a ser utilizadas en el desarrollo de la aplicación son:

- Compilador/Depurador: Microsoft Visual Studio 6.0
- Editor de imágenes: Adobe Photoshop CS2
- Editor de sonidos: Cool Edit 2000
- Diagramador: Rational Rose 2002

Diagrama de clases:

Motor:

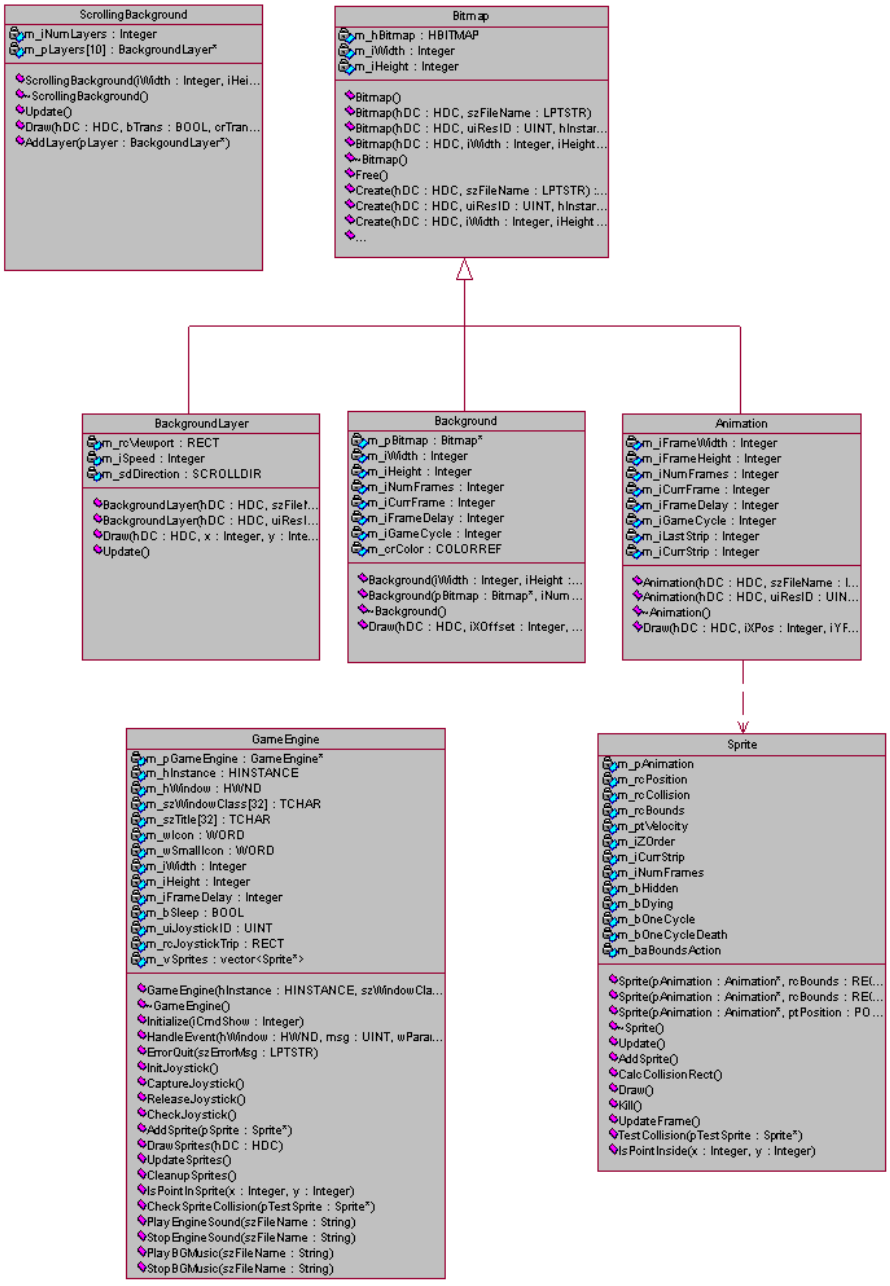

Figura 17: Diagrama de clases para el motor de juego

Juego:

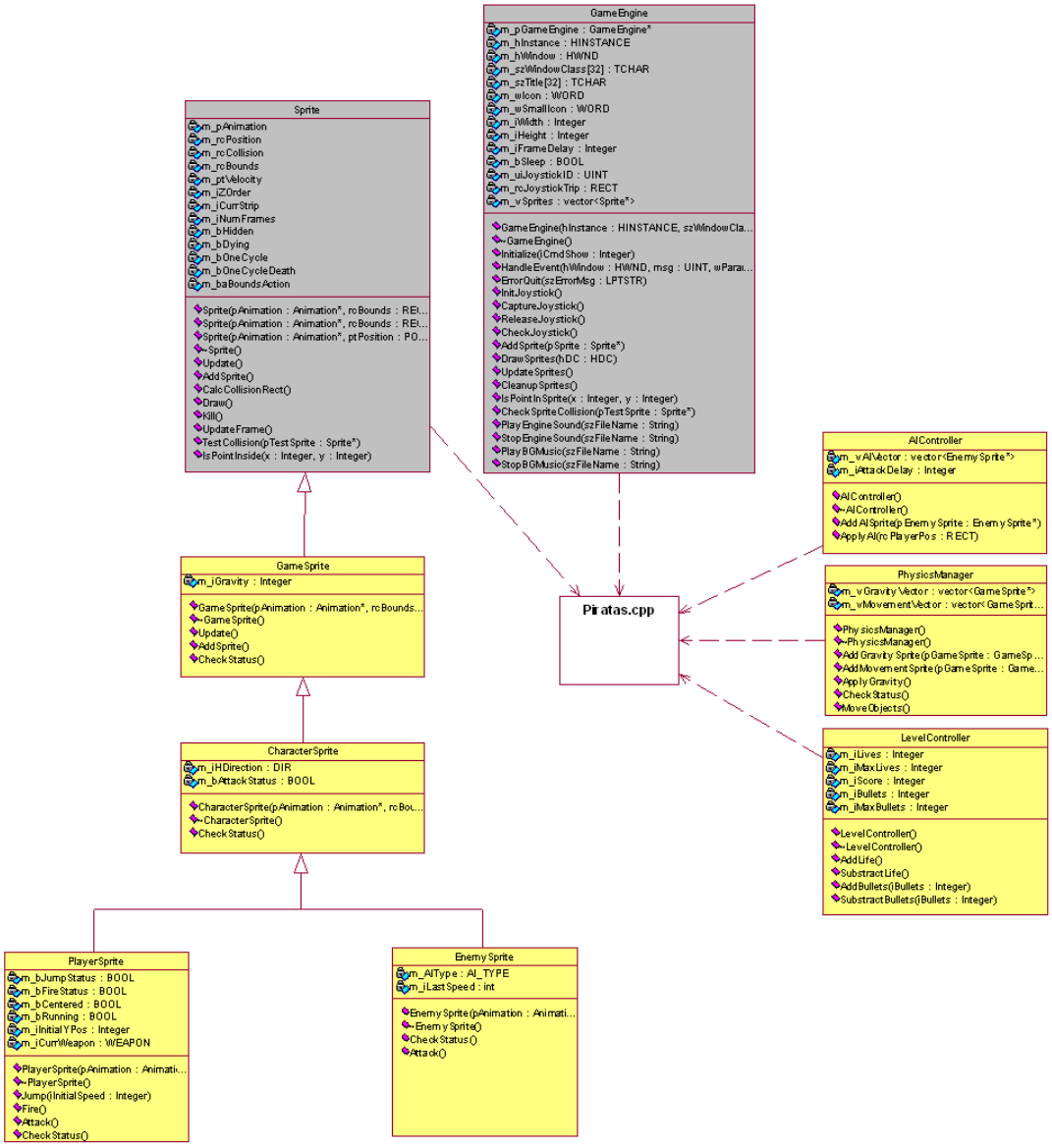

Figura 18: Diagrama de clases para el videojuego

Diccionario de clases:

Motor:

- **Clase Animation**

Clase Animation – Deriva de clase Bitmap
Maneja animaciones por medio de secuencias dentro de un mapa de bits. Estos mapas de bits pueden ser convertidos en recursos (los cuales son compilados con la aplicación), o ser directamente leídos desde el disco duro (Archivos de 24 bits). Dentro de los mismos pueden existir una o más secuencias (horizontales) con cada uno de los cuadros de la animación.

Atributos:
m_iFrameWidth: Ancho del cuadro de animación m_iFrameHeight: Alto del cuadro de animación m_iNumFrames: Número total de cuadros en la animación m_iCurrFrame: Número del cuadro actual m_iFrameDelay: Retardo entre cuadros de animación m_iGameCycle: Contador de ciclos de juego m_iLastStrip: Almacena la última secuencia de animación usada m_iCurrStrip: Almacena la secuencia de animación actual en la hoja

Constructores y métodos:
Animation(hDC, szFileName, iFrameWidth, iFrameHeight, iNumFrames, iFrameDelay): Crea una animación desde un archivo de mapa de bits situado en una ubicación del disco duro local. Animation(hDC, uiResID, hInstance, iFrameWidth, iFrameHeight, iNumFrames, iFrameDelay): Crea una animación a partir de un recurso de hoja de sprites compilado en conjunto con la aplicación y definido en el archivo de recursos. Draw(hDC, iXPos, iYPos, iCurrStrip, iNumFrames): Dibuja los cuadros de animación en secuencia para generar un efecto de movimiento.

- **Clase Background**

Clase Background – Deriva de clase Bitmap
Maneja fotos sin desplazamiento ni envolvimiento, sin embargo éstos pueden ser animados, en un caso similar a los objetos Animation. Provee el soporte para crear fondos de un solo color en la aplicación dados el ancho y el alto.

Atributos:
m_pBitmap: Puntero al mapa de bits de fondo m_iWidth: Ancho del fondo m_iHeight: Alto del fondo m_iNumFrames: Número de cuadros de la animación m_iCurrFrame: Número del cuadro actual m_iFrameDelay: Retardo entre cuadros de animación m_iGameCycle: Contador de ciclos de juego m_crColor: Color de fondo

Constructores y métodos:
Background(iWidth, iHeight, crColor): Crea un fondo de un solo color, con un alto y ancho determinados. Este fondo no tiene asignado ningún archivo de mapa de bits o un recurso. Background(pBitmap, iNumFrames, iFrameDelay): Crea un fondo animado a partir de un Objeto de tipo Bitmap. También se establece el número de cuadros de animación y el retardo entre cuadros. Draw(hDC, iXOffset, iYOffset): Dibuja los cuadros de animación en secuencia para generar un efecto de movimiento.

- **Clase BackgroundLayer**

Clase BackgroundLayer – Deriva de clase Bitmap
Crea una imagen de fondo desplazable. El conjunto de estas imágenes en unión con la transparencia de las mismas permiten el uso de capas en los fondos. Cada una de estas capas tiene su velocidad independiente de las otras, por lo que se pueden crear efectos de desplazamiento paralelo a diferentes velocidades.

Atributos:
m_rcViewport: Rectángulo que define la vista actual del fondo m_iSpeed: Velocidad de desplazamiento m_sdDirection: Dirección de movimiento

Constructores y métodos:
BackgroundLayer(hDC, szFileName, iSpeed, sdDirection): Crea una capa de fondo a partir de un archivo de mapa de bits situado en el disco duro local. BackgroundLayer(hDC, uiResID, hInstance, iSpeed, sdDirection): Crea una capa de fondo a partir de un identificador de recurso. Draw(hDC, x, y, bTrans, crTransColor): Dibuja la sección del fondo que se interseca con el rectángulo de vista. Update(): Actualiza posición de la capa de acuerdo a su velocidad.

- **Clase Bitmap**

Clase Bitmap
Clase que maneja el despliegue de mapas de bits en pantalla. Estos pueden ser archivos en el disco duro local o recursos compilados con el programa. Dichos elementos son imágenes con profundidad de color de 24 bits. Esta clase implementa la transparencia parcial de acuerdo a un color determinado.

Atributos:
m_hBitmap: Identificador del mapa de bits m_iWidth: Ancho del mapa de bits m_iHeight: Alto del mapa de bits

Constructores y métodos:
Bitmap(): Crea un mapa de bits sin parámetros Bitmap(hDC, szFilename): Crea un mapa de bits a partir de Create(hDC, szFileName) Bitmap(hDC, uiResID, hInstance): Crea un mapa de bits a partir de hDC, uiResID, hInstance) Bitmap(hDC, iWidth, iHeight, crColor): Create(hDC, iWidth, iHeight, crColor) Free():Elimina un objeto de tipo Bitmap Create(hDC, szFileName): Crea un mapa de bits a partir de un nombre de archivo situado en una ubicación local Create(hDC, uiResID, hInstance): Crea un mapa de bits a partir de un identificador de recurso definido Create(hDC, iWidth, iHeight, crColor): Crea un mapa de bits de un alto, ancho y color determinado Draw(hDC, x, y, bTrans, crTransColor): Dibuja el mapa de bits actual en una posición determinada. Aquí también se maneja la transparencia del dibujo. DrawPart(hDC, x, y, xPart, yPart, wPart, hPart, bTrans, crTransColor): Dibuja una parte del mapa de bits en una posición determinada. Este método es usado para dar soporte de hojas de sprites ya que dibuja los mapas de bits parcialmente.

- **Clase GameEngine**

Clase GameEngine
Clase principal del motor. Utilizada para instanciar un nuevo juego, manejar los sprites, colisiones, interacción de dispositivos, etc. Controla el estado de la aplicación y define métodos para manejo de eventos y mensajes de windows.

Atributos:
m_pGameEngine: Puntero estático al objeto del motor m_hInstance: Instancia de la aplicación m_hWindow: Ventana de la aplicación m_szWindowClass[32]: Clase de la ventana m_szTitle[32]: Título de la ventana m_wIcon: Icono grande de la aplicación m_wSmallIcon: Icono pequeño de la aplicación m_iWidth: Ancho de la ventana m_iHeight: Alto de la ventana m_iFrameDelay: Retardo entre cuadros de animaciones del juego m_bSleep: Variable de estado indica si el juego está inactivo o no m_uiJoystickID: Identificador de joystick en el motor m_rcJoystickTrip: Determina el rectángulo de efectividad del joystick m_vSprites: Vector manejador de sprites

Constructores y métodos:
GameEngine(hInstance, szWindowClass, szTitle, wIcon, wSmallIcon, iWidth, iHeight): Crea el objeto GameEngine, dándole los parámetros básicos de la ventana. Initialize(iCmdShow): Inicializa la ventana de juego HandleEvent(hWindow, msg, wParam, lParam): Función para manejo de mensajes de windows ErrorQuit(szErrorMsg): Imprime un mensaje de error en pantalla en caso de haber algún problema con los mensajes de windows

InitJoystick(): Inicializa el Joystick para su utilización

CaptureJoystick():Toma el control del joystick para que solo pueda ser usado por el motor cuando la aplicación esté activa

ReleaseJoystick():Libera al joystick para que pueda ser usado por otra aplicación, cuando la ventana principal del juego está inactiva.

CheckJoystick():Comprueba el movimiento y pulsaciones de botón del joystick.

AddSprite(pSprite): Añade un sprite al vector manejador

DrawSprites(hDC): Dibuja los sprites en el vector

UpdateSprites(): Actualiza los sprites en el vector

CleanupSprites(): Limpia el vector de sprites

IsPointInSprite(x, y): Comprueba si un punto está dentro del sprite

CheckSpriteCollision(pTestSprite): Revisa si el sprite choca con otros

PlayEngineSound(szFileName): Inicia un sonido

StopEngineSound(szFileName): Detiene un sonido

PlayBGMusic(szFileName): Inicia un sonido/música ambiental

StopBGMusic(szFileName) : Detiene un sonido/música ambiental

- **Clase ScrollingBackground**

Clase ScrollingBackground
Clase que maneja las diversas capas de fondo y las dibuja de acuerdo a su orden en el eje Z. Cada una de estas capas puede tener movimiento independiente y diferentes velocidades. Esto produce un efecto de “parallax scrolling”, donde las capas se mueven a diferentes velocidades para dar un efecto de tridimensionalidad

Atributos:
m_iNumLayers: Número de capas en el manejador de fondos m_pLayers[10]: Arreglo para colocación de capas

Constructores y métodos:
ScrollingBackground(iWidth, iHeight): Crea un manejador de fondos a partir de alto y ancho Update(): Actualiza posiciones de cada capa Draw(hDC, bTrans, crTransColor): Dibuja cada capa AddLayer(pLayer): Añade una capa al manejador

- **Clase Sprite**

Clase Sprite
Manejador de sprites. Estos se crean a partir de mapas de bits o animaciones e interactúan entre sí para definir las reglas del juego a ser creado. Entre otras cosas poseen ya una posición y un rectángulo para comprobar colisiones entre los mismos.

Atributos:
m_pAnimation: Puntero a la animación relacionada m_rcPosition: Posición del sprite m_rcCollision: Rectángulo de colisión del sprite m_rcBounds: Rectángulo limitador del sprite m_ptVelocity: Velocidad del sprite m_iZOrder: Orden eje Z del sprite (para dibujado en capas) m_iCurrStrip: Num. de secuencia de animación m_iNumFrames: Num. de cuadros en la secuencia actual m_bHidden: Indica si el sprite está escondido m_bDying: Variable que según su estado elimina o deja vivir al sprite m_bOneCycle: Variable que permite ejecutar un solo ciclo de animación m_bOneCycleDeath: Variable que elimina el sprite luego de un ciclo de animación m_baBoundsAction: Acción de colisión del sprite con su rectángulo limitador

Constructores y métodos:
Sprite(pAnimation, rcBounds): Crea un sprite a partir de una animación y su respectivo bounding box (rectángulo limitador) Sprite(pAnimation, rcBounds, baBoundsAction): Crea un sprite a partir de una animación y sus respectivos bounding box (rectángulo limitador) y propiedades de colisión con el mismo. Sprite(pAnimation, ptPosition, ptVelocity, iZorder, rcBounds, baBoundsAction): Crea un sprite Update(): Actualiza el sprite AddSprite(): Crea un sprite a partir de otro. CalcCollisionRect(): Calcula el rectángulo de colisión del sprite Draw(hDC, iCurrStrip, iNumFrames): Dibuja el sprite TestCollision(pTestSprite): Comprueba colisión entre sprites IsPointInside(x, y): Comprueba si un punto se encuentra dentro de un sprite

Juego

- **Clase AIController**

Clase AIController
Maneja la inteligencia artificial de los enemigos por medio de un vector de objetos, y un esquema de inteligencia artificial asignado a los mismos al momento de su creación. Se definen reglas individuales para cada uno de estos esquemas.

Atributos:
m_vAIVector: Vector de objetos a los que se aplica el esquema de IA m_iAttackDelay: Retardo entre ataques de enemigos

Constructores y métodos:
AIController(): Crea el objeto controlador de inteligencia artificial AddAISprite(pEnemySprite): Añade un enemigo al vector de AI para su manejo ApplyAI(): Aplica el esquema de AI individual definido en cada uno de los enemigos

- **Clase CharacterSprite**

Clase CharacterSprite – Deriva de GameSprite
Clase particular para manejo de sprites de personajes dentro del juego. Incluye prototipos de métodos para determinar la animación que se debe ejecutar dependiendo del status del personaje.

Atributos:
m_iHDirection: Dirección de movimiento horizontal del sprite m_bAttackStatus: Comprueba si el personaje está atacando

Constructores y métodos:
CharacterSprite(pAnimation, rcBounds, baBoundsAction, iGravity): Crea un personaje a partir de una animación. Recibe la gravedad como parámetro, para que el manejador de física pueda aplicar velocidades en el eje ‘y’ al mismo. CheckStatus(): Revisa el estado del sprite y de acuerdo a esto aplica una animación determinada.

- **Clase EnemySprite**

Clase EnemySprite – Deriva de CharacterSprite
Clase particular para manejo de sprites de enemigos dentro del juego. Incluye atributos para asignación de diversos esquemas de IA dependiendo del tipo de enemigo.

Atributos:
m_iAIType: Esquema de AI asignado al enemigo m_iLastSpeed: Ultima velocidad auxiliar (para ciertos métodos de ataque)

Constructores y métodos:
EnemySprite(pAnimation, rcBounds, baBoundsAction, iGravity, iAIType): Crea un sprite de enemigo

CheckStatus(): Revisa el estado del sprite y de acuerdo a esto aplica una animación determinada.

Attack(): Método que asigna la animación de ataque al sprite

- **Clase GameSprite**

Clase GameSprite – Deriva de Sprite
Clase particular para manejo de sprites en el juego. Estos a su vez pueden dividirse en sprites para personajes, objetos, power-ups, elementos de entorno destruibles, etc.

Atributos:
m_iGravity: Atributo que define la fuerza de gravedad para el objeto

Constructores y métodos:
GameSprite(pAnimation, rcBounds, baBoundsAction, iGravity): Crea un sprite de juego
Update(): Actualiza el sprite
AddSprite(): Agrega un sprite a partir de otro
CheckStatus(): Revisa el estado del sprite y de acuerdo a esto aplica una animación determinada.

- **Clase LevelController**

Clase LevelController
Clase que controla los datos generales del nivel, como puntaje, número de vidas del personaje, etc.

Atributos:
m_iLives: Vidas del personaje m_iMaxLives: Número máximo de vidas m_iScore: Puntaje del juego m_iBullets: Balas del personaje m_iMaxBullets: Número máximo de balas

Constructores y métodos:
LevelController(): Crea el controlador de nivel
AddLife(): Añade una vida al personaje
SubstractLife(): Sustrae una vida del personaje
AddBullets(iBullets): Añade un número determinado de balas al personaje
SubstractBullets(iBullets): Sustrae un número determinado de balas al personaje

- **Clase PhysicsManager**

Clase PhysicsManager
Clase que maneja las leyes físicas de cada uno de los sprites del juego. Esto se hace por medio de un vector de sprites, y propiedades dentro de los mismos, tales como el valor de la gravedad. Así mismo controla el desplazamiento relativo por el movimiento del personaje principal y los fondos del juego.

Atributos:
m_vGravityVector: Vector de física m_vMovementVector: Vector de movimiento relativo

Constructores y métodos:
PhysicsManager(): Crea un manejador de física AddGravitySprite(pGameSprite): Añade un objeto al vector de física AddMovementSprite(pGameSprite): Añade un objeto al vector de movimiento relativo ApplyGravity(): Aplica la gravedad a un objeto del vector de física CheckStatus(): Revisa el status de cada uno de los objetos MoveObjects(iOffset): Realiza el movimiento relativo de cada uno de los objetos

- **Clase PlayerSprite**

Clase PlayerSprite – Deriva de CharacterSprite
Clase que controla el sprite del jugador. Se definen métodos para saltos, ataques y demás movimientos, los cuales están apegados a las pulsaciones del teclado o del joystick.

Atributos:
m_bJumpStatus: Determina si el personaje está saltando m_bFireStatus: Determina si el personaje está disparando m_bCentered: Determina si el personaje está centrado en pantalla m_bRunning: Determina si el personaje está corriendo m_iInitialYPos: Almacena la posición en el eje ‘y’ antes de un salto m_iCurrWeapon: Determina el arma actual del personaje

Constructores y métodos:
PlayerSprite(pAnimation, rcBounds, baBoundsAction, iGravity): Crea al personaje principal Jump(int iInitialSpeed): Maneja los saltos del personaje principal Fire(): Método para disparo del personaje Attack(): Método para ataque del personaje CheckStatus():Revisa el estado del sprite y de acuerdo a esto aplica una animación determinada.

Matriz de interacción:

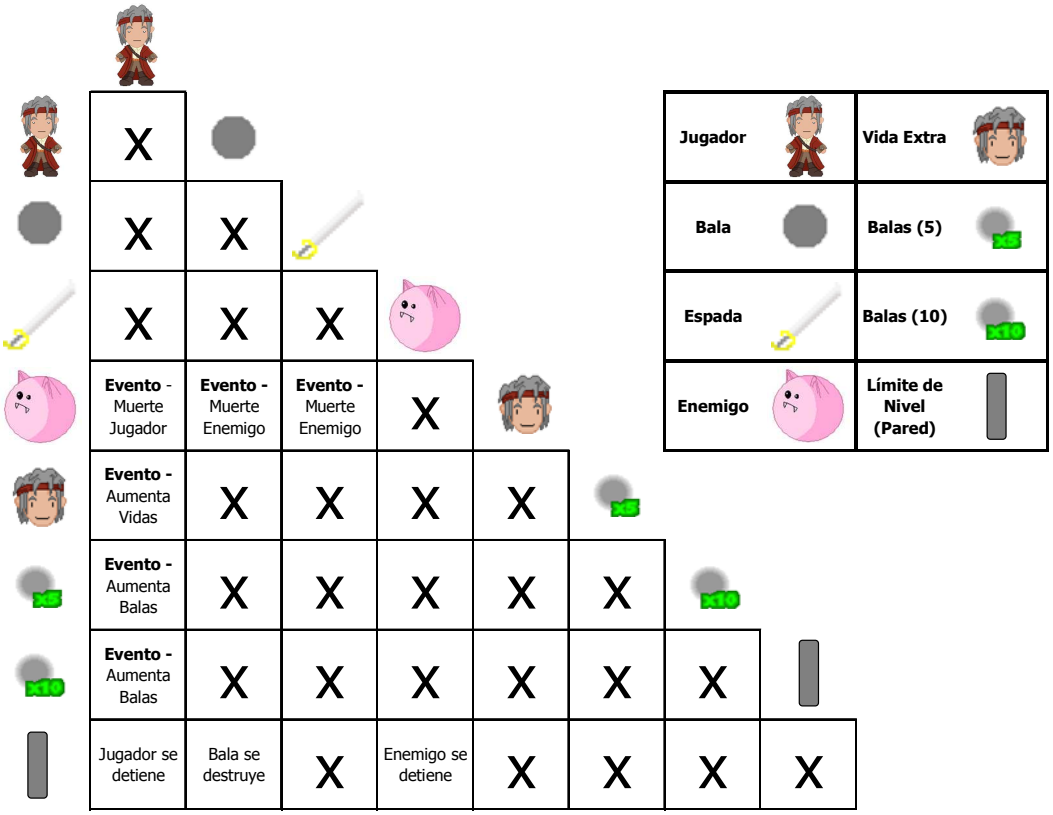

Figura 19: Matriz de interacción para el videojuego

Descripción de eventos:

Muerte Jugador:

- Muestra la animación de muerte del jugador
- Disminuye una vida

Muerte Enemigo:

- Muestra la animación de muerte del enemigo
- Aumenta el puntaje en 100
- Aleatoriamente hace aparecer un ítem
- Hace aparecer un nuevo enemigo

Aumenta vidas:

- Muestra la animación de destrucción del ítem

- Aumenta una vida al personaje principal

Aumenta balas:

- Muestra la animación de destrucción del ítem
- Aumenta un número definido de balas al personaje principal

Sprints del proyecto:

Número de Sprint: 1

Pila de Sprint:

Pila de Sprint - Número 1			
Motor		**Juego**	
Tarea	**Prioridad**	**Tarea**	**Prioridad**
Creación del motor	1	Creación de animaciones - Personaje principal	1
Dibujar imágenes estáticas en pantalla	1	Creación de animaciones - Enemigos	1
Soporte para transparencia	1	Creación de animaciones - Objetos	1
Dibujar animaciones en pantalla	1		

Tabla 10: Pila de sprint número 1

Fecha: 17-03-2008

Duración: 3 semanas

Descripción de tareas del sprint:

En este sprint se pretende alcanzar los siguientes objetivos:

- Crear la clase principal del motor. Esta clase controlará la interacción del juego con el sistema operativo. Incluirá métodos que son usados comúnmente en todo juego de video, tales como la creación de la ventana de la aplicación y/o el manejo de dispositivos.
- Incluir el soporte para imágenes de mapas de bits estáticas, que servirán como base para las animaciones. Así mismo se debe soportar transparencia en estas imágenes, de acuerdo a un color definido.

- Incluir el soporte para animaciones, en base a mapas de bits estáticos. Estos mapas de bits contendrán la secuencia completa de animación de un personaje u objeto del videojuego (comúnmente conocidos como hojas de sprites).
- Crear mapas de bits para el personaje principal, los enemigos y los objetos del videojuego. Se deben incluir todos los cuadros de las animaciones descritas

Número de Sprint: 2

Pila de sprint:

Pila de Sprint - Número 2			
Motor		**Juego**	
Tarea	**Prioridad**	**Tarea**	**Prioridad**
Dibujar fondos estáticos	2	Creación de fondos	2
Dibujar fondos con desplazamiento	2	Creación de logotipos y pantallas splash	2
Dibujar fondos en capas	2		

Tabla 11: Pila de sprint número 2

Documento de objetivo de Sprint

Fecha: 07-04-2008

Duración: 2 semanas

Descripción de tareas del sprint:

En este sprint se pretende alcanzar los siguientes objetivos:

- Incluir el soporte para el dibujado de fondos estáticos de un nivel del juego.
- Incluir el soporte para el dibujado de fondos con desplazamiento de un nivel del juego.
- Incluir un manejador de fondos en capas para dar la sensación de profundidad en los niveles del juego
- Crear los diferentes fondos para el producto y sus respectivas capas.

- Crear los logotipos de juego para su futuro uso
- Crear las pantallas de menú, de fin de juego y de carga para su futuro uso

Número de Sprint: 3

Pila de Sprint:

Pila de Sprint - Número 3			
Motor		**Juego**	
Tarea	**Prioridad**	**Tarea**	**Prioridad**
Manejar sprites	3	Creación de clase de sprites de juego	3
Manejar acciones de límite de pantalla	3	Creación de subclases de sprites de juego	3
Manejar colisiones entre sprites	3		

Tabla 12: Pila de sprint número 3

Documento de objetivo de Sprint

Fecha: 21-04-2008

Duración: 2 semanas

Descripción de tareas del sprint:

En este sprint se pretende alcanzar los siguientes objetivos:

- Incluir el soporte para sprites, los cuales utilizarán las imágenes previamente creadas para interactuar entre sí y definir las reglas del juego.
- Incluir el manejo de colisiones entre dos sprites, o entre un sprite y el rectángulo limitador en el nivel, lo cual permitirá hacer que los mismos interactúen entre sí
- Crear clases independientes para los diferentes tipos de sprites identificados en el juego, e implementar las diferencias entre las mismas.

Número de Sprint: 4

Pila de Sprint:

Pila de Sprint - Número 4			
Motor		**Juego**	
Tarea	**Prioridad**	**Tarea**	**Prioridad**
Manejar pulsaciones del teclado	4	Programación de movimientos de personaje principal	4
Manejar dispositivos de juego (joysticks)	4	Programación de movimientos de enemigos	4
		Programación de colisiones entre sprites del juego	4

Tabla 13: Pila de sprint número 4

Documento de objetivo de Sprint

Fecha: 05-05-2008

Duración: 2 semanas

Descripción de tareas del sprint:

En este sprint se pretende alcanzar los siguientes objetivos:

- Incluir el soporte para el manejo de dispositivos externos, tales como el teclado y el joystick.
- Programar los movimientos del personaje principal asociados a las acciones realizadas con el dispositivo de control.
- Programar los movimientos de los enemigos para que puedan interactuar con el personaje principal.
- Programar las colisiones entre sprites que son específicas para el juego, y que definirán las reglas del mismo.

Número de Sprint: 5

Pila de Sprint:

Pila de Sprint - Número 5			
Motor		**Juego**	
Tarea	**Prioridad**	**Tarea**	**Prioridad**
Manejar métodos de dibujo generales	5	Programación de esquemas de IA	5
Manejar métodos de actualización generales	5	Vector manejador de IA	5
		Vector manejador de física	5
		Creación de controlador de nivel	5

Tabla 14: Pila de sprint número 5

Documento de objetivo de Sprint

Fecha: 19-05-2008

Duración: 2 semanas

Descripción de tareas del sprint:

En este sprint se pretende alcanzar los siguientes objetivos:

- Crear un manejador de sprites, para que sean dibujados continuamente por el motor, y se revisen las colisiones entre ellos de forma automática.
- Programar los diversos esquemas de inteligencia artificial que serán asignados a los enemigos del videojuego.
- Crear un manejador de inteligencia artificial que haga los cálculos para cada enemigo de forma automática.
- Crear un controlador de física para la aplicación de leyes de cinemática a los personajes y objetos del juego, para simular un ambiente físico real.
- Crear un controlador de nivel que lleve cuenta de puntajes, número de vidas, etc., y que modifique estos valores de acuerdo a los eventos del juego.

Número de Sprint: 6

Pila de Sprint:

Pila de Sprint - Número 6			
Motor		**Juego**	
Tarea	**Prioridad**	**Tarea**	**Prioridad**
Manejar sonidos y música de fondo	6	Inserción de sonidos	6
		Creación de panel de estado	6
		Programación de flujo de pantallas	6

Tabla 15: Pila de sprint número 6

Documento de objetivo de Sprint

Fecha: 02-06-2008
Duración: 2 semanas

Descripción de tareas del sprint:
En este sprint se pretende alcanzar los siguientes objetivos:

- Incluir el soporte para la inserción de sonidos y música de fondo dentro del videojuego.
- Crear o conseguir sonidos para los diversos eventos del videojuego.
- Crear el panel de estado del juego, el cual indicará datos importantes sobre el nivel, tales como número de vidas, puntaje, etc.
- Programar el flujo de pantallas de presentación, carga y fin de juego.

4. IMPLEMENTACIÓN, PRUEBAS Y EVALUACIÓN

4.1. DESARROLLO DEL MOTOR GRÁFICO

Para el desarrollo del motor gráfico se han tomado los sprints definidos en el documento de diseño de arquitectura del sistema, los cuales serán detallados en esta sección.

Número de Sprint: 1

Pila de Sprint:

Motor	
Tarea	**Prioridad**
Creación del motor	1
Dibujar imágenes estáticas en pantalla	1
Soporte para transparencia	1
Dibujar animaciones en pantalla	1

Tabla 16: Pila de sprint del motor – Número 1

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- Se creó la clase principal del motor, con nombre *GameEngine*, la cual implementa funciones generales para el desarrollo de cualquier videojuego. Esta clase permite:
  - Controlar la creación de la ventana principal del juego, y establecer un título, íconos, y tamaño para la misma.
  - Manejar funciones de inicio, fin, pausa, paso a primer y segundo plano e inicialización de componentes del juego.
  - Manejar mensajes estándar de eventos de Windows
- Se implementó la clase *Bitmap*, que dibuja imágenes estáticas en pantalla mediante los métodos *Draw* (dibuja la imagen completa) y *DrawPart* (dibuja parte de una imagen). Esta clase puede tomar

tanto mapas de bits ubicados en el disco duro de forma dinámica en tiempo de ejecución, o tomar imágenes definidas como recurso (en los archivos Resource.h y Piratas.rc), las cuales son compiladas en conjunto con el archivo ejecutable de la aplicación. Dentro de esta clase, se invocan a funciones del sistema operativo, las cuales permiten entre otras cosas definir un color para simular transparencia en un mapa de bits. Es así que se define el Magenta (RGB 255, 0, 255) como el color transparente para todas las imágenes del videojuego.

```
void Bitmap::DrawPart(HDC hDC, int x, int y, int xPart, int yPart,
                int wPart, int hPart, BOOL bTrans, COLORREF crTransColor)
{
      if (m_hBitmap != NULL)
      {
            HDC hMemDC = CreateCompatibleDC(hDC);

            // Seleccionar el mapa de bits en el device context
            HBITMAP hOldBitmap = (HBITMAP)SelectObject(hMemDC, m_hBitmap);

            // Dibuja el mapa de bits en el device context
            if (bTrans)
                  TransparentBlt(hDC, x, y, wPart, hPart, hMemDC, xPart,
                  yPart, wPart, hPart, crTransColor);
            else
                  BitBlt(hDC, x, y, wPart, hPart, hMemDC, xPart, yPart,
                  SRCCOPY);

            // Restaurar y eliminar el device context
            SelectObject(hMemDC, hOldBitmap);
            DeleteDC(hMemDC);
      }
}
```

- A partir de la clase *Bitmap* se creó la clase *Animation*, que tan solo tiene el método *Draw*. Sin embargo, se sobrecargó el método de la clase padre para que dada una imagen de mapa de bits con la secuencia de animación completa, éste pueda dibujar cada uno de los cuadros de la animación por separado, generando el efecto de movimiento.

Número de Sprint: 2

Pila de Sprint:

Motor	
Tarea	**Prioridad**
Dibujar fondos estáticos	2
Dibujar fondos con desplazamiento	2
Dibujar fondos en capas	2

Tabla 17: Pila de sprint del motor – Número 2

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- A partir de la clase *Bitmap*, se desarrolló *Background*, la cual maneja fondos estáticos de la aplicación. Como se ha mencionado, estos fondos no tienen animación ni desplazamiento de ningún tipo.
- Para dibujar fondos con desplazamiento, se ha derivado una clase *BackgroundLayer* de la clase *Bitmap*. Esta dibuja un fondo, pero le asigna propiedades de velocidad, con lo cual el fondo puede desplazarse, para generar una ilusión de movimiento en el primer plano.
- Para el manejo de diversas capas en el fondo de la aplicación, se ha creado la clase *ScrollingBackground*, la cual agrupa objetos de tipo *BackgroundLayer*, y los dibuja uno por uno. Almacena estos objetos en un arreglo y con su método *Update* barre el mismo, invocando a los métodos *Draw* individuales de cada una de las capas. Los objetos son insertados al arreglo por medio de la función *AddLayer*.

Número de Sprint: 3

Pila de Sprint:

Motor	
Tarea	**Prioridad**
Manejar sprites	3
Manejar acciones de límite de pantalla	3
Manejar colisiones entre sprites	3

Tabla 18: Pila de sprint del motor – Número 3

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- Se desarrolló una clase *Sprite*, la cual encapsula una animación y le da diferentes propiedades para que pueda interactuar con otros sprites. Algunas de estas propiedades son:
    - Velocidad
    - Posición
    - Rectángulo de colisión
    - Número de cuadros de la animación asociada
    - Rectángulo limitador del sprite en la aplicación

  Por medio del método *Update*, cada sprite actualiza su posición dependiendo de la velocidad asignada. Igualmente, la clase contiene un método *Draw*, que dibuja la animación asociada al sprite. Para el cálculo posterior de colisiones, se ha desarrollado un método llamado *CalcCollisionRect*, el cual hace cálculos sobre el rectángulo de posición del sprite, y le asigna un rectángulo de colisión, lo que hará posible los choques con otros objetos y la definición de reglas del juego.
- En la clase *Sprite*, dentro del método *Update*, se han definido acciones estándar para el choque del objeto con el rectángulo limitador definido en el mismo. Estas acciones son de parada

(BA_STOP), de envolvimiento (BA_WRAP), de rebote (BA_BOUNCE) y de muerte (BA_DIE).

- Dentro de la clase *GameEngine* se ha definido el método *CheckSpriteCollision*, el cual comprueba la colisión de los diferentes sprites del juego. Para esto, se ha definido un vector de Sprites dentro de la clase, al cual se agregan cada uno de los objetos, para su posterior barrido con dicho método.

Número de Sprint: 4

Pila de Sprint:

Motor	
Tarea	**Prioridad**
Manejar pulsaciones del teclado	4
Manejar dispositivos de juego (joysticks)	4

Tabla 19: Pila de sprint del motor – Número 4

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- Dentro de la clase *GameEngine*, se agregó el método *HandleKeys*, el cual controla las pulsaciones del teclado. Sin embargo, en esta clase el método está vacío, ya que la implementación del mismo es independiente para cada juego que se desarrolle.
- Se agregaron métodos a la clase *GameEngine* para el control de un Joystick o Gamepad. Estas clases se encargan de inicializar el dispositivo, o de capturarlo o liberarlo dependiendo del plano en que se encuentra la aplicación. Un método importante es *CheckJoystick*, ya que se encarga de manejar las pulsaciones de botones en el dispositivo.

Número de Sprint: 5

Pila de Sprint:

Motor	
Tarea	**Prioridad**
Manejar métodos de dibujo generales	5
Manejar métodos de actualización generales	5

Tabla 20: Pila de sprint del motor – Número 5

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- Dentro de la clase *GameEngine*, se agregó el método *DrawSprites*, el cual mediante el vector que fue previamente creado para la implementación del método *CheckSpriteCollision*, llama al método *Draw* de cada uno de los objetos.

```
void GameEngine::DrawSprites(HDC hDC)
{
	// Itera en el vector y dibuja los sprites
	vector<Sprite*>::iterator siSprite;
	for (siSprite = m_vSprites.begin(); siSprite != m_vSprites.end();
	siSprite++)
		(*siSprite)->Draw(hDC, (*siSprite)->GetCurrStrip(),
		(*siSprite)->GetNumFrames());
}
```

- Así mismo, se definió un método *UpdateSprites*, el cual tiene un esquema similar al método *DrawSprites*, ya que barre el vector de sprites, llamando a los métodos *Update* de cada uno de los objetos.

Número de Sprint: 6

Pila de Sprint:

Motor	
Tarea	**Prioridad**
Manejar sonidos y música de fondo	6

Tabla 21: Pila de sprint del motor – Número 6

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- Se han creado dentro de la clase *GameEngine* cuatro métodos para la reproducción y parada de música y sonidos. Estos métodos hacen uso de la funcionalidad de la interfaz de control multimedia de Windows (MCI), para llamar a archivos de sonido en tiempo de ejecución de la aplicación.[10]

4.2. DESARROLLO DEL VIDEOJUEGO

Para el desarrollo del juego se han tomado los sprints definidos en el documento de diseño de arquitectura del sistema, los cuales serán detallados en esta sección.

Número de Sprint: 1

Pila de Sprint:

Juego	
Tarea	**Prioridad**
Creación de animaciones - Personaje principal	1
Creación de animaciones - Enemigos	1
Creación de animaciones - Objetos	1

Tabla 22: Pila de sprint del videojuego – Número 1

[10] *GameDev.net - Using MCI for MP3 Playback* – Rob Zimmerman

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se han creado las imágenes descritas en la figura 20.

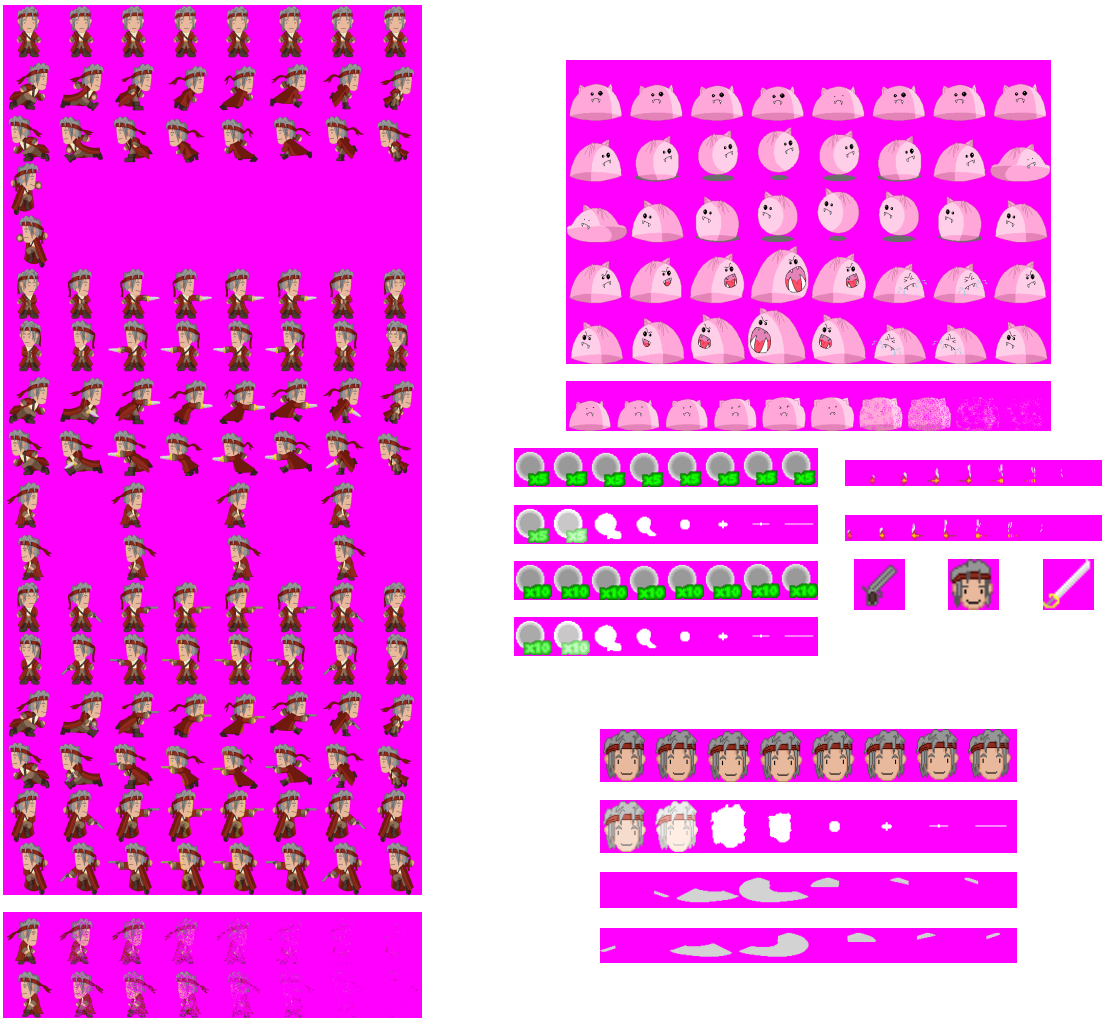

Figura 20: Arte de personajes y objetos para el juego

Número de Sprint: 2

Pila de Sprint:

Juego	
Tarea	**Prioridad**
Creación de fondos	2
Creación de logotipos y pantallas splash	2

Tabla 23: Pila de sprint del videojuego – Número 2

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se han creado las imágenes descritas en la figura 21.

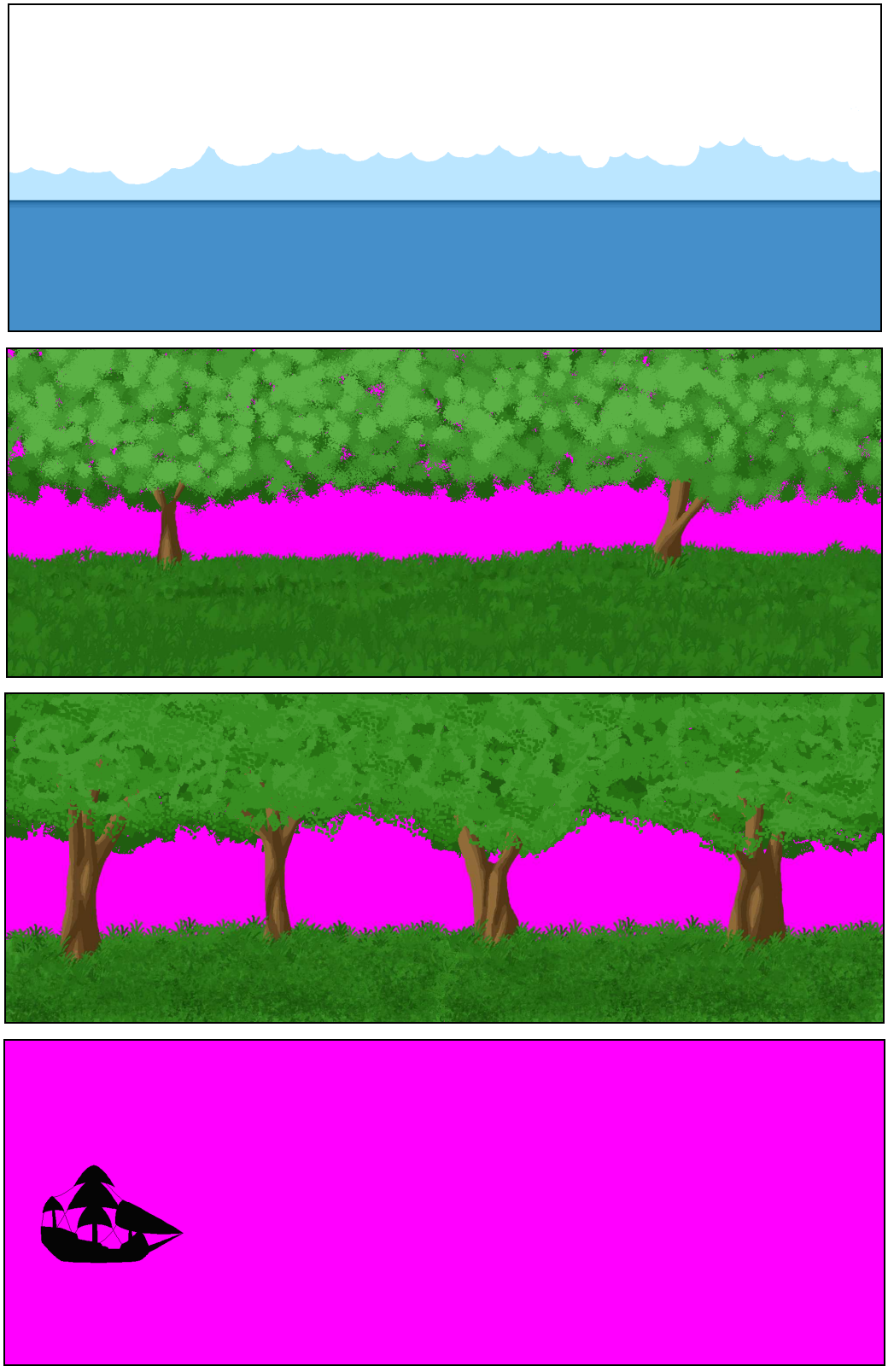

Figura 21: Arte de fondos y logotipos para el juego

Número de Sprint: 3

Pila de Sprint:

Juego	
Tarea	**Prioridad**
Creación de clase de sprites de juego	3
Creación de subclases de sprites de juego	3

Tabla 24: Pila de sprint del videojuego – Número 3

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- Se desarrolló una clase *GameSprite*, la cual se deriva de la clase *Sprite* y le da propiedades adicionales, tal como un valor para la gravedad. Esta es la clase básica para el árbol de sprites en el juego.

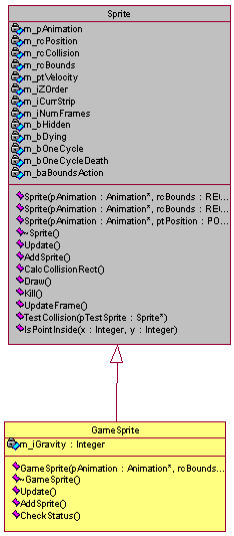

Figura 22: Diagrama de clase Sprite

- Se definió el árbol de clases para sprites del juego, incluyendo una clase para sprites de personajes (*CharacterSprite*), enemigos (*EnemySprite*) y el personaje principal (*PlayerSprite*). Cada una de estas clases tiene métodos y atributos diferentes, para realizar una tarea específica del tipo de sprite. Cabe destacar que en la clase *PlayerSprite* se incluyen los métodos para que el personaje principal pueda saltar o atacar a los enemigos.

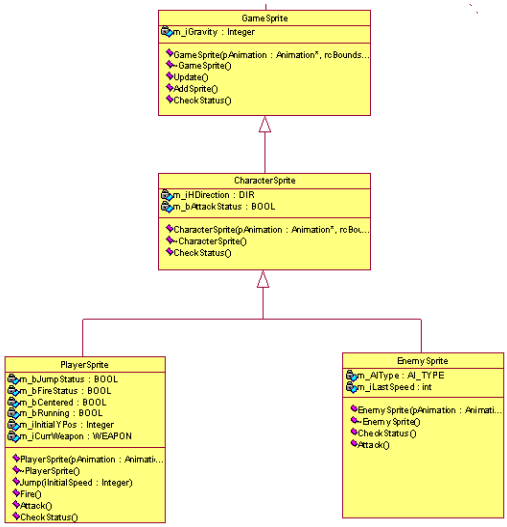

Figura 22: Árbol de clases para personajes

Número de Sprint: 4

Pila de Sprint:

Juego	
Tarea	**Prioridad**
Programación de movimientos de personaje principal	4
Programación de movimientos de enemigos	4
Programación de colisiones entre sprites del juego	4

Tabla 25: Pila de sprint del videojuego – Número 4

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- Dentro de las clases *PlayerSprite* y *EnemySprite* se agregó un método *CheckStatus*, el cual se encarga de determinar el estado actual del personaje (corriendo, saltando o atacando) para aplicar la animación correspondiente al mismo.
- En el archivo de programa principal *Piratas.cpp* se añadieron los métodos de pulsaciones de teclas y botones del dispositivo de control, los cuales fueron enlazados con los diferentes estados del personaje principal. Es decir, al momento de pulsar un botón, el motor cambia el estado del sprite de acuerdo a la tecla pulsada y al estado actual del mismo.
- En la clase *PlayerSprite* se implementaron los métodos *Jump, Fire* y *Attack*, los cuales se encargan tanto de actualizar el estado del personaje, como de realizar una acción específica, como un cambio de velocidad o de animación. Estas acciones están en concordancia con los movimientos del personaje principal definidos en los documentos de concepto del juego. Estos métodos hacen uso de las leyes de la cinemática básica.[11]

[11] BOURG, David; *Physics for Game Developers* – LENGYEL, Eric; *Mathematics for Game Programming & Computer Graphics*

```
PlayerSprite::Jump(int iInitialSpeed)
{
	if (!m_bJumpStatus)
	{
		m_bJumpStatus = true;
		m_iInitialYPos = GetPosition().bottom;
		SetPosition(GetPosition().left, GetPosition().top - 10);
		SetVelocity(GetVelocity().x, -iInitialSpeed);
	}

}
```

- En el archivo de programa principal, se implementaron las funciones *SpriteCollision* y *SpriteDying*, que controlan la colisión entre dos sprites y la eliminación de los mismos respectivamente.

Número de Sprint: 5

Pila de Sprint:

Juego	
Tarea	**Prioridad**
Programación de esquemas de IA	5
Vector manejador de IA	5
Vector manejador de física	5
Creación de controlador de nivel	5

Tabla 26: Pila de sprint del videojuego – Número 5

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- Se creó la clase *AIController*, la cual define los diferentes esquemas de inteligencia artificial que un enemigo puede tener. Para esto, se añadió un atributo en la clase *EnemySprite*, el cual indica el tipo de esquema asociado al enemigo. La clase *AIController* se encarga de barrer un vector que contiene a los enemigos, para aplicar uno por uno el esquema de inteligencia artificial definido en el objeto.

- Se definieron dos esquemas básicos de Inteligencia Artificial[12]: AI_NONE, que es utilizado para que el enemigo no tenga movimiento, y AI_BLOBB, que compara la posición actual del enemigo con la del personaje principal y ocasiona que éste se acerque al jugador de una manera progresiva y controlada. También define el ataque del enemigo cuando se encuentra a una distancia muy pequeña del jugador.
- Se creó la clase *PhysicsManager*, la cual maneja las leyes físicas del juego. Funciona de una manera similar a la clase *AIController*, ya que también consta de un vector de objetos, sobre los cuales se aplican las operaciones de cambio de velocidad o posición de forma independiente.
- Se desarrolló la clase *LevelController*, la cual maneja aspectos generales de un nivel, tales como el puntaje, las vidas del personaje principal y su munición.

Número de Sprint: 6

Pila de Sprint:

Juego	
Tarea	**Prioridad**
Inserción de sonidos	6
Creación de panel de estado	6
Programación de flujo de pantallas	6

Tabla 27: Pila de sprint del videojuego – Número 6

Elementos desarrollados en el sprint:

Para cumplir los objetivos planteados en el sprint actual se desarrollaron las siguientes funcionalidades:

- Dentro del archivo principal de la aplicación se ingresaron los diferentes sonidos asociados a eventos del juego. Esto se hizo en

[12] BUCKLAND, Mat; *AI Techniques for Game Programming*

diversas secciones del archivo, ya que los eventos pueden ocurrir en diferentes partes del código.

- Se definió la programación para el panel de estado superior, integrando la información de la clase *LevelController* con la interfaz gráfica del juego. Es así que se incluyeron íconos creados en el primer sprint dentro del desarrollo del sprint actual.
- Se creó el método *NewGame* en el archivo principal de la aplicación. Éste permite la creación de los objetos para el juego cada vez que sea necesario reiniciarlo. Adicionalmente se creó el flujo de pantallas y sus respectivas transiciones (pantallas de presentación, carga del juego, nivel principal y fin de juego)

4.3.PRUEBAS

En esta sección se desplegarán los documentos de reporte de errores o bugs del sistema, con la respectiva solución implementada.

Código:	PTC-0001
Nombre del probador:	Luis Fernando Alvarez A.

Fecha:	02-06-2008

Descripción del error:
En ciertas ocasiones, al matar a un enemigo, el juego se congela sin mostrar la animación de muerte del mismo.

Solución al problema:	**Fecha:** 04-06-2008
Se implementó una mejor solución para la eliminación de sprites, ya que con el esquema anterior los elementos se eliminaban de la memoria, pero los vectores de actualización todavía tenían el puntero a ese objeto almacenado en su lista. Esto ocasionaba que el programa trate de leer una ubicación de memoria vacía al momento de barrer el vector.	

Código:	PTC-0002
Nombre del probador:	Luis Fernando Alvarez A.

Fecha:	03-06-2008

Descripción del error:
La animación del personaje principal cuando dispara hacia el lado izquierdo no se está desplegando correctamente, ya que muestra la animación de disparo hacia la derecha. La bala se genera al lado izquierdo del personaje, lo cual es correcto, pero no corresponde con la animación.

Solución al problema:	**Fecha:** 03-06-2008
Se cambió la animación de disparo del personaje. La animación correcta fue asignada	

Código:	PTC-0003
Nombre del probador:	Esteban Alvarez

Fecha:	10-06-2008

Descripción del error:
El juego permite al personaje tener una cantidad negativa de balas, lo cual es incorrecto.

Solución al problema:	**Fecha:** 10-06-2008
Se implementó un control para que el personaje no pueda disparar su arma cuando la cantidad de balas sea menor o igual que 0	

Código:	PTC-0004
Nombre del probador:	Esteban Alvarez

Fecha:	10-06-2008

Descripción del error:
Al pasar aproximadamente 5 minutos del juego, el tercer fondo (nubes) desaparece completamente. Las otras dos capas de fondo quedan intactas.

Solución al problema:	**Fecha:** 11-06-2008
Existía un error con el método de dibujado de los fondos desplazables, para lo cual se mejoró el método y se añadieron más controles.	

Código:	PTC-0005
Nombre del probador:	Luis Fernando Alvarez A.

Fecha:	10-06-2008

Descripción del error:
El personaje principal todavía tenía 2 vidas extra. Sin embargo al chocar contra un enemigo el juego finalizó inesperadamente, tal como si se hubieran agotado las vidas.

Solución al problema:	**Fecha:** 10-06-2008
Existía un error en el dibujado del número de vidas en el panel superior, por lo cual en ciertos momentos se estaban desplegando más vidas de las que en realidad tenía el personaje. Este error fue corregido en el método de dibujado de vidas.	

4.4.EVALUACIÓN DEL PRODUCTO

De acuerdo al producto final obtenido en este proyecto, la evaluación del mismo determina que:

- Es viable continuar con el proyecto para desarrollar el juego completo, ya que se cuenta con un diseño de concepto muy detallado y la idea no ha sido explotada con anterioridad.

- En caso de continuar con este proyecto se debe realizar una mejora en el aspecto gráfico del juego, ya que para ser competitivo en el mercado debe contar con una interfaz gráfica agradable y moderna, tomando en cuenta la posible fusión del motor actual con tecnologías de gráficos tridimensionales.

- El desarrollo del juego sería una tarea relativamente rápida, ya que al contar con la estructura del motor ya desarrollada, el proyecto se vería reducido al desarrollo de la lógica de reglas de juego y a la inclusión de los elementos multimedia.

5. CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

Luego de haber desarrollado el videojuego Piratas: La Tumba del Corsario, con su respectivo motor, se ha llegado a las siguientes conclusiones.

- El desarrollo de un motor es una tarea implícita en el desarrollo de un videojuego, y representa aproximadamente un 70% del tiempo total de la fase de implementación del proyecto. Por esta razón para el desarrollo de un videojuego, se ha determinado que utilizar un motor previamente existente es una alternativa que trae beneficios tanto en costos como en tiempos.

- Scrum es una metodología muy ágil y flexible, ya que al no requerir una documentación obligatoria provee al equipo de desarrollo la capacidad de generar la documentación estrictamente necesaria para la implementación del proyecto. Así mismo, la metodología propone un esquema de trabajo que propicia a la comunicación entre los miembros del equipo y el cliente, permitiendo que los requerimientos iniciales se llevan al producto final de manera adecuada. De acuerdo a estas características se puede decir que Scrum es una metodología adecuada para el desarrollo de productos de este tipo.

- Indudablemente en un producto de este tipo la interfaz gráfica, incluyendo elementos como sonidos y música ambiental, juega un papel muy importante en la aceptación del mismo en el mercado. Sin embargo, en los últimos años se ha hecho énfasis en este aspecto, descuidando el diseño de concepto de los videojuegos. Se debe encontrar un equilibrio entre el aspecto audiovisual y la

manejabilidad de los productos desarrollados para obtener así un interés más grande por parte del mercado.

- Usualmente los desarrolladores de software concentran sus esfuerzos en el correcto funcionamiento de la aplicación, mas no toman en cuenta la correcta utilización de los recursos del sistema. En el desarrollo de un motor gráfico, este aspecto es crucial, ya que finalmente este será utilizado para producir videojuegos que aprovechen al máximo la capacidad de procesamiento del hardware.

- Al diseñar un motor para el desarrollo de videojuegos, se debe fundar una estructura generalizada, que abarque funciones comunes a la mayoría de géneros de videojuegos. De esta forma, los desarrolladores de nuevos videojuegos pueden generar productos de manera rápida y abarcar un rango más extenso de diseños de concepto.

- La compatibilidad del producto final con una gran gama de dispositivos de hardware, de diferentes fabricantes y con diferentes tecnologías subyacentes es extremadamente importante en el desarrollo de un videojuego. Al tener un producto que sea soportado por una mayor cantidad de dispositivos de hardware, la comercialización del mismo será mucho más efectiva y traerá más beneficios económicos a la empresa desarrolladora.

- El manejo imágenes es primordial dentro del desarrollo de un videojuego, ya que el aspecto gráfico juega un papel muy importante en la industria. Si bien el esquema de manejo de mapas de bits del trabajo desarrollado no trae optimizaciones de ningún tipo en, se debe resaltar la facilidad de uso de los métodos de dibujo del motor, lo que agilita el proceso de desarrollo

- El lenguaje de programación utilizado para el desarrollo fue C++. La ventaja primordial de éste fue el control avanzado que se tiene sobre las funciones del sistema operativo. Sin embargo se debe destacar la alta curva de aprendizaje que tiene este lenguaje de programación.

- Las pruebas son un aspecto muy importante en el desarrollo de un videojuego y deben estar propiciamente documentadas, ya que van de la mano con la corrección los errores de la aplicación y la toma de nuevos requerimientos por la demanda del público.

5.2.RECOMENDACIONES

Las recomendaciones obtenidas en base al desarrollo del presente proyecto son:

- Para el desarrollo de un motor es recomendable utilizar un lenguaje de programación de bajo nivel. Si bien éstos tienen una curva de aprendizaje muy elevada, proveen al programador con un control total sobre los dispositivos del sistema, y aspectos tan importantes como el control de memoria o procesamiento.

- El motor es un conjunto de componentes que sirven de interfaz entre el programador y las funciones de control de hardware del sistema operativo. Es por esto que sería de gran utilidad generar una herramienta visual de desarrollo de juegos en base al motor subyacente, lo que haría aún más ágil al proceso de desarrollo.

- Por la alta informalidad que existe en el campo del desarrollo de juegos de video, es recomendable utilizar una metodología ágil y flexible, que no requiera de una alta generación documental.

- Un motor es una herramienta que trae una alta rentabilidad, tanto a la empresa desarrolladora del mismo como a los equipos de desarrollo que lo utilizan. Se debe tratar de plantear este campo del desarrollo de la informática en nuestro país, ya que esto nos traería grandes beneficios económicos a nivel mundial.

BIBLIOGRAFÍA

Libros:

1. BOURG, David; *Physics for Game Developers*; 1a. Edición; Editorial O'Reilly Media; U.S.A; 2001

2. BUCKLAND, Mat; *AI Techniques for Game Programming*; 1a. Edición; Editorial Course Technology PTR; U.S.A; 2002

3. DAWSON, Michael; *Beginning C++ Game Programming*; 1a. Edición; Editorial Course Technology PTR; U.S.A; 2004

4. GREEN, Jeff; FARKAS, Bart; *The Art of Warcraft*; 1a. Edición; Editorial BradyGames; 2002

5. LENGYEL, Eric; *Mathematics for Game Programming & Computer Graphics*; 2a. Edición; Editorial Charles River Media; U.S.A., 2002

6. PALACIO, Juan; *Flexibilidad con Scrum: Principios de Diseño e implantación de campos de Scrum;* 2ª. Edición; 2007

7. ROLLINGS, Andrew; MORRIS, Dave; *Game Architecture and Design*; 1a. Edición; Editorial Coriolis; U.S.A; 2000

Páginas Web:

1. List of Game Engines – Wikipedia: The Free Encyclopedia
   http://en.wikipedia.org/wiki/List_of_game_engines

2. GameDev.net - Using MCI for MP3 Playback – Rob Zimmerman
   http://www.gamedev.net/reference/articles/article2053.asp