UNIVERSIDAD DEL BÍO BÍO

FACULTAD DE CIENCIAS EMPRESARIALES

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN Y TECNOLOGÍAS DE LA INFORMACIÓN

DESARROLLO DE JUEGO 3D PARA PLATAFORMA ANDROID

PABLO CHRISTIAN ALVAREZ TORO

MARÍA FERNANDA DE LA HOZ SÁNCHEZ

Memoria para optar al título de
Ingeniero Civil en Computación e Informática

CHILLÁN, DICIEMBRE 2010

RESUMEN

En la actualidad el gran crecimiento en el número de dispositivos móviles es innegable, siendo Chile uno de los países con mayor número de dispositivos por persona 19.388.000 celulares (prácticamente uno por habitante). Este crecimiento va de la mano con el desarrollo de nuevas tecnologías y el consecuente aumento en las capacidades de los móviles.

Este proyecto toma como motivación este crecimiento de las tecnologías móviles, así como también los nuevos mercados que se abren para el área de desarrollo de software. Esta motivación está sustentada en la masificación de los dispositivos móviles y en la escasez de proyectos en nuestra universidad centrados en esta área, existiendo sólo uno, el cual ya no está acorde a las tecnologías y dispositivos utilizados actualmente. También como motivación se toma el desarrollar un software para el popular sistema operativo Android, que es orientado a dispositivos móviles como Smartphones o Tablets y es propiedad de Google.

Debido a lo anteriormente expuesto, se definió como objetivo del proyecto el "Diseñar e implementar un juego estilo plataforma en un entorno 3D para dispositivos móviles con sistema operativo Android", específicamente desarrollar un juego para Smartphones. Para tal propósito se utilizó la metodología de desarrollo de software Iterativa Incremental, considerando dos incrementos, los cuales cuentan con dos actividades generales: Diseño e Implementación. Para el diseño se utilizó UML o Lenguaje Unificado de Modelado, el cual permite visualizar, especificar, construir y documentar un sistema. Para la implementación, al no existir framework de alto nivel, se utilizó jPCT-AE (librería externa utilizada en la creación de videojuegos para Android) junto con OpenGL ES (librería que permite el soporte para gráficos 3D en dispositivos tales como Smartphones y consolas de videojuegos). Además, debido a las limitantes en cuanto a memoria existentes en los dispositivos, se aplicó técnicas de programación avanzadas: Billboard, Caché y Multithreading.

Finalmente, como resultado, se obtuvo un juego 3D completo, el cual demuestra tanto las capacidades de los dispositivos, como lo que se puede llegar a lograr en el ámbito de desarrollo de software para dispositivos móviles.

Tabla de contenido

Índice de figuras

Índice de tablas

PRIMER CAPÍTULO – MARCO INTRODUCTORIO

1.1. Introducción

En la actualidad la creciente demanda de información a llevado al desarrollo de las tecnologías en dispositivos móviles, especialmente los teléfonos celulares, haciendo que estos pasen de ser sólo utilizados para hacer o recibir llamadas, a aparatos en los cuales se puede realizar infinidad de tareas, como: leer el diario, mandar un mail, llevar una agenda, jugar, y muchas más funcionalidades, que van en aumento junto con la tecnología que ofrecen, como pantalla táctil, reconocimiento por voz, captura de video, etc.

Lo anteriormente mencionado nos lleva a concluir que las funcionalidades ofrecidas por los dispositivos móviles se asemejan cada día más a las que ofrece un computador moderno, permitiendo incluso la instalación de programas externos. Es en este punto donde se amplía el público al que apunta el desarrollo de software, esto es debido principalmente a la portabilidad y comodidad que ofrecen estos dispositivos, permitiendo el acceso a la información y entretenimiento en cualquier momento y lugar.

Aprovechando el auge y capacidades de los dispositivos móviles se ha comenzado a desarrollar un amplio mercado de aplicaciones para estos, entre ellos los videojuegos, que demuestran de forma gráfica e interactiva las posibilidades que ofrecen.

El desarrollo de un videojuego implica una combinación armónica entre el arte y la tecnología, siendo necesario para su creación amplios conocimientos computacionales y de diseño gráfico, llegando a considerarse hoy en día por algunos entendidos como un medio para el arte, ya que en ellos se expone música, videos, diseños, tanto de personajes como escenarios.

El proyecto presentado constituye una indagación en nuevas tecnologías, como lo son las plataformas móviles, en este caso, el popular sistema operativo de Google, Android; aprovechando el aumento en las capacidades de estos equipos, especialmente el procesamiento de gráficos en tres dimensiones y las características propias de los equipos modernos, como las pantallas táctiles.

1.2. Descripción de cada capítulo

Primer Capítulo – Marco introductorio

En este capítulo se presenta de manera general el proyecto y la estructura que sigue el informe.

Segundo Capítulo – Antecedentes generales

En este capítulo se detallan los aspectos generales del proyecto: los objetivos, o logros que se desea obtener con el trabajo a realizar, tanto generales como específicos, la justificación del tema escogido ampliamente especificada, la metodología que se utilizó para llevar a cabo el proyecto, los aportes esperados al término del desarrollo del tema y finalmente los límites y alcances del trabajo a realizar.

Tercer Capítulo – Marco teórico

En este capítulo se hace un esbozo de la teoría necesaria para llevar a cabo el proyecto, comenzando con el sistema operativo Android, luego el estándar OpenGL ES y para finalizar una mirada al mundo de los videojuegos actual, los tipos de juegos, y la construcción o desarrollo de videojuegos.

Cuarto Capítulo – Primer Incremento

En este capítulo se detalla lo realizado según la metodología iterativa incremental, partiendo con el primer incremento definido el cual consta de dos etapas: diseño e implementación; dentro de la primera etapa se detalla el guión o historia del juego que incluye la definición de personajes, el diagrama de casos de uso y diagrama de clases según el estándar UML. En la segunda etapa implementación se describe el manejo de cámara, la construcción del escenario, implementación de personajes y objetos.

Quinto Capítulo – Segundo Incremento

Continuando con la metodología Iterativa incremental, en este capítulo se describe lo concerniente al segundo incremento, y al igual que en incremento anterior es dividido en dos etapas. En la primera etapa diseño se complementa lo expuesto en el capítulo anterior: el diagrama de casos de uso, con su respectiva descripción y el diagrama de clases, con su respectiva descripción de clases. En la segunda etapa implementación se

refina lo logrado en la etapa anterior agregando texturas, iluminación gravedad, colisiones y sonido.

Sexto Capítulo – Resultados

En este sexto capítulo se expone mediante capturas de pantalla del trabajo resultante y breves explicaciones lo logrado en este proyecto.

Séptimo Capítulo – Conclusiones Generales

En este último capítulo se presenta el análisis de los resultados obtenidos con respecto a los objetivos plateados y el trabajo realizado. También se detalla el posible trabajo a futuro derivado de este proyecto.

SEGUNDO CAPÍTULO - ANTECEDENTES GENERALES

2.1. Objetivos

2.2.1. Objetivo general

- Diseñar e implementar un juego estilo plataforma en un entorno 3D para dispositivos móviles con sistema operativo Android.

2.2.2. Objetivos específicos

- Comprender y utilizar el estándar OpenGL ES en Android.
- Dotar de propiedades físicas a los objetos y personajes del juego, como gravedad y aceleración.
- Implementar la detección de colisiones simple y manejo de coordenadas.
- Incorporar modelos, texturas, color, iluminación, sonido y animación en el juego.
- Lograr control de la cámara para seguir al personaje (vista en tercera persona).
- Utilizar las características del dispositivo móvil para diseñar la interacción física con el usuario.

2.2. Justificación

Hoy en día es bien sabido el auge de los llamados Smartphone, los cuales ya pasaron de ser sólo utilizados para realizar llamadas e incluyen características propias de los computadores modernos. Una característica esencial de estos dispositivos es que permiten la instalación de programas lo que aumenta su potencial; entre estos se encuentra los videojuegos, los cuales son los más descargados en las tiendas de aplicaciones, abriendo un gran mercado para las empresas desarrolladoras de videojuegos, las cuales siempre buscan ampliar su público objetivo.

Siguiendo esta línea, el proyecto propuesto representa una indagación en nuevas tecnologías, como lo son las plataformas móviles, en este caso, Android; aprovechando el aumento en las capacidades de estos equipos, especialmente el procesamiento de gráficos y las características propias de los equipos modernos, como las pantallas táctiles.

Al existir escasos frameworks de alto nivel que ayuden en la labor de desarrollo y diseño de juegos para Android, constituye un desafío lograr crear algo de tal complejidad, ya que un juego se basa en el diseño, multimedia y sobre todo programación.

2.3. Metodología

La metodología a utilizar será Iterativa Incremental, el enfoque orientado a objetos y el lenguaje de modelado UML.

Se han considerado dos incrementos:

Primer Incremento

- Diseño
- Guión del juego
- Diagrama de casos de uso
- Diagrama de clases
- Implementación
- Cámara
- Escenario
- Personajes
- Objetos

Segundo Incremento

- Diseño
- Detalle casos de uso
- Descripción de clases
- Implementación
- Texturas
- Iluminación
- Gravedad
- Colisiones
- Sonido

2.4. Aportes esperados

Servir de ejemplo para el desarrollo de futuras aplicaciones para dispositivos móviles, tanto juegos como aplicaciones tradicionales en los siguientes aspectos:

- Exponer la factibilidad del desarrollo de aplicaciones 3D en smartphones con sistema operativo Android.
- Mostrar los alcances en cuanto a gráficas de los dispositivos móviles actuales
- Aprovechar las capacidades en la interacción con el usuario de los móviles
- Comparar el desarrollo de juegos para plataformas tradiciones (PC) y dispositivos móviles.

2.5. Límites y alcances del trabajo

El juego a desarrollar será estilo plataforma, los cuales se caracterizan por hacer que el protagonista camine, salte o escale sobre una serie de obstáculos, con enemigos, a la vez que se alcanzan objetivos para poder completar el juego. Además será de un jugador y con una cierta cantidad de niveles en un ambiente 3D. Podemos destacar que el objetivo principal será la interacción entre el jugador y el entorno.

El proyecto no contempla que el juego tenga conexión a internet, base de datos, ni interacción con otro dispositivo.

TERCER CAPÍTULO - MARCO TEÓRICO

3.1. Sistema Operativo Android

Un sistema operativo es un software que actúa de interfaz entre los dispositivos de hardware y los programas de usuario o el usuario mismo para utilizar un computador (ver Figura 1). Es responsable de gestionar, coordinar las actividades y llevar a cabo el intercambio de los recursos y actúa como intermediario para las aplicaciones que se ejecutan. Ejemplos de sistemas operativos son Windows, Macintosh y Linux. Para dispositivos móviles también existen sistemas operativos, como Android, Symbian y Windows Mobile.

Figura 1: Interacción entre el SO y el resto de los componentes.

Android es un sistema operativo orientado a dispositivos móviles que usa una versión modificada de la versión 2.6 del Kernel Linux (en la Figura 2 se puede apreciar el logo característico de Android). Es desarrollado por la Open Handset Alliance, que aglutina a fabricantes de software y hardware, entre los que destacan Google, T Mobile, HTC, Qualcomm y Motorola entre otros.

Figura 2: Logo del Sistema Operativo Android

3.1.1. Historia

Este sistema operativo fue inicialmente desarrollado por Android Inc. Firma que fue adquirida por Google el 2005 y se ha posicionado fuertemente en el mercado llegando al primer lugar en los rankings de ventas de smartphones en Estados Unidos, superando al IPhone de Apple y Blackberry de RIM.

Android posee una gran comunidad de desarrolladores, actualmente existen alrededor de 70.000 aplicaciones disponibles para Android, lo que la convierte en el segundo sistema operativo para móviles con más aplicaciones disponibles.

Cuatro son las principales versiones de Android hoy en día: 1.5, de nombre en clave Cupcake, 1.6 o Donut, 2.1 Eclair y la actual 2.2 Foyo. Por supuesto, también existieron versiones anteriores: la 1.0, con la que se lanzó el primer móvil Android del mercado, el HTC Dream o G1, de octubre de 2008; y la 1.1, de febrero de 2009, que solucionaba varios errores y añadía alguna que otra funcionalidad no demasiado importante.

3.1.2. Características

La plataforma Android posee las siguientes características:

- Framework de aplicaciones: permite reutilización y reemplazo de componentes.
- Máquina virtual Dalvik: optimizada para dispositivos móviles.
- Navegador integrado: basado en el motor de código abierto WebKit.
- Gráficos optimizados, con una biblioteca de gráficos 2D; gráficos 3D basado en la especificación OpenGL ES 1.0 (aceleración por hardware opcional).
- SQLite para almacenamiento de datos estructurados.
- Soporte para medios con formatos comunes de audio, vídeo e imágenes planas (MPEG4, H.264, MP3, OGG, AAC, AMR, JPG, PNG, GIF)
- Telefonía GSM (dependiente del hardware)
- Bluetooth, EDGE, 3G, y WiFi (dependiente del hardware)
- Cámara, GPS, brújula, y acelerómetro (dependiente del hardware)
- Ambiente rico de desarrollo incluyendo un emulador de dispositivo, herramientas para depurar, perfiles de memoria y rendimiento, y un complemento para el IDE Eclipse.

3.1.3. Arquitectura

Los componentes principales del sistema operativo de Android son (ver Figura 3):

- Aplicaciones: las aplicaciones base incluirán un cliente de email, programa de SMS, calendario, mapas, navegador, contactos, y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.
- Framework de aplicaciones: los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.
- Bibliotecas: Android incluye un set de bibliotecas C/C++ usadas por varios componentes del sistema Android. Estas características se exponen a los desarrolladores a través del framework de aplicaciones de Android; algunas son: System C library (implementación biblioteca C standard), bibliotecas de medios, bibliotecas de gráficos, 3d, SQLite, entre otras.
- Runtime de Android: Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para memoria mínima. La Máquina Virtual está basada en registros, y corre clases compiladas por el compilador de Java que han sido transformadas al formato.dex por la herramienta incluida "dx".
- Núcleo Linux: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, stack de red, y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto del stack de software.

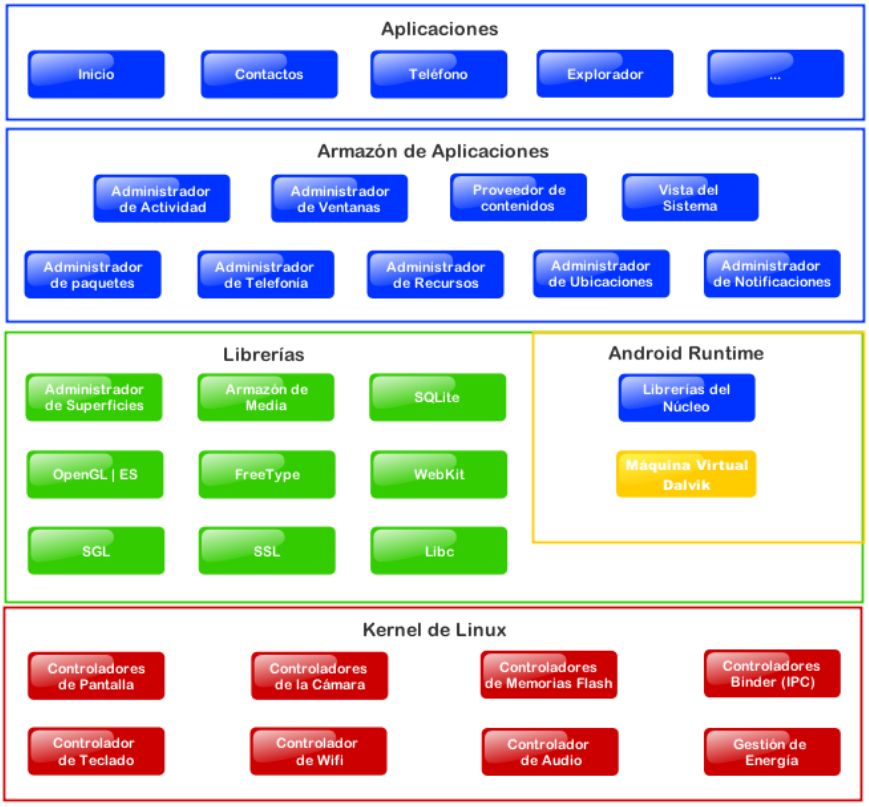

Figura 3: Diagrama arquitectura Android

3.2. OpenGL ES

OpenGL ES (OpenGL for Embedded Systems) es una API (Application Programming Interface) gráfica OpenGL diseñada para dispositivos integrados tales como teléfonos móviles, PDAs y consolas de videojuegos (en la Figura 4 se puede apreciar el logo de la API OpenGL ES). La define y promueve el Grupo Khronos, un consorcio de empresas dedicadas a hardware y software gráfico interesadas en APIs gráficas y multimedia.

Figura 4: Logo de OpenGL ES

Actualmente existen dos APIs consideradas como estándares: DirectX y OpenGL. DirectX es el estándar de facto 3D para cualquier sistema con sistema operativo Microsoft Windows y es usado para la mayoría de los juegos 3D en esta plataforma. OpenGL es una API estándar multiplataforma para sistemas basados en Linux, Mac OS X y también Microsoft Windows. Debido a lo anteriormente mencionado OpenGL ES se basa en OpenGL y no en DirectX.

OpenGL ES se ha transformado en un estándar a la hora de programar para dispositivos integrados, facilitando y haciendo accesible una variedad de funciones para trabajar con gráficos 3D avanzados y desarrollar juegos para la mayoría de los dispositivos móviles y sistemas integrados. Debido a que OpenGL ES está basado en OpenGL, no es necesario desarrollar nueva tecnología, esto permite ir adaptando el estándar OpenGL, la API gráfica multiplataforma más ampliamente utilizada, a las necesidades del OpenGL ES.

La razón de la adaptación de las funciones y métodos de OpenGL, es debido a que los dispositivos integrados poseen limitada capacidad de procesamiento y disponibilidad de memoria que los computadores de escritorio, es por esto que a la hora de definir OpenGL ES se tomaron en cuenta los siguientes criterios:

- La API de OpenGL es grande y compleja y la meta es crear una API para dispositivos limitados. Para cumplir esto, se removió la redundancia; por ejemplo la especificación de la geometría, donde en OpenGL una aplicación puede utilizar, nodos, listas o arreglos de vértices, en cambio en la nueva API solo se puede utilizar arreglos de vértices.
- Compatibilidad con OpenGL. OpenGL ES contiene una parte de las funcionalidades de OpenGL, pero optimizadas para los dispositivos a los cuales está orientada.
- Nuevas características orientadas a los dispositivos integrados fueron incluidas; por ejemplo, la reducción del consumo de la batería.
- Asegurar que OpenGL ES cubriera el mínimo de características usadas para dar calidad a las imágenes. Esto es debido, a que la mayoría de los dispositivos poseen una pantalla de tamaño limitado, haciendo esencial la calidad con que los pixeles son dibujados en la pantalla.
- Garantizar el cumplimiento de estándares de calidad de imagen, segura y robusta.

A la fecha existen tres especificaciones de OpenGL ES, lanzadas por el grupo Khronos: OpenGL ES 1.0, OpenGL ES 1.1 y OpenGL ES 2.0

3.2.1. OpenGL ES y Android

Android incluye soporte para gráficos de gran calidad mediante OpenGL ES, específicamente la versión 1.0 que corresponde a la versión 1.3 de OpenGL.

Para utilizar esta API se debe:

- Crear una subclase de Vista.
- Obtenerla y manejarla mediante OpenGLContext, el cual provee el acceso a las funcionalidades de OpenGL.
- En la clase de Vista contar con el método onDraw(), el cual obtiene un objeto GL y utiliza este métodos para realizar operaciones sobre él.

3.3. Videojuegos

Un videojuego (del inglés videogame) o juego de video es un software creado para el entretenimiento en general y basado en la interacción entre una o varias personas y un aparato electrónico que ejecuta dicho videojuego; este dispositivo electrónico puede ser una computadora, un sistema arcade, una videoconsola, un dispositivo handheld o un teléfono móvil, los cuales son conocidos como "plataformas".

La historia de los videojuegos comenzó en 1947, cuando la idea de un videojuego fue concebida y patentada por Thomas T. Goldsmith Jr y Estle Ray Mann, los cuales llenaron una aplicación de patente en Estados Unidos el 25 de enero de 1947. El éxito de los videojuegos se ha extendido hasta hoy día y posee un futuro prometedor para beneficio de la Industria de los videojuegos y de los usuarios.

La evolución de los videojuegos va estrechamente ligada a la evolución en la tecnología disponible en las videoconsolas, así para estas últimas existe una clasificación:

- Primera generación (1972 – 1977) de 2 bits: Atari Pong
- Segunda generación (1976 – 1984) 4 y 8 bits: Game & Watch, Atari 2600
- Tercera generación (1983 – 1992) 8 bits: Nintendo Entertainment System, GameBoy, Sega Master System, Sega Game Gear
- Cuarta generación (1988 – 1996) 16 bits: Sega Mega Drive, Neo-Geo, Super Nintendo Entertainment System
- Quinta generación (1993 - 2002) 32 y 64 bits: Atari Jaguar, Sega Saturn, PlayStation, Nintendo 64.

- Sexta generación (2002 – 2006) 128 y 32 bits portátiles: Sega Dreamcast, PlayStation 2, Xbox, Nintendo GameCube, GameBoy Advance
- Séptima generación (2005 - actualidad) Nextgen: Wii, Xbox 360, PlayStation 3, Nintendo DS, Nintendo DSi, PlayStation Portable

3.3.1. Tipos de juegos

Existe una gran variedad de tipos o clases de videojuegos. Se pueden clasificar según la cantidad de personas que interactúan: un jugador o multijugador. Los juegos multijugador se pueden clasificar en:

- Por turnos.
- Simultáneo: dos o más jugadores al mismo tiempo.
- Pantalla dividida: se divide la pantalla según la cantidad de jugadores.
- Red local: dos o más dispositivos conectados en forma local.
- En línea: dos o más dispositivos conectados mediante internet.
- PBEM (Play by mail): similar a jugar por turnos, pero se sincronizan mediante correos electrónicos.

Sobre los géneros de videojuegos, son muy variados; van desde aventura a horror, y generalmente un videojuego mezcla estos géneros, por lo que la mayoría de estos no caben dentro de una sola categoría.

Los principales tipos o géneros de los videojuegos son:

- Aventura: Son videojuegos en los que el protagonista debe avanzar en la trama interactuando con diversos personajes y objetos.
- Disparos: En estos videojuegos el protagonista ha de abrirse camino a base de disparos.
- Educativos: Videojuegos cuyo objetivo es dar a conocer al usuario algún tipo de conocimiento.
- Estrategia: Se caracterizan por la necesidad de manipular a un numeroso grupo de personajes, objetos o datos para lograr los objetivos.
- Lucha: Son videojuegos basados en el combate cuerpo a cuerpo.
- Terror: El/los protagonistas viven aventuras dónde deben salir airosos de situaciones típicas de una película de terror (escapar de una casa llena de zombis, huir de un asesino, resolver misterios para aplacar a los fantasmas, etc.).

- Plataformas: Los videojuegos de plataformas se caracterizan por tener que recorrer, saltar o escalar una serie de plataformas y acantilados, con enemigos, mientras se recogen objetos para poder completar el videojuego.
- Rol: Son videojuegos donde el protagonista interpreta un papel y ha de mejorar sus habilidades mientras interactúa con el entorno y otros personajes.
- Musicales: Su desarrollo gira en torno a la música.
- Party games: En este género los jugadores habrán de ir avanzando por turnos en un tablero virtual e ir superando diversas pruebas de tipos muy diversos en los que compiten entre sí por llegar lo antes posible a la meta, o conseguir la máxima cantidad posible de puntos.
- Simulación: Involucran al jugador en una situación simulada determinada, ya sea de gestor de un zoológico, una ciudad o una vida propia virtual.
- Deportivos: Son videojuegos basados en el mundo del deporte.
- Carreras: Son videojuegos en los que se pilotan diferentes vehículos, ya sean reales o ficticios, para ganar en diferentes carreras.
- Sandbox: Se caracterizan por ser videojuegos en los que el jugador puede elegir el orden de las misiones o viajar libremente por el mapa del videojuego, e interactuar con casi todo lo que este a su disposición.

3.3.2. Desarrollo de videojuegos

El desarrollo de videojuegos es la actividad en la cual se diseña y crea un videojuego, desde el concepto inicial hasta el videojuego en su versión final. Ésta es una actividad multidisciplinaria, que involucra profesionales de la informática, el diseño, el sonido, la actuación, etc.

El desarrollo de un videojuego generalmente sigue el siguiente proceso:

- Concepción de la idea del videojuego
- Diseño
- Planificación
- Producción
- Pruebas
- Mantenimiento

El proceso es similar a la creación de software en general, aunque difiere en la gran cantidad de aportes creativos (música, historia, diseño de personajes, niveles, etc.) necesarios. El desarrollo también varía en función de la plataforma objetivo (PC, móviles,

consolas), el género (estrategia en tiempo real, rpg, aventura gráfica, plataformas, etc.) y la forma de visualización (2D, 2.5D y 3D).

3.3.3. Desarrolladores de videojuegos

Los desarrolladores de videojuegos (Video Game Developer) son empresas o individuos dedicados a la creación de videojuegos. Estos desarrolladores pueden especializarse según el tipo de consola a la cual está destinado el juego a crear, por ejemplo PlayStation 3 de Sony, Xbox 360 de Microsoft, Nintendo DS o Wii de Nintendo, o también desarrollar juegos para una gran variedad de sistemas, incluyendo los computadores o dispositivos móviles.

Muchos desarrolladores también pueden especializarse según el tipo de juego, por ejemplo juegos de rol o shooters; otros se enfocan a portar juegos de un sistema a otro, por ejemplo Xbox 360 a PlayStation 3; como también existen desarrolladores dedicados a traducir juegos.

Muchos de los llamados publicadores de videojuegos (videogame publishers) mantienen estudios de desarrollo de videojuegos, como los conocidos Electronic Arts con EA Canada, Activision con Radical Entretainment, Nintendo con EAD y Sony con Polyphony Digital.

Entre los puestos o actividades que desempeña un desarrollador de videojuegos se encuentran:

- Game art design
- Game design
- Game development /Game developer
- Game director
- Game modification
- Game producer
- Game programming /Game programmer
- Game publisher
- Game studies
- Game testing
- Game journalism
- Level design

3.3.4. Herramientas para el desarrollo de video juegos

Dentro de las herramientas para el desarrollo de video juegos tenemos:

- Las encargadas un aspecto del juego, por ejemplo el modelado y diseño 3D
- Los llamados Frameworks, que engloban el desarrollo de un video juego, combinando la programación con los modelos, sonido, videos, etc. los cuales se centran en el desarrollo para una o varias plataformas.
- Las librerías que facilitan la programación de estas aplicaciones.

En la primera categoría, herramientas para el desarrollo de un aspecto del video juego, tenemos gran variedad, sobretodo en el ámbito de diseño y animación de personajes: 3DS Max, Maya, Google SketchUp, Misfit3D, donde las dos primeras herramientas son herramientas a nivel profesional, lo que involucra una alta complejidad en su manejo. Debido a lo anterior se opto por utilizar en el desarrollo del proyecto Google SketchUp y Misfit3D, los cuales son más intuitivos y requieren menor tiempo en aprender a utilizarlos.

En cuanto a frameworks, para el desarrollo de video juegos, tenemos el popular Unity3D, el cual tiene soporte para diversas plataformas: Windows, Mac OS X , Wii, iPhone/iPad, Xbox 360 , PS3 y recientemente Android.

Las librerías engloban ciertos procedimientos para facilitar la labor de desarrollo de aplicaciones. Es así, que por ejemplo, la librería utilizada en el desarrollo del proyecto jPCT-AE, utiliza OpenGL ES.

3.4. Smartphones

Un Smartphone es un teléfono móvil que ofrece habilidad de cómputo y conectividad más avanzada que un teléfono básico. Los Smartphones son pensados como computadores integrados a un dispositivo móvil, ya que permiten al usuario instalar y correr aplicaciones más avanzadas que las que se ven en un teléfono común y además utilizan un completo sistema operativo, proveyendo una plataforma para el desarrollo de aplicaciones.

Para el desarrollo de este proyecto se utilizaron dos Smartphone: Samsung S Galaxy y Samsung i5500 Galaxy 5. Si bien el Samsung S Galaxy es notoriamente superior al Samsung i5500 Galaxy 5, se utilizó esencialmente este último puesto que poseíamos uno, no así el Samsung S Galaxy.

3.4.1. Samsung S Galaxy

Este modelo es uno de los dispositivos móviles con sistema operativo Android más modernos y destacados (ver Figura 5). Destacan el tamaño y calidad de su pantalla y su potencia.

Figura 5: Imágenes Samsumg S Galaxy

Tabla 1: Características Samsung S Galaxy

General	Red	GSM 850 / 900 / 1800 / 1900 - HSDPA 900 / 1900 / 2100 o HSDPA 850 / 1900 / 2100 o HSDPA 900 / 1700 / 2100
	Anunciado	2010, Marzo
	Estado	Disponible
Tamaño	Dimensiones	122.4 x 64.2 x 9.9 mm
	Peso	118 g
Pantalla	Tipo	Super AMOLED touchscreen capacitivo, 16M colores
	Tamaño	480 x 800 pixels, 4.0 pulgadas
		Interfaz de usuario TouchWiz 3.0
		Sensor de proximidad para auto apagado
		Ingreso de datos Swype
		Sensor acelerómetro para auto rotación
		Ingreso de datos multi-touch
Sonido	Tipos de alertas	Polifónico, MP3, WAV
		Conector de audio 3.5 mm
Memoria	Agenda	Entradas y campos prácticamente ilimitados
	Registro de llamadas	Prácticamente ilimitados
	Interna	8 GB/16GB memoria interna, 512 MB RAM
	Card slot	microSD hasta 32GB
Datos	GPRS	Clase 12 (4+1/3+2/2+3/1+4 slots)
	EDGE	EDGE Clase 12
	3G	3G HSDPA / HSUPA
	WLAN	Wi-Fi, DLNA
	Bluetooth	Bluetooth v3.0
	Puerto infrarrojo	No
	USB	microUSB 2.0
Cámara	Primaria	5 MP, 2592 x 1944 pixels, autofocus,flash LED, foco táctil, geo-tagging, detección de rostro y sonrisa, cámara secundaria videollamadas VGA
	Video	Sí, video 720p@30fps
Características	OS	Android OS v2.1 (Eclair)
	CPU	ARM Cortex A8 1GHz
	Mensajes	SMS, MMS, Email, Push Mail, IM, RSS
	Browser	HTML
	Juegos	Sí
	Colores	Negro y Gris
	GPS	Sí
		Brújula digital
		Integración con redes sociales
		Reproductor de video MP4/DivX/WMV/H.264/H.263
		Reproductor de audio MP3/WAV/eAAC+/AC3/FLAC
		Radio FM Stereo con RDS
		Visor de documentos (Word, Excel, PowerPoint,

		PDF)
		Integración Google Search, Mapas, Gmail, YouTube, Calendario, Google Talk, Picasa
		Organizador
		Editor de fotos y videos
		Flash Lite v3.1
		Manoslibres incorporado
		Memo/comandos/discado de voz
		T9
Batería		Standard, Li-Ion 1500 mAh
	Stand-by	Hasta 750 h (2G) / Hasta 576 h (3G)
	Talk time	Hasta 13 h 30 min (2G) / Hasta 6 h 30 min (3G)

3.4.2. Samsung i5500 Galaxy 5

Este dispositivo es otro de los utilizados en el desarrollo del proyecto. En resumen posee una pantalla QVGA touchscreen de 2.8 pulgadas, 3G, cámara de 2 megapixels, Wi-Fi, Bluetooth 2.1, A-GPS, ranura microSD para memorias, widgets para conectarse con redes sociales y servicios Google.

Tabla 2: Características Samsung i5500

General	Red	GSM 850 / 900 / 1800 / 1900 - HSDPA 900 / 2100
	Anunciado	2010, Junio
	Estado	Disponible desde lanzamiento
Tamaño	Dimensiones	108 x 56 x 12.3 mm
	Peso	102 g
Pantalla	Tipo	TFT touchscreen capacitivo, 256K colores
	Tamaño	240 x 320 pixels, 2.8 pulgadas
		Sensor acelerómetro para auto rotación
		Touch Wiz v3.0
Sonido	Tipos de alertas	Tipo Polifónico, MP3, WAV
	Customización	Vibración Descargas Conector de audio 3.5 mm
Memoria	Agenda	Entradas y campos prácticamente ilimitados, Foto de llamada
	Registro de llamadas	Prácticamente ilimitados
	Interna	170 MB memoria interna
	Card slot	microSD (TransFlash) hasta 16GB, 1GB incluidos
Datos	GPRS	Clase 10 (4+1/3+2 slots)
	EDGE	Clase 10, 236.8 kbps
	3G	HSDPA, 7.2 Mbps; HSUPA, 2 Mbps
	WLAN	Wi-Fi 802.11 b/g
	Bluetooth	Bluetooth v2.1 A2DP
	Puerto infrarrojo	No

	USB	Sí, miniUSB
Cámara	Primaria	2 MP, 1600x1200 pixels, geo-tagging, video
	Video	Sí, QVGA@15fps
Características	OS	Android OS, v2.1 (Eclair)
	CPU	600MHz
	Mensajes	SMS, MMS, Email, Push Mail, IM, RSS
	Browser	HTML
	Radio	Si
	Juegos	Sí
	Colores	Negro
	GPS	Sí, con soporte A-GPS
	Java	Mediante terceras aplicaciones
		Wi-Fi
		Integración con redes sociales
		Reproductor de video MP4/H.264/H.263
		Reproductor de audio MP3/WAV/eAAC+
		Integración Google Search, Maps, Gmail,
		YouTube
		Manoslibres incorporado
		Memo de voz
		Radio FM Stereo con RD
		Organizador
		Visor de documentos (Word, Excel, PowerPoint, PDF)
Batería		Standard, Li-Ion 1200 mAh
	Stand-by	Hasta 521 h (2G) / Hasta 375 h (3G)
	Talk time	Hasta 9 h 30 min (2G) / Hasta 6 h 30 min (3G)

3.5. Hardware disponible en los dispositivos móviles

Está sección tiene como objetivo describir de manera general el hardware con el que cuentan los dispositivos con sistema operativo Android, en especial aquel hardware relacionado con los videojuegos, es decir, aceleración gráfica, sonido, etc.

3.5.1. Pantallas

Las pantallas son el principal periférico de salida en los dispositivos móviles, ya que permiten la interacción principal entre el usuario y el dispositivo. Debido a lo anteriormente mencionado es que la tecnología aplicada a esos dispositivos ha ido explosivamente en aumento, desde las clásicas pantallas monocromáticas a las TFT[1] con millones de colores utilizadas en los dispositivos actuales y los próximamente dispositivos móviles con pantallas 3D (ver Figura 6) . Además con la incorporación de pantallas touch, estas también se han convertido en el principal periférico de entrada.

Figura 6: Adelanto de pantalla 3D de la empresa Sharp

Para comprender de mejor maneta estos dispositivos se debe conocer algunos conceptos:

- **LCD**: (Liquid Crystal Display) Pantalla de Cristal Liquido.

[1] TFT, siglas de Thin Film Transistor (en inglés: Transistor de Película Fina), es un tipo especial de transistor de efecto campo que se fabrica depositando finas películas de un semiconductor activo así como una capa de material dieléctrico y contactos metálicos sobre un sustrato de soporte. Un sustrato muy común es el cristal. Una de las primeras aplicaciones de los TFTs son las pantallas de cristal líquido.

Todas las pantallas que se utilizan en dispositivos móviles, a excepción de las OLED, son LCD planas. Las LCD convencionales funcionan de la siguiente manera: constan de dos filtros polarizados, situados en un ángulo de 90 grados, por lo que la luz no pasa; entre ambos filtros se sitúa un cristal que tiene la propiedad de girar 90 grados el plano de la luz polarizada, por lo que el conjunto parece transparente. Al aplicar electricidad al cristal, cambia y deja de girar la luz, con lo que se oscurece.

Hay que aclarar que todas las LCD deben iluminarse por detrás, ya sea de forma activa mediante una luz (fluorescente, LEDs), lo que es imprescindible en las de color, o mediante un panel reflector, como llevan los relojes y calculadoras.

- **Tamaño**

  El tamaño de las pantallas se mide en pulgadas, al igual que los televisores. Hay que tener en cuenta que lo que se mide es la longitud de la diagonal. El tamaño es importante porque nos permite tener varias tareas a la vez de forma visible, y poder trabajar con ellas de manera cómoda.

- **Pixel**

  De Picture Element, es decir, elemento de la imagen. Es la menor unidad en la que se descompone una imagen digital, ya sea en una fotografía, un grafico o un fotograma de video. Un pixel representa puntos individuales de luz usados para componer una imagen en la pantalla.

  Un pixel está compuesto de 3 sub-pixeles en los colores primarios, rojo, verde y azul. En cada una de las posiciones de los píxeles en un monitor LCD, 3 células de material de cristal líquido forman los sub píxeles rojos, verdes y azules que conjuntamente permiten el rango total de colores que se pueden mostrar. Los transistores individuales están acomodados ordenadamente en la parte de atrás del vidrio para controlar cada sub píxel.

- **Resolución**

  Resolución de pantalla se denomina a la cantidad de pixels que se pueden ubicar en un determinado modo de pantalla. Estos pixels están a su vez distribuidos entre el total de horizontales y el de verticales.

- **Color**

  Mientras que las pantallas a color se están convirtiendo en un estándar para los teléfonos móviles, los de gama baja todavía presentan una pantalla "Monocromática", término mal empleado puesto que el significado de Monocromático es un color de primer plano y un color de fondo, como el negro sobre blanco. Los dispositivos móviles usan pantallas con varios tonos de grises del negro al blanco (para ser exactos 16). La ventaja de las pantallas monocromáticas o a escala de grises es que estas necesitan menos energía para

funcionar que las pantallas a color. La desventaja es que estas pantallas son más incomodas de leer, y por supuesto no son tan vistosas como las pantallas a color.

Las primeras pantallas a color que aparecieron fueron las de 256 colores u 8 bits, en el 2003 se dio un gran salto pasando de las pantallas de 8 bits a 12 bits y posteriormente a 16 bits, que es el estándar actual; hoy en día estamos en la transición de pantallas de 16 a 18 bits (65 536 a 262 144 colores) y pronto se dará el gran salto de 18 a 24 bits (262 144 a 16' 777 216 colores). Cabe resaltar que casi no existe diferencia alguna entre una pantalla de 16 bits y una de 18 bits, el "usuario común" no puede distinguir entre las dos y solo la puede distinguir como si fuera más brillante.

- **Pantalla Táctil**

  Una pantalla táctil (touchscreen en inglés) es una pantalla que mediante un toque directo sobre su superficie permite la entrada de datos y órdenes al dispositivo (ver Figura 7). A su vez, actúa como periférico de salida, mostrándonos los resultados introducidos previamente. Este contacto también se puede realizar con lápiz u otras herramientas similares. Actualmente hay pantallas táctiles que pueden instalarse sobre una pantalla normal. Así pues, la pantalla táctil puede actuar como periférico de entrada y periférico de salida de datos, así como emulador de datos interinos erróneos al no tocarse efectivamente.

- **Pantallas Multitáctiles**

  Multitáctil es el nombre con el que se conoce a una técnica de interacción persona-computador y al hardware que la implementa. La tecnología multitáctil consiste en una pantalla táctil o touchpad que reconoce simultáneamente múltiples puntos de contacto, así como el software asociado a esta que permite interpretar dichas interacciones simultáneas (ver Figura 8).

Figura 7: Pantalla táctil de un PDA

Figura 8: Ejemplo pantalla multitáctil

3.5.2. Sensores

- **Acelerómetro**

  El Acelerómetro de los móviles es un dispositivo insertado dentro del móvil que mide la aceleración y las fuerzas inducidas por la gravedad, lo que nos permite detectar el movimiento y el giro del teléfono. Esto permite por ejemplo, girar una foto con solo dar vuelta el aparato.

- **Compas electrónico**

  El compás electrónico sirve para determinar la dirección a la que un objeto está apuntando. Es utilizado para determinar la orientación del dispositivo móvil (ver Figura 9).

  Al contrario de los compases magnéticos, los electrónicos no tiene elementos mecánicos que puedan influir en las mediciones, desapareciendo los problemas de inercia y ofreciendo una mayor rapidez de respuesta.

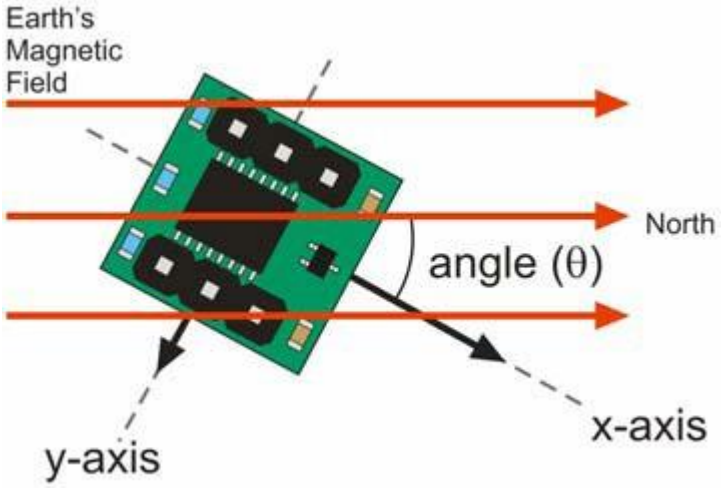

Figura 9: Compás electrónico por efecto Hall[2] HM55B

- **Sensor de proximidad**

  El sensor de proximidad es un transductor que detecta objetos o señales que se encuentran cerca del elemento sensor.

[2] El efecto Hall consiste en la aparición de un campo eléctrico en un conductor cuando es atravesado por un campo magnético. A este campo eléctrico se le llama campo Hall. Llamado efecto Hall en honor a su descubridor Edwin Herbert Hall.

Existen varios tipos de sensores de proximidad según el principio físico que utilizan. Los más comunes son los interruptores de posición, los detectores capacitivos, los inductivos y los fotoeléctricos, como el de infrarrojos.

Estos sensores se encargan por ejemplo, de detectar la proximidad de la cara del usuario cuando este habla por teléfono para desconectar la pantalla del mismo con el propósito de ahorrar batería.

- **GPS**

  El GPS (Global Positioning System: sistema de posicionamiento global) o NAVSTAR-GPS1 es un sistema global de navegación por satélite (GNSS) que permite determinar en todo el mundo la posición de un objeto, una persona, un vehículo o una nave, con una precisión hasta de centímetros (si se utiliza GPS diferencial), aunque lo habitual son unos pocos metros de precisión.

  Algunos dispositivos móviles pueden vincularse a un receptor GPS diseñado a tal efecto. Suelen ser módulos independientes del teléfono que se comunican vía inalámbrica bluetooth, o implementados en el mismo terminal móvil, y que le proporcionan los datos de posicionamiento que son interpretados por un programa de navegación.

3.5.3. Sonido

El sonido en los dispositivos móviles es la capacidad de reproducir melodías polifónicas, MP3 u otros formatos de reproducción de audio.

Los formatos de audio soportados por los móviles son:

- **WAV**: abarca una gran cantidad de posibilidades en cuanto a su calidad (desde el sonido telefónico hasta el sonido estéreo de alta fidelidad). A pesar de ser el menos sofisticado resulta muy útil cuando su contenido ha de ser manipulado por elementos (DSP).
- **MP3**: un fichero en MP3 es una secuencia de tramas MPEG I layer III, descritas según la norma IO3-11172[3]. Es muy flexible en cuanto a modos de almacenamiento y en cuanto al ahorro de espacio en disco. Como desventaja

[3] "Coding of moving Pictures and associated audio for storage media up to about 1.5 Mbits/s" (codificación de imágenes en movimiento y audio asociado para un almacenamiento medio hasta 1.5 Mbits/s).

presenta que al usar un algoritmo de codificación/decodificación más complejo, la grabación/reproducción es más lenta y consume algunos recursos informáticos.

- **MIDI** (Musical Instrument Digital Interface): interfaz digital para instrumentos musicales. Archivo liviano que sólo contiene información sobre lo que tocan los instrumentos musicales, no almacena sonido.
- **AMR**: formato de propiedad de Apple, la empresa que distribuye el conocido reproductor multimedia QuickTime.
- **MMF** (Multimedia Mobile File): formato de propiedad de Yamaha. Archivo liviano utilizado para los timbres de teléfono.

3.5.4. Memoria

Los dispositivos móviles actuales cuentan con dos clases de memoria disponible: Interna y Externa.

- **Interna**: Esta memoria es propia del dispositivo y generalmente muy limitada en comparación a la memoria externa, pero su principal ventaja es la rapidez en el acceso a lectura.
- **Externa**: Esta memoria es externa al dispositivo, generalmente mediante microSD[4], lo que amplía enormemente la capacidad del dispositivo hasta hoy en día 64GB.

3.5.5. Datos

- **3G**

  Es la tercera-generación de transmisión de voz y datos a través de la telefonía móvil. Se le llama 3G por 3rd Generation. Los servicios asociados con 3G proporcionan la posibilidad de transferir tanto voz y datos (una llamada telefónica o una videollamada) y datos no-voz (como la descarga de programas, intercambio de email, y mensajería instantánea).

  Aunque esta tecnología estaba orientada a la telefonía móvil, desde hace unos años las operadoras de telefonía móvil ofrecen servicios exclusivos de

[4] Las tarjetas microSD o Transflash corresponden a un formato de tarjeta de memoria flash más pequeña que la MiniSD, desarrollada por SanDisk; adoptada por la Asociación de Tarjetas SD (SD Card Association) bajo el nombre de «microSD» en julio de 2005. Mide tan solo 15 × 11 × 1 milímetros, lo cual le da un área de 165 mm²

conexión a Internet mediante módem usb, sin necesidad de adquirir un teléfono móvil, por lo que cualquier computador puede disponer de acceso a Internet.

- **4G**

  Son las siglas de la cuarta generación de tecnologías de telefonía móvil. Es llamada 4G por 4th Generation.

  La 4G está basada totalmente en IP siendo un sistema de sistemas y una red de redes, alcanzándose después de la convergencia entre las redes de cables e inalámbricas así como en ordenadores, dispositivos eléctricos y en tecnologías de la información así como con otras convergencias para proveer velocidades de acceso entre 100 Mbps en movimiento y 1 Gbps en reposo, manteniendo una calidad de servicio (QoS) de punta a punta (end-to-end) de alta seguridad para permitir ofrecer servicios de cualquier clase en cualquier momento, en cualquier lugar, con el mínimo costo posible.

- **Bluetooth**

  Es una especificación industrial para Redes Inalámbricas de Área Personal (WPANs) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son:

  - Facilitar las comunicaciones entre equipos móviles y fijos.
  - Eliminar cables y conectores entre éstos.
  - Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

  Los dispositivos que con mayor frecuencia utilizan esta tecnología pertenecen a sectores de las telecomunicaciones y la informática personal, como PDA, teléfonos móviles, computadoras portátiles, ordenadores personales, impresoras o cámaras digitales.

- **USB**

  El Universal Serial Bus (bus universal en serie) o Conductor Universal en Serie (CUS), abreviado comúnmente USB, es un puerto que sirve para conectar periféricos a un ordenador. Fue creado en 1996 por siete empresas (que actualmente forman el consejo directivo): IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC. 1

- **Wi-Fi**

  Wi-Fi es una marca de la Wi-Fi Alliance (anteriormente la WECA: Wireless Ethernet Compatibility Alliance), la organización comercial que adopta, prueba y certifica que los equipos cumplen los estándares 802.11 relacionados a redes inalámbricas de área local.

Existen diversos tipos de Wi-Fi, basado cada uno de ellos en un estándar IEEE 802.11 aprobado. Los estándares son los siguientes: IEEE 802.11b, IEEE 802.11g e IEEE 802.11n. Estos estándares disfrutan de una aceptación internacional debido a que la banda de 2.4 GHz está disponible casi universalmente, con una velocidad de hasta 11 Mbps, 54 Mbps y 300 Mbps, respectivamente.

En la actualidad ya se maneja también el estándar IEEE 802.11a, conocido como WIFI 5, que opera en la banda de 5 GHz y que disfruta de una operatividad con canales relativamente limpios. La banda de 5 GHz ha sido recientemente habilitada y, además, no existen otras tecnologías (Bluetooth, microondas, ZigBee, WUSB) que la estén utilizando, por lo tanto existen muy pocas interferencias. Su alcance es algo menor que el de los estándares que trabajan a 2.4 GHz (aproximadamente un 10%), debido a que la frecuencia es mayor (a mayor frecuencia, menor alcance).

Un primer borrador del estándar IEEE 802.11n que trabaja a 2.4 GHz y a una velocidad de 108 Mbps. Sin embargo, el estándar 802.11g es capaz de alcanzar ya transferencias a 108 Mbps, gracias a diversas técnicas de aceleramiento. Actualmente existen ciertos dispositivos que permiten utilizar esta tecnología, denominados Pre-N.

3.5.6. Cámara

La mayoría de los teléfonos móviles actuales cuentan con cámaras, las cuales permiten capturar tanto fotos como videos. La mayoría de las cámaras de teléfonos son más simples que las cámaras digitales especializadas. Se nota sus limitaciones en tomas con poca luz.

Las fotografías y videos tienen diferentes calidades dependiendo de las características de la cámara del dispositivo, así por ejemplo el HTC Magic toma fotos con

una calidad 3.15 MPx[5], con una resolución de 2048x1536 pixeles y autofocus, mientras que los videos son QVGA[6] a 15fps[7]

3.5.7. CPU

La unidad central de procesamiento o CPU (por el acrónimo en inglés de central processing unit), o simplemente el procesador o microprocesador, es el componente del computador y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos. Desde mediados de los años 1970, los microprocesadores de un solo chip han reemplazado casi totalmente todos los tipos de CPU, y hoy en día, el término "CPU" es aplicado usualmente a todos los microprocesadores.

La CPU de los dispositivos móviles claramente no posee la potencia que tienen los actuales PC. Por ejemplo, el Smartphone HTC Magic viene con un Procesador Qualcomm MSM 7200A 528 MHz, en cambio ahora se pueden encontrar PC de 4 GHz o más.

[5] Un megapíxel o megapixel (Mpx) equivale a 1 millón de píxeles (a diferencia de otras medidas usadas en la computación en donde se suele utilizar la base de 1024, en lugar de 1000, para los prefijos debido a su conveniencia con el uso del sistema binario. Usualmente se utiliza esta unidad para expresar la resolución de imagen de cámaras digitales; por ejemplo, una cámara que puede tomar fotografías con una resolución de 2048 × 1536 píxeles se dice que tiene 3,1 megapíxeles (2048 × 1536 = 3.145.728).

[6] Quarter Video Graphics Array (también conocido como Quarter VGA o QVGA) es un término usado para referirse a una pantalla de ordenador con una resolución de 320x240 píxeles

[7] Las imágenes por segundo (en inglés frames per second, FPS) es la medida de la frecuencia a la cual un reproductor de imágenes genera distintos fotogramas (frames).

CUARTO CAPÍTULO – PRIMER INCREMENTO

4.1. Diseño

Esta sección presenta el diseño que se creó en la primera iteración para ser implementado, en forma de diagramas UML, específicamente diagrama de casos de uso y diagrama de clases. Además se expone el entorno argumental del juego, diseñado en la primera iteración también.

4.1.1. Guión del juego

El comienzo para el desarrollo del juego fue el describir el mundo en el que se insertaría al jugador. El crear un guión fue una necesidad para darle una base al diseño de los elementos que contiene el mundo.

La historia se basa en el gran misterio que encierra la leyenda sobre una isla embrujada y el porqué del protagonista en esta isla. Para su desarrollo se utilizó diversos fragmentos de historias e imágenes que representaban el concepto a desarrollar (ver Figura 10).

El guión se irá revelando a lo largo del desarrollo del juego, pero el pie que da comienza a esta historia es el siguiente:

Hace incontables siglos una leyenda seduce a reyes y cazadores de tesoros. Historias sobre una isla errante llena de los más maravillosos tesoros de ayer y hoy.

Antiguos documentos señalan que la isla de verdad existe y aparece sólo una vez cada era, en ella existe un extraño castillo cuyo interior está adornado con tesoros de distintas culturas y épocas de la historia de la humanidad. Sin embargo no es posible acceder a sus mayores secretos puesto que el castillo está repleto de trampas contra intrusos.

1719 – Era dorada de la piratería

Un capitán surca los mares en busca de tesoros y sueños en los océanos. Sin embargo, la mala fama de este capitán hizo que su tripulación desatara una rebelión contra él. Fue arrojado a su suerte en la soledad del Atlántico Sur.

Este despierta aturdido en una isla… "¡Milagro divino!" piensa él. Camina unos metros y encuentra decenas de viejos esqueletos humanos desparramados en la extensa vegetación allí presente.

Armado con su espada recorre otros metros más, con mucha precaución… "¡matar o morir!". Pero ante sus ojos apareció algo que le hizo temblar. Un enorme y extraño castillo apareció de entre los árboles… ¡la leyenda es real!

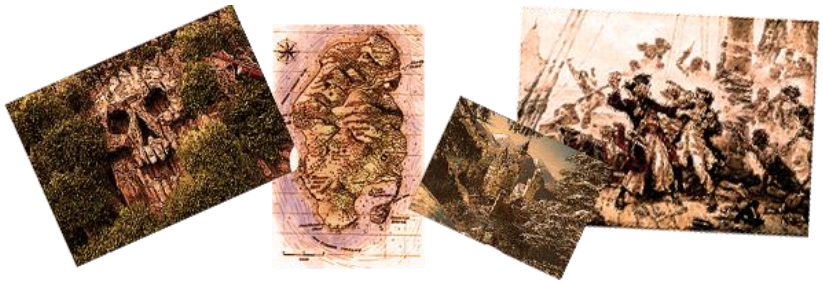

Figura 10: Algunas imágenes que inspiraron el guión del juego

En el transcurso del juego el protagonista encontrará pistas que le indicarán la verdadera historia de la isla:

La isla en un principio fue el trono de un poderoso rey hace miles de años, hasta que un terrible terremoto dividió su reino y su castillo se separó de la tierra, convirtiéndose en un trozo de tierra suelto que deambuló por los mares. La isla perdida fue a parar a las orillas de otro poderoso reino, donde su rey asombrado de las maravillas del castillo decidió convertir esa isla en su nuevo trono, embelleciendo el castillo con su riqueza.

Un día la isla comenzó nuevamente su marcha con todos sus habitantes adentro, hasta parar en un misterioso torbellino en medio del océano. La isla fue tragada, pero increíblemente el agua no tocaba su interior, puesto que por extrañas razones físicas el torbellino mantuvo a la isla en su centro, así como en una especie de burbuja dentro de sí. Así la isla se mantuvo intacta mientras sus habitantes hallaban la muerte por asfixia.

Después de algunos siglos esta isla logra escapar del torbellino para seguir con su marcha y repetir la historia nuevamente. Distintos reinos y eras, pero siempre la codicia como protagonista. Reyes agrandaron cada vez más este castillo y lo repletaron de tesoros y con trampas lo hicieron impenetrable, sólo para encontrar la muerte tiempo después. Sus almas no descansan en paz por lo que sus guerreros cuidan el palacio aún después de muertos. Dentro se contienen los más grandes tesoros y misterios de la humanidad, quien sabe qué puede haber en lo más profundo de este.

Esta historia se repitió por muchos siglos… ¡y sigue repitiéndose!

4.1.2. Diagrama de casos de uso

El siguiente diagrama muestra gráficamente los casos de uso asociados a este proyecto y además muestra la interacción de los actores con estos. En este proyecto sólo existe un actor que interactúa con el juego, el jugador.

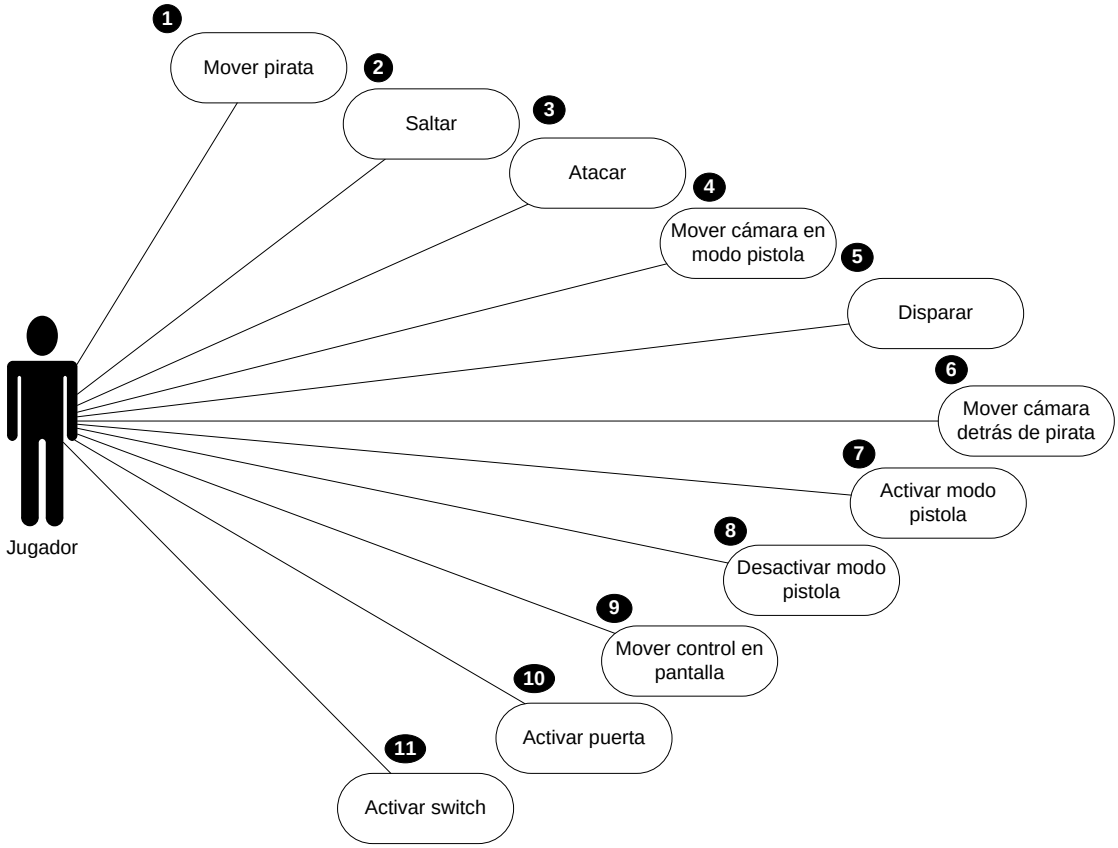

Figura 11: Diagrama de casos de uso

4.1.3. Diagrama de clases

El diagrama de clases describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos.

En la Figura 12 se muestra el diagrama de clases simplificado, con todas sus clases, y desde la Figura 13 a la Figura 17 se muestran los detalles de cada clase.

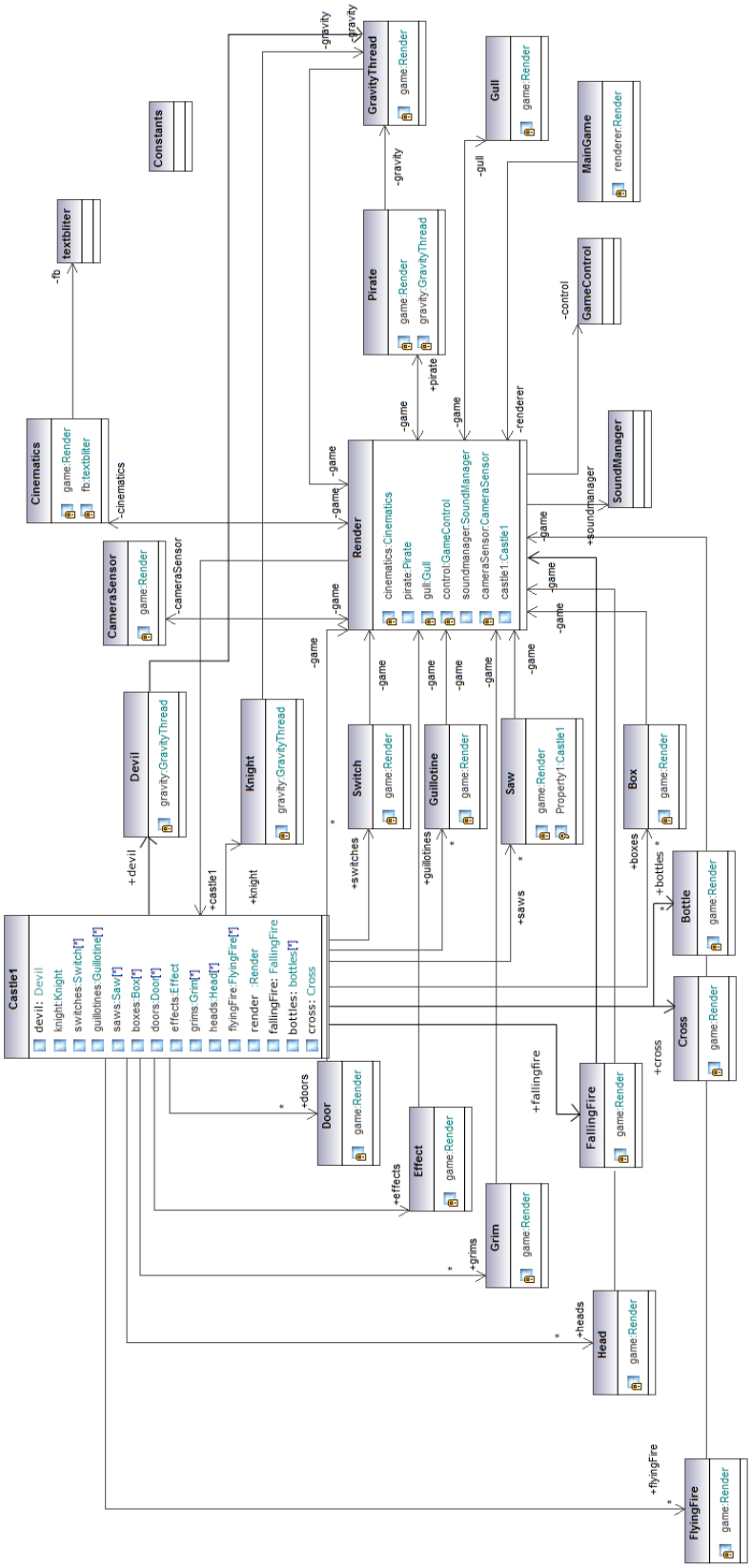

Figura 12: Diagrama de clases simplificado

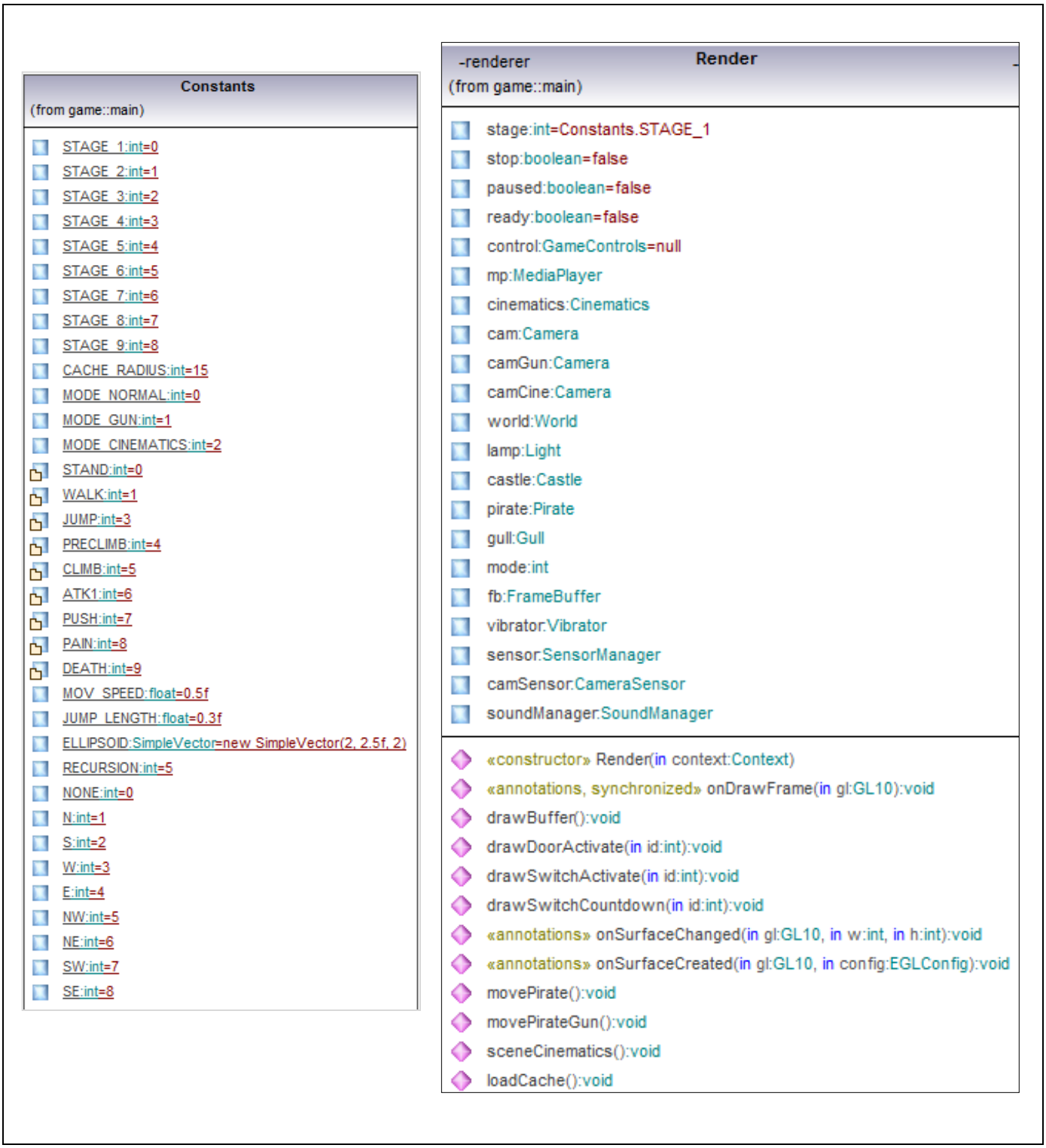

Figura 13: Clases Constants y Render

Figura 14: Clases Castle, Effect, CameraSensor y Cinematics

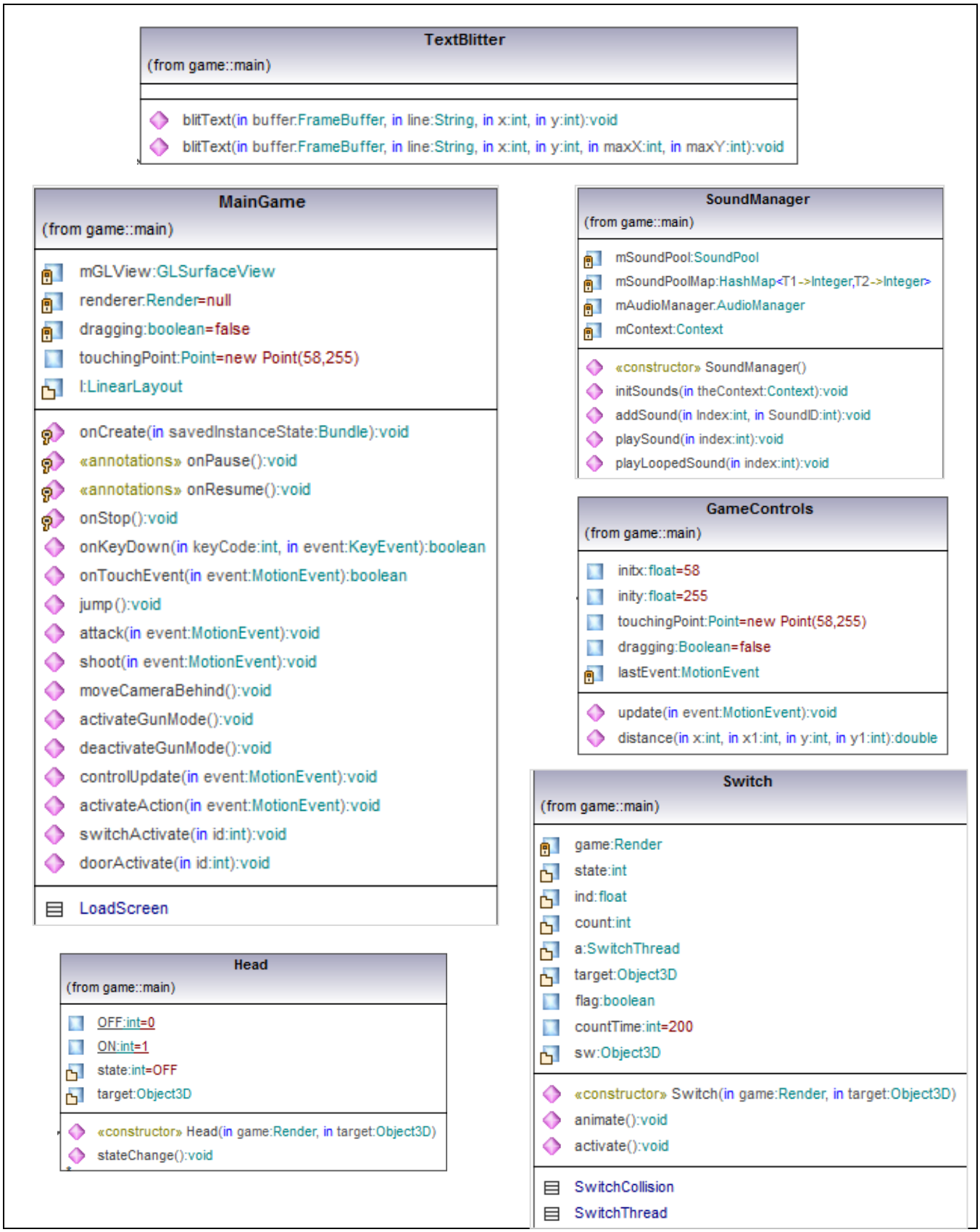

Figura 15: Clases TextBlitter, MainGame, SoundManager, GameControl, Switch y Head.

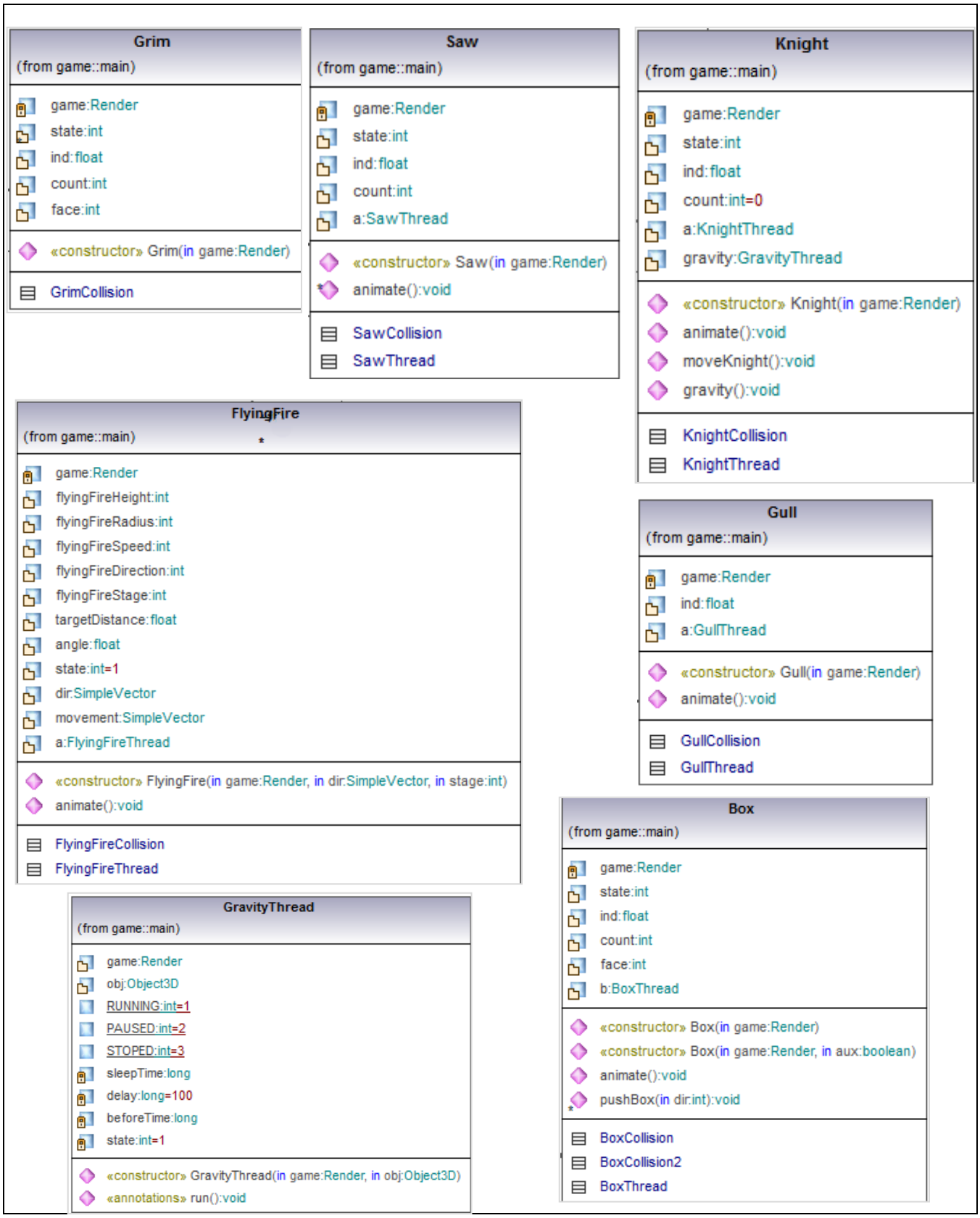

Figura 16: Clases Grim, Saw, Knight, FlyingFire, Gull, GravityThread y Box.

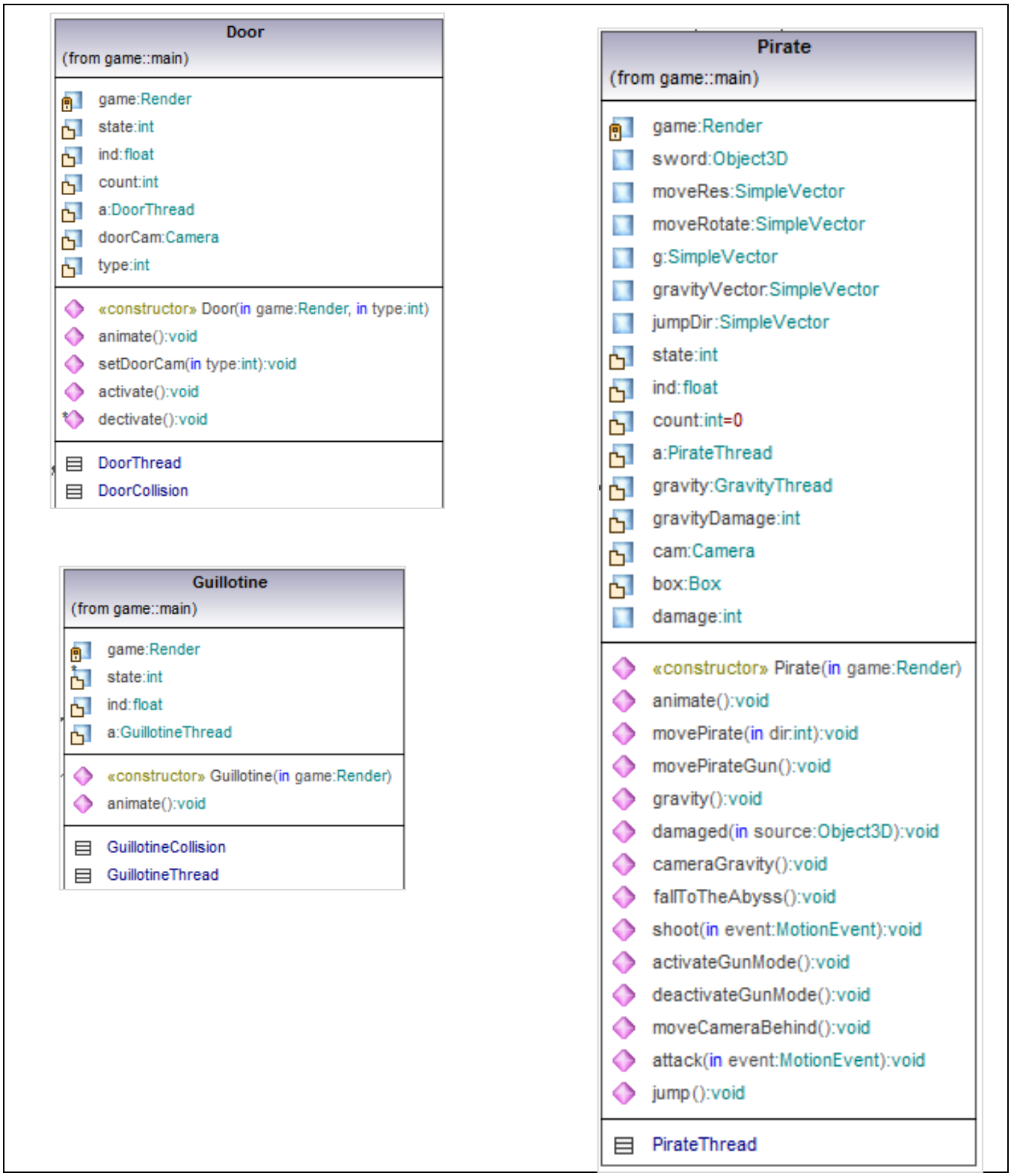

Figura 17: Clases Door, Pirate y Guillotine.

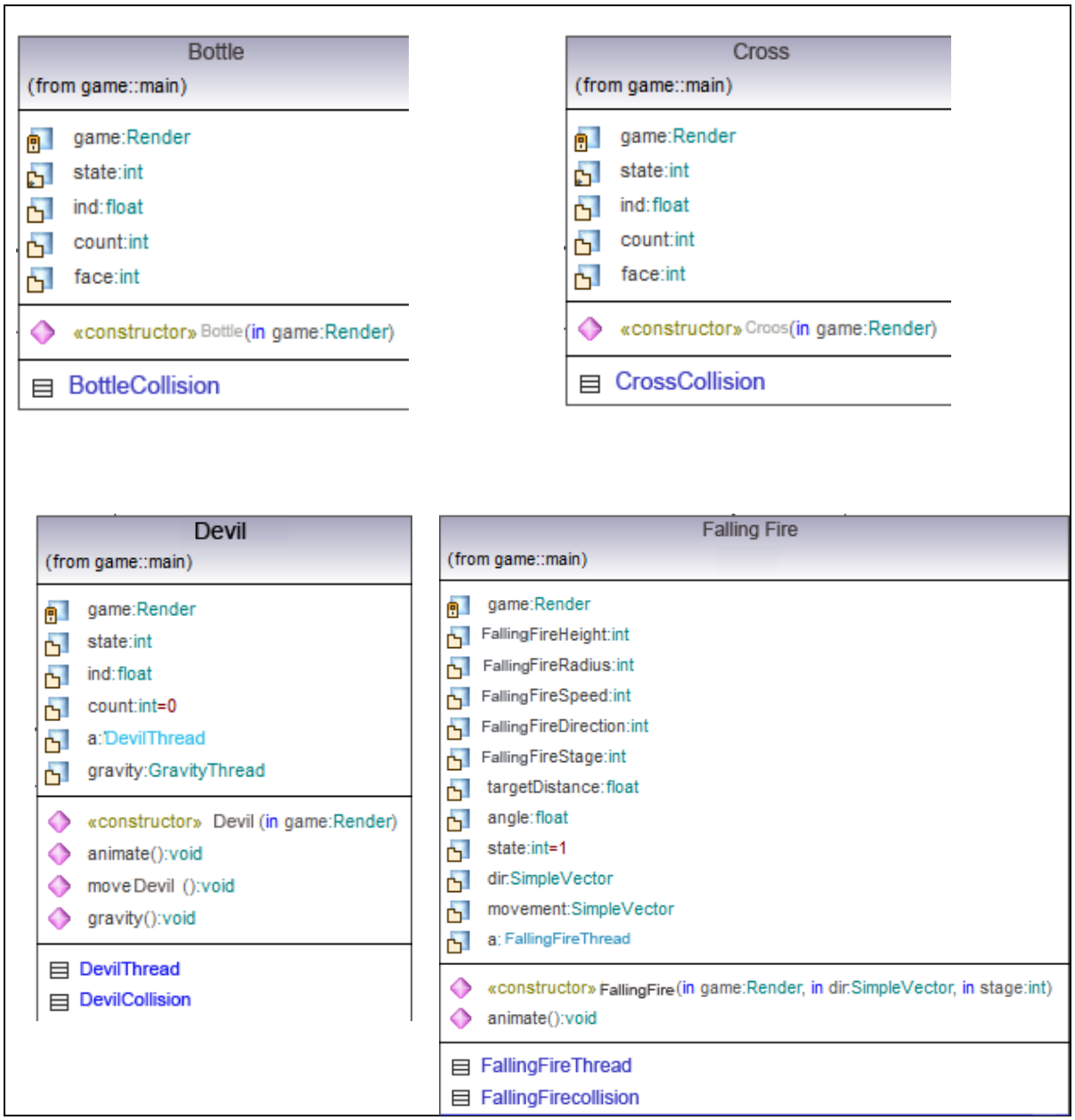

Figura 18: Clases Bottle, Cross, Devil y FallingFire.

4.1.4. Descripción de las clases

Constants

Esta clase contiene todas las constantes utilizadas en el software, valores que no cambiarán y ayudarán a hacer más estándar y entendible la codificación del programa.

GameControls

Clase que se encarga de actualizar el estado del pad virtual en la pantalla, según como se presione la sobre ella.

Door

Clase que como su nombre lo indica representa una puerta; en si es un modelo 3D por lo que extiende de la clase *Object3D*. En esta clase se importa el modelo de la puerta y le aplica la textura. Controla el estado de este objeto, abierto – cerrado, las animaciones, y las colisiones que ocasiona este objeto.

Adicionalmente, esta clase contiene una clase anidada llamada *DoorThread*, que extiende de *Thread*, y al ser arrancado ejecuta un loop constante llamando al método que anima la puerta hasta que se finaliza la aplicación. Este hilo es creado y ejecutado en la clase Door.

También posee otra clase anidada llamada *DoorCollision*, que implementa una interfaz especial de la librería jPCT-AE llamada *CollisionListener*. Esta interfaz provee a la clase de un método llamado collision, que se ejecuta cada vez que existe una colisión entre la puerta y algún elemento del mundo.

Switch

Clase que representa un switch en el mundo, el cual activa puertas y mecanismos cuando el jugador se acerca a este y presiona en el ícono de "activar switch". Esta clase extiende de *Object3D*, y controla el estado, animación y colisiones de este objeto.

Adicionalmente, esta clase contiene una clase anidada llamada *SwitchThread*, que extiende de *Thread*, y al ser arrancado ejecuta un loop constante llamando al método que anima el switch hasta que se finaliza la aplicación. Este hilo es creado y ejecutado en la clase *Switch*.

También posee otra clase anidada llamada *SwitchCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre el switch y algún elemento del mundo.

Head

Clase que representa la estatua de la cabeza de un faraón. Este objeto se mantiene estático en el aire. Si se le dispara, ocurre una acción determinada.

Grim

Clase que representa a la estatua de un hombre encapuchado sosteniendo un libro abierto. Si el pirata toca a este objeto, ocurre una acción determinada.

Box posee una clase anidada llamada *GrimCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre la estatua y algún elemento del mundo.

FlyingFire

Clase que representa un pequeño fuego volador que da vueltas aleatoriamente por una habitación, y que de a poco se va acercando al pirata hasta tocarlo, lo que le produce daño.

FlyingFire posee una clase anidada llamada *FlyingFireCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre el fuego y algún elemento del mundo.

Adicionalmente, esta clase contiene una clase anidada llamada *FlyingFireThread*, que extiende de *Thread*, y al ser arrancado ejecuta un loop constante llamando al método que anima el fuego hasta que se finaliza la aplicación. Este hilo es creado y ejecutado en la clase *FlyingFire*.

FallingFire

Clase que representa un pequeño fuego que cae desde el techo. Este fuego es utilizado por el diablo, jefe final del juego.

FallingFire posee una clase anidada llamada *FallingFireCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre el fuego y algún elemento del mundo.

Adicionalmente, esta clase contiene una clase anidada llamada *FallingFireThread*, que extiende de *Thread*, y al ser arrancado ejecuta un loop constante llamando al método que anima el fuego hasta que se finaliza la aplicación. Este hilo es creado y ejecutado en la clase *FallingFire*.

Box

Clase que representa una caja en el mundo, la cual se mueve constantemente en forma vertical, esto para poder aplastar al pirata si este está debajo de la caja cuando esta caiga. Esta clase extiende de *Object3D* y al igual que las anteriores controla el estado, animaciones y colisiones de este objeto.

Box posee una clase anidada llamada *BoxCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre la caja y algún elemento del mundo.

Adicionalmente, esta clase contiene una clase anidada llamada *BoxThread*, que extiende de *Thread*, y al ser arrancado ejecuta un loop constante llamando al método que anima la caja hasta que se finaliza la aplicación. Este hilo es creado y ejecutado en la clase *Box*.

Guillotine

Clase que representa una guillotina colgada del techo de una habitación y que describe el movimiento de un péndulo. Si el pirata toca esta guillotina recibirá daño. Esta clase extiende de *Object3D* y controla la animación, estado y colisiones de este objeto.

Adicionalmente, esta clase contiene una clase anidada llamada *GuillotineThread*, que extiende de *Thread*, y al ser arrancado ejecuta un loop constante llamando al método que anima la guillotina hasta que se finaliza la aplicación. Este hilo es creado y ejecutado en la clase *Guillotine*.

También posee otra clase anidada llamada *GuillotineCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre la guillotina y algún elemento del mundo.

Saw

Clase que representa una sierra en el piso la cual se mueve en forma recta, y dificulta el paso del pirata. Si el pirata toca la sierra recibirá un daño. Esta clase extiende de *Object3D* y controla la animación, estado y colisiones de este objeto.

Adicionalmente, esta clase contiene una clase anidada llamada *SawThread*, que extiende de *Thread*, y al ser arrancado ejecuta un loop constante llamando al método que anima la sierra hasta que se finaliza la aplicación. Este hilo es creado y ejecutado en la clase *Saw*.

También posee otra clase anidada llamada *SawCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre la sierra y algún elemento del mundo.

Bottle

Objeto que representa una botella de ron que el pirata utiliza para recuperar vida.

Posee una clase anidada llamada *BottleCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre la botella y algún elemento del mundo, esto para detectar si el pirata la toca.

Cross

Objeto que representa una cruz. Aparece en el mundo cuando se derrota a un caballero.

Posee una clase anidada llamada *CrossCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre la cruz y algún elemento del mundo, esto para detectar si el pirata la toca.

Effect

Clase que representa un efecto en el mundo, que puede ser una chispa de luz, fuego o polvo. Esta clase extiende de *Object3D* y controla la animación y estado de este. La particularidad de esta clase es que el objeto que contenga tendrá el atributo de *Billboarding*.

Adicionalmente, esta clase contiene una clase anidada llamada *animateEffect*, que extiende de *Thread*, y al ser arrancado ejecuta un loop constante llamando al método que anima el efecto hasta que se finaliza la aplicación. Este hilo es creado y ejecutado en la clase *Effect*.

También posee otra clase anidada llamada *FireCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre el fuego y

algún elemento del mundo. Esto es para detectar cuando el fuego choca con el pirata, ya que esto le produce daño.

Pirate

Clase que representa al personaje principal del juego: el pirata William El Desquiciado. Esta clase extiende de *Object3D*, por lo que también contiene el modelo a ser mostrado en el mundo. En el constructor se carga el modelo, se le aplica su textura y se le asocia como hijo al *Object3D sword*, que es la espada del pirata que siempre tiene en su mano derecha.

Como métodos posee todas las acciones que realiza el pirata: caminar, saltar y atacar, además de otras acciones propias del juego, como cambiar posición de la cámara e iniciar el modo pistola. Además está el método de gravedad (que simula la gravedad del pirata), gravedad de la cámara (mueve la cámara en forma vertical u horizontal dependiendo de la posición del pirata) y animación (que controla la animación del pirata).

Además esta clase posee una clase anidada llamada *PirateThread*, que extiende de *Thread* y es un hilo que se crea e inicializa en el constructor de la clase *Pirate*, y ejecuta un loop infinito llamando al método animación del pirata, esto para mantener su animación por todo el transcurso del juego.

Knight

Clase que representa al caballero, enemigo del pirata en el juego. Esta clase extiende de *Object3D*, por lo que también contiene el modelo a ser mostrado en el mundo. En el constructor se carga el modelo y se le aplica su textura.

Como métodos posee todas las acciones que realiza el enemigo además de su animación.

Además esta clase posee una clase anidada llamada *KnightThread*, que extiende de *Thread* y es un hilo que se crea e inicializa en el constructor de la clase *Knight*, y ejecuta un loop infinito llamando al método de animación del enemigo, esto para mantener su animación constante.

También posee otra clase anidada llamada *KnightCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre el enemigo y algún elemento del mundo.

Devil

Clase que representa al jefe final del juego, el diablo. Esta clase extiende de *Object3D*, por lo que también contiene el modelo a ser mostrado en el mundo. En el constructor se carga el modelo y se le aplica su textura.

Como métodos posee todas las acciones que realiza el diablo además de su animación.

Además esta clase posee dos clases anidadas que extienden de Thread, llamadas *DevilThread* y *DevilThreadMode*. *DevilThread* es un hilo que se crea e inicializa en el constructor de la clase *Devil*, y ejecuta un loop infinito llamando al método de animación del enemigo, esto para mantener su animación constante. *DevilThreadMode*, al igual que DevilThread, es un hilo que se crea e inicializa en el constructor de la clase *Devil*, y ejecuta un loop infinito, con una duración de 20 segundos cada iteración. En cada iteración cambia el modo de ataque del enemigo.

También posee otra clase anidada llamada *DevilCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre el enemigo y algún elemento del mundo.

GravityThread

Esta clase extiende de *Thread* y es un hilo que simula la gravedad de un objeto. En el constructor acepta pasarle como parámetro un objeto de tipo *Pirate* o de tipo *Knight*. Al objeto que se pasó como parámetro se le llamará a su método de gravedad en un loop infinito hasta el fin de la aplicación luego de ser arrancado el hilo.

Gull

Clase que representa a la gaviota, compañera del pirata durante el juego. Esta clase extiende de *Object3D*, por lo que también contiene el modelo a ser mostrado en el mundo. En el constructor se carga el modelo y se le aplica su textura.

Esta clase posee una clase anidada llamada *GullThread*, que extiende de *Thread* y es un hilo que se crea e inicializa en el constructor de la clase *Gull*, y ejecuta un loop infinito llamando al método de animación de la gaviota, esto para mantener su animación constante.

También posee otra clase anidada llamada *GullCollision*, que implementa la interfaz *CollisionListener*, que detecta cada vez que hay una colisión entre la gaviota y

algún elemento del mundo. Este método es utilizado para cuando el pirata se encuentra con la gaviota por primera vez solamente, ya que después el objeto gaviota pasa a ser hija del pirata.

Castle

Clase que se encarga de cargar, agregar y eliminar los elementos de cada escenario del juego. Aquí es donde se contienen los métodos que van realizando el cargado de escenarios en el caché.

Posee la clase anidada *FlyingFireSpreading*, que extiende de *Thread* y se encarga de crear los fuegos voladores en las habitaciones donde se supone se crean estos.

Cinematics

Clase que se encarga de preparar y animar las escenas de "cinemáticas", que son los diálogos y otros eventos que ocurren sin interacción con el jugador. Para escribir los diálogos en pantalla, esta clase requiere ayuda de la clase *TextBlitter*.

TextBlitter

Clase que define el área donde se escribirá un texto en la pantalla y posee el método que permite escribir en la pantalla un *String*. Para lograr el efecto de escritura de texto caracter por caracter, se llama a este método para que dibuje el primer caracter, luego se llama de nuevo para que dibuje el primer más el segundo caracter, y así sucesivamente hasta completar el texto.

SoundManager

Clase que carga y maneja los sonidos que tendrá el juego, tanto efectos de sonido como la música.

CameraSensor

Clase que implementa la interfaz *SensorEventListener* que sirve para capturar los movimientos, posiciones y rotaciones del teléfono. Esta clase captura los movimientos del celular y los traduce en rotaciones de la cámara actual del juego. Esta clase es la que permite la rotación de la cámara en función a la posición y movimiento del celular en el modo pistola del juego.

Render

Clase que implementa la interfaz *GLSurfaceView.Renderer*, que es propia de OpenGL y es la encargada de hacer el renderizado de la escena en pantalla. Posee 3 métodos: *onSurfaceCreated*, *onSurfaceChanged* y *onDrawFrame*.

onSurfaceCreated se ejecuta cuando se crea la superficie para renderizar. *onSurfaceChanged* se ejecuta cuando se modifica esta superficie (por ejemplo cuando se cambia la orientación del teléfono). *onDrawFrame* se ejecuta para renderizar la escena actual en la superficie. *onDrawFrame* se llama continuamente mientras la superficie esté creada, lo que crea un loop que se puede utilizar para ejecutar todos los métodos que requieran llamadas continuas y hace posible el movimiento de los elementos del mundo.

MainGame

Clase que extiende de la clase de Android *Activity*. Un activity crea una ventana donde el programador puede ubicar una interfaz de usuario. Esta es la clase principal del juego, ya que acá se carga como interfaz de usuario la clase *Render*, que realiza el renderizado del mundo.

En esta clase además se ubica el método *onTouchEvent*, que captura el evento cuando se toca la pantalla del teléfono, vital para crear la interación usuario-aplicación del juego.

4.2. Implementación

En esta sección se expondrá la implementación de elementos necesarios para la creación del tipo de juego planeado para el proyecto. Estos son los elementos desarrollados durante la primera iteración.

4.2.1. Librería jPCT-AE

La implementación del juego fue llevada a cabo con la librería gratuita jPCT-AE, que es una versión para Android de la librería jPCT.

jPCT fue creada en base a OpenGL ES, que es una versión para portátiles del popular motor 3D OpenGL. Esta librería fue pensada especialmente para el desarrollo de juegos, con herramientas acorde para el control de la cámara, importación de modelos 3D, detección de colisiones y otros.

jPCT-AE es una versión más compacta de jPCT, esto por las obvias diferencias en procesamiento entre un teléfono móvil y un computador. jPCT-AE excluye algunas clases de jPCT, pero mantiene las esenciales para desarrollar una aplicación 3D.

La librería posee una página web en donde se encuentran las versiones de esta para su descarga, además de toda su documentación, un foro y ejemplos creados por usuarios de jPCT. La página es *www.jpct.net*.

4.2.2. Cámara

La implementación de la cámara se llevó a cabo mediante la clase *Camera* de la librería jPCT – AE.

Esta clase representa la posición y la dirección del observador o cámara en la escena actual. Esta también contiene información sobre el campo de vista actual, en ingles FOV Field Of View [8]. Un aspecto relevante de esta clase es su matriz de rotación la cual se aplica a todos los elementos del mundo. Esto es importante cuando se escoge el ángulo

[8] Es el ángulo o área observable del mundo. Así por ejemplo dependiendo de la localización de los ojos en los animales, el FOV varia; los humanos poseen un FOV cercano a 180º y las aves casi 360º.

de rotación de la cámara: una rotación (virtual) con un ángulo w en un determinado eje sobre la cámara, es lo mismo que rotar el mundo alrededor del eje con un ángulo w.

El constructor de la clase Camera es el siguiente:

- `public Camera()`

  Este constructor crea una nueva Camera con una vista por defecto: dirección a lo largo del eje z, posición en el origen y un FOV de 1.25 unidades.

El juego contiene principalmente dos cámaras, normal y modo pistola, representado por las siguientes constantes:

```
public static final int MODE_NORMAL = 0;
public static final int MODE_GUN = 1;
```

El **modo normal** es utilizado en la mayoría del juego. Este posee una cámara situada atrás del personaje, unas 15 unidades atrás inicialmente, y acompaña al personaje simulando una cámara en tercera persona.

El **modo pistola**, es utilizado cuando se desea disparar a un enemigo o activar un mecanismo que de otra forma sería imposible. Este modo aprovecha el acelerómetro de los aparatos con sistema operativo Android, para mover la cámara según se mueve el sensor, simulando estar dentro del universo virtual.

La cámara dentro del juego se crea mediante el siguiente código:

```
   //añadir cámara
1. cam = world.getCamera();
2. SimpleVector moveCam = new SimpleVector(0,0,0);
3. cam.moveCamera(moveCam, 0);
4. cam.moveCamera(Camera.CAMERA_MOVEOUT, Constants.CAM_DIST);
5. camGun = new Camera();
```

La primera línea instancia la cámara obteniendo la cámara actual que contiene el mundo. La segunda línea utiliza la clase *SimpleVector* de la librería jPCT-AE, la cual representa un vector tridimensional básico con coordenadas x,y y z. Este vector es utilizado para mover la cámara al origen. En la línea tres se mueve la cámara utilizando el método *moveCamera*, el cual traslada la cámara según la dirección dada por el vector y las unidades indicadas, en este caso cero. La cuarta línea mueve la cámara utilizando el modo *CAMERA_MOVEOUT* el cual mueve la cámara hacia a tras respecto a la posición actual según la constante *CAM_DIST* que indica un movimiento de 15 unidades. La última línea crea la cámara utilizada en el modo pistola, con los parámetros por defecto (dirección a lo largo del eje z, posición en el origen y un FOV de 1.25).

4.2.3. Escenario

El escenario en el código es representado por el objeto *world*, el cual es una instancia de la clase *World* de la librería jPCT-AE. Esta clase es la más importante dentro de esta librería orientada a juegos, ya que contiene todos los elementos del mundo, luces, modelos, cámaras, etc.

El constructor de esta clase es el siguiente:

- `public World()`

  Este constructor crea un nuevo mundo, como una sala vacía que debe ser llenada con objetos y luces.

  Dentro de los métodos utilizados se encuentran los siguientes:

- `void addObject(Object3D obj)`

  Añade un objeto a la colección de objetos del mundo. Parámetros: *obj*, representa al objeto a añadir al mundo

- `Camera getCamera()`

  Obtiene la cámara actual del mundo

- `void setCameraTo(Camera cam)`

  Cambia la cámara actual del mundo por la cámara dada

- `void buildAllObjects()`

  Llama la función *build()* para cada objeto del mundo

Modelado

El juego se desarrolla por completo dentro del extraño castillo, debido a esto el diseño del escenario se centró en el diseño del castillo mismo.

Para el modelado del castillo se utilizó el software de diseño 3D Google Sketchup (ver Figura 19).

Figura 19: Primer prototipo fachada exterior castillo

El castillo según la historia cuenta con un sinfín de trampas, las cuales están ubicadas de manera dispersa en el interior del castillo.

4.2.4. Personajes

Programación de personajes

En el código del video juego cada personaje y objeto es representado por la una clase que extiende de *Object3D*.

La clase *Object3D* es usada para representar objetos en tres dimensiones. Estos objetos son usualmente añadidos a una instancia de *World* para ser renderizados [9]. Un objeto de esta clase sólo puede ser añadido a una instancia de *World* a la vez. Estos objetos pueden ser definidos como hijos o padres, para así obtener un comportamiento jerárquico.

La clase *Object3D* posee cuatro constructores, donde el utilizado en este proyecto es el siguiente:

- `Public Object3D(Object3D obj)`
    - Este constructor trabaja de forma similar al método *cloneObject()* pero permite extender *Object3D* y continuar usando los métodos estáticos para cargar o usar primitivas utilizando la sentencia *super(Object3D)* en el constructor de la clase. Parámetros: *obj*, el objeto desde el cual se construye el objeto.

[9] Renderizado (render en inglés) es un término usado en jerga informática para referirse al proceso de generar una imagen desde un modelo.

Algunos métodos usados de esta clase son:

- `void addChild(Object3D obj)`

  Define un objeto como hijo de este objeto. Parámetros: *obj*, es el objeto que será hijo del objeto en cuestión.

- `void animate(float index)`

  Calcula una nueva forma para este objeto basada en los frames de la animación en la secuencia dada, donde *index* es un valor entre cero y uno.

- `void setCulling(boolean mode)`

  Habilita o deshabilita el renderizado de las caras invertidas. Parámetros: *mode*, indica si se habilita o deshabilita el modo.

- `void translate(SimpleVector trans)`

  Mueve el objeto en el mundo modificando la matiz de translación del objeto 3D. Parámetros: *trans*, vector de translación.

Para animar cada personaje se utilizo un hilo (*Thread*), el cual se encarga se revisar cada determinado tiempo el estado del personaje, para así actualizar su animación. Por otro lado, estos hilos contienen sus propias variables de estado, que se sincronizan con los estados del juego, para así poder terminarlos cuando el juego lo requiera. Como se muestra en el siguiente código de ejemplo, mientras el hilo tenga el estado *RUNNING*, este seguirá actualizará el estado de animación y gravedad de la cámara que se explicará en el segundo incremento.

```
while (state == RUNNING) {
      beforeTime = System.nanoTime();
      synchronized(game){
            pirate.animate();
            pirate.cameraGravity();
}
```

Modelado de personajes

Para el diseño 3D de personajes se utilizó el software Google SketchUp y para la animación Misfit Model 3D. El número de polígonos de los modelos fueron reducidos utilizando el programa VIZup, esto porque ante modelos muy detallados la fluidez del juego se ve muy afectada y además que en la pequeña pantalla de los teléfonos no se aprecian todos los detalles de los modelos.

- **Pirata**

Es el personaje principal del juego. Se pensó en él como un ser humano amargado, malvado y despreciable, al punto de incluso ser odiado por su tripulación.

El modelo 3D contiene alrededor de 800 triángulos y una textura que se mapea al modelo completo, esto es debido a la limitante del formato md2, el cual estipula que un modelo sólo puede contener un material.

Para la animación se construyó, gracias al software Misfit Model 3D, un esqueleto mediante "bones", los cuales influyen sobre ciertos vértices, auto asignado o definidos por el usuario (ver Figura 20). Así, de esta forma se construye una serie de "skeletal animation" que facilitan de gran forma la tarea de animar al modelo 3D. Luego de construir las secuencias de animación se procede a transformarlas en animaciones de frames, ya que el formato md2 sólo soporta está clase de animaciones.

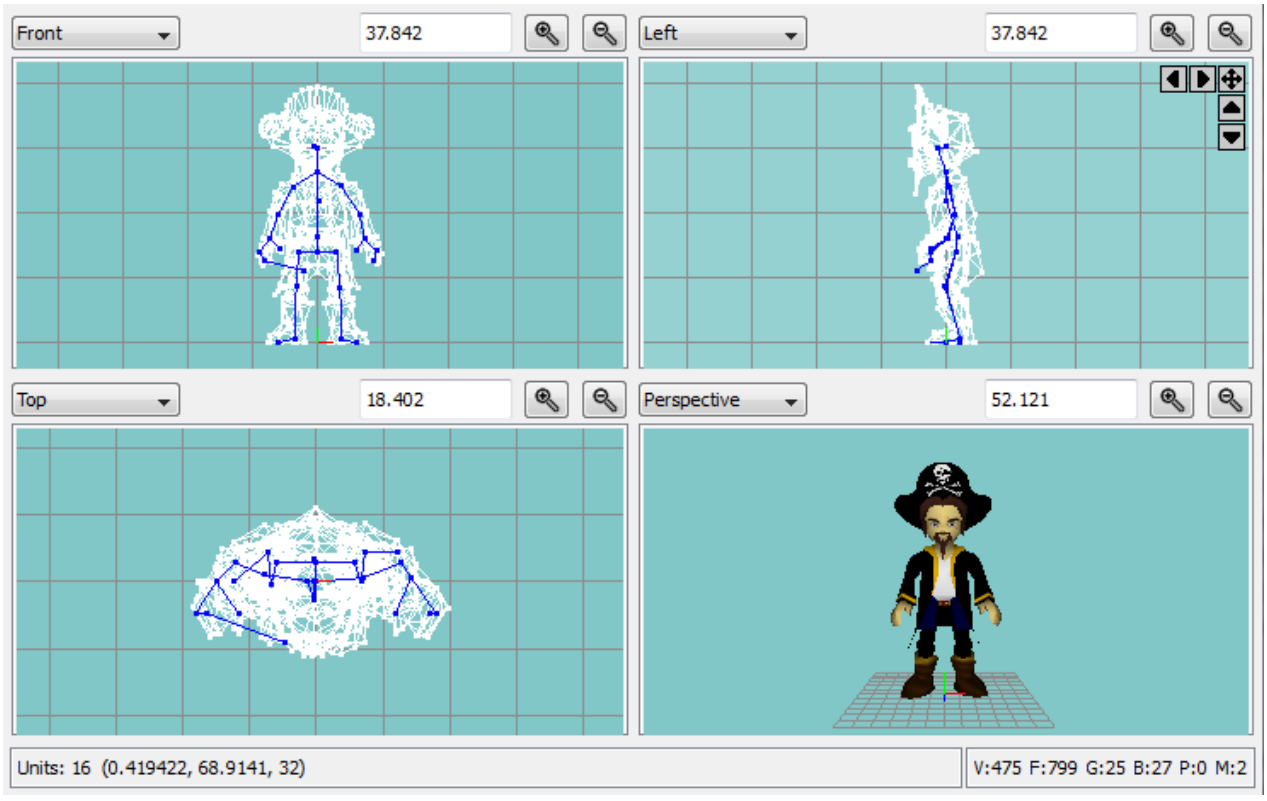

Figura 20: Modelado de personaje principal en software Misfit Model 3D

Para el propósito del juego se definió una serie de animaciones necesarias:

- *Stand*: el personaje está de pie sin hacer ningún movimiento, sólo respirar.
- *Walk*: el personaje camina (ver Figura 21).

- *Jump*: el personaje salta.
- *Atk*: el personaje ataca mediante su espada.
- *Death*: el personaje muere.
- *Pain*: el personaje es dañado.

Figura 21: Frames de animación Walk

- **Gaviota**

Esta gaviota es un personaje secundario que ayuda al protagonista dando pistas sobra la historia y la forma de resolver ciertos acertijos.

El modelo contiene 250 triángulos y contempla las mismas animaciones que el pirata, necesarias para la interacción con el personaje principal, más las propias del modelo en cuestión (ver Figura 22).

Figura 22: Algunos frames de Pirata y Gaviota en animación Walk

- **Caballero**

Este personaje proviene de la edad medieval y fue un antiguo guardián del castillo que ya dejó este mundo y aun resguarda este, cumpliendo su misión más allá de la muerte.

Al igual que el pirata, este modelo contiene ciertas animaciones: *Stand*, *Walk*, *Atk* (ver Figura 23), *Death* y *Pain*.

Figura 23: Algunos frames de la animación Atk

- **Diablo**

Este personaje es un extraño demonio encerrado en la última habitación del juego (ver Figura 24). Es el último enemigo y el más difícil de vencer, por lo que corresponde al jefe final.

Al igual que el cabellero, este modelo contiene ciertas animaciones: *Stand*, *Walk*, *Atk*, *Death* y *Pain*.

Figura 24: Imagen del diablo

4.2.5. Objetos

Espada

Este objeto es la preciada espada del personaje principal, la cual permite al personaje defenderse ante los enemigos.

Este objeto es especial, ya que al ser necesario el moverse a la par que el personaje que la porta, se construyó un modelo con la misma cantidad y coordenadas de animación que el Pirata, aumentado en gran medida la complejidad de la animación (ver Figura 25).

Debido a lo anterior la espada cuanta con las siguientes secuencia de animación: *Stand*, *Walk*, *Jump*, *Atk*, *Death* y *Pain*.

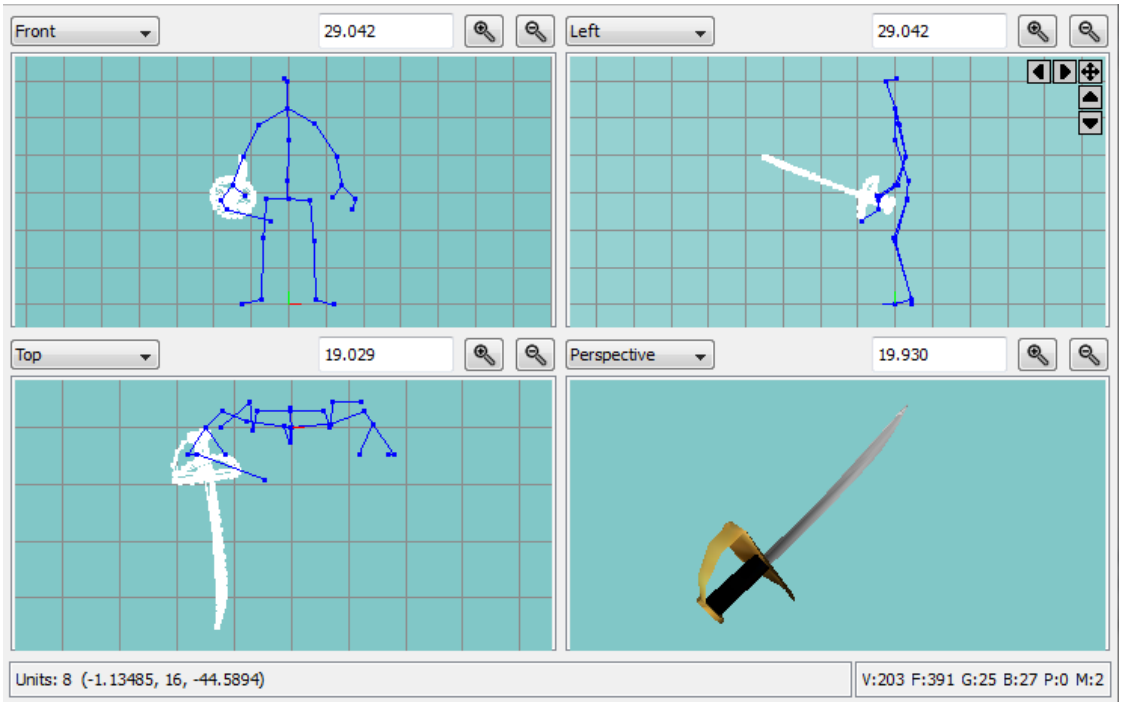

Figura 25: Modelado Espada en software Misfit Model 3D

Objetos pertenecientes al escenario

- **Puerta**

  Este objeto es un modelo md2 que representa a la puerta para pasar de una habitación a otra en el juego. Para tal propósito se debió realizar las animaciones correspondientes a una puerta: abrir y cerrar.

- **Switch**

Este objeto es un modelo md2 que representa un switch el cual es utilizado para abrir puertas o activar mecanismos. Contiene una animación la cual simula la activación del switch.

- **Caja**

  Este objeto representa una caja de madera, la cual sube y baja constantemente para aplastar al pirata si este se encuentra debajo al caer esta.

- **Fuego**

  Este objeto representa una llama la cual daña al personaje principal. Es un plano que cambia de textura simulando que está animado.

- **Guillotina**

  Elemento que representa una guillotina colgada desde el techo de una habitación y que se mueve en forma de péndulo, la cual debe ser esquivada por él protagonista. Es un modelo md2 con una animación que simula el efecto péndulo.

- **Sierra**

  Es un elemento en forma circular que se mueve por el piso de una habitación en forma recta, y al hacer contacto con el protagonista causa daño. Es un modelo md2 con la animación que simula el movimiento por el piso.

- **Cabeza**

  Es un objeto que representa la estatua de la cabeza de un faraón que flota en el aire. Ejecuta una acción determinada cuando se le dispara.

4.2.6. Control

Aprovechando las capacidades del dispositivo móvil se implementó un pad virtual, es decir, un control que interactúa con el usuario mediante la pantalla táctil. Debido a lo anterior fue necesario desarrollar algún mecanismo de retroalimentación con el usuario.

El mecanismo implementado reacciona al tocar la pantalla en un área específica, movimiento el pad virtual en la dirección presionada (ver Figura 26).

Figura 26: Ejemplo de distintas posiciones del pad virtual

Para calcular la forma en que el pad se debía mover se utilizó el siguiente código:

```
angle = Math.atan2(touchingPoint.y - inity,touchingPoint.x - initx)
                  /(Math.PI/180);
if ( dragging ){
      d=distance(touchingPoint.x,(int)initx,touchingPoint.y,(int)inity);
      r=d/40;
if(r>1) r=1;
      touchingPoint.y = inity+(Math.sin(angle*(Math.PI/180))*40*r);
      touchingPoint.x = initx+(Math.cos(angle*(Math.PI/180))*40*r);
}else if (!dragging){
      touchingPoint.x = initx;
      touchingPoint.y = inity;
}
```

En el código anterior primero se calcula el ángulo en que el pad se moverá y se guarda en en *angle*, donde la variable touchingPoint guarda el punto donde se presiono la pantalla (x,y) e initx e inity, representan el centro del pad (ver Figura 27).

Luego si se presionó la pantalla se debe calcular el movimiento del pad virtual, para lo cual primero se calcula la distancia entre el punto presionado y el centro del pad, para luego calcular la razón entre este valor y el valor máximo designado para la distancia entre el pad y el centro, en este caso 40 px, si este valor es superior la razón será 1. Después se actualiza el valor de la variable *touchingPoint* al punto deseado, en un radio menor o igual a 40px desde el centro del pad virtual.

Si no se presionó la pantalla, el pad vuelve al centro seteando los valores de *touchingPoint* al centro del pad.

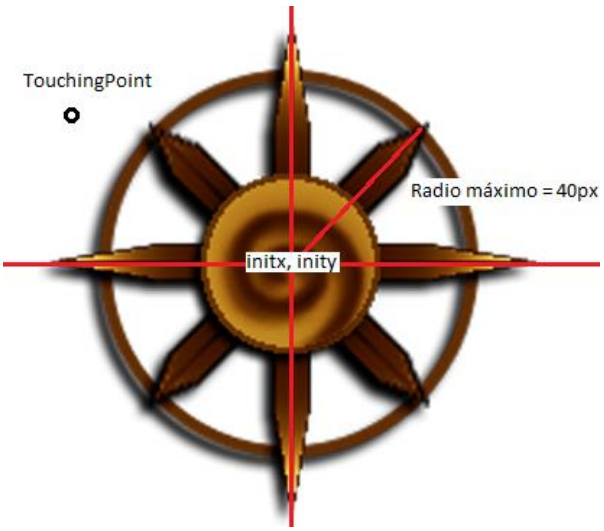

Figura 27: Figura explicativa del cálculo del movimiento del pad virtual

4.3. Técnicas utilizadas

4.3.1. Billboard

La palabra billboard significa "cartelera". En términos generales billboard es un gráfico en dos dimensiones situado en un mundo de tres dimensiones y orientado de manera que mire hacia la cámara y siempre se vea de frente. Para lograr esto se debe girar el cartel cuando giremos o movamos la cámara, o cuando movamos el propio cartel (ver Figura 28).

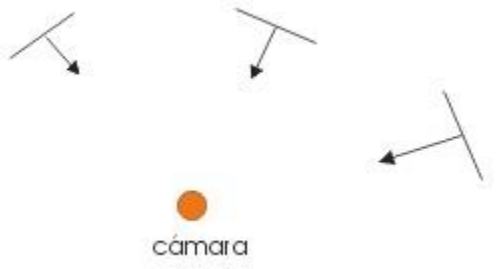

Figura 28: Ejemplo explicativo de cómo se orienta un plano mediante billboard

En la librería usada, este comportamiento está implementado mediante la clase *Object3D* y permite activar o desactivar este comportamiento mediante el método *setBillboarding(boolean)*.

En el software esta técnica se utilizó para realizar efectos con texturas, por ejemplo el fuego (ver Figura 29).

Figura 29: Ejemplo de utilización de billboard en el fuego

QUINTO CAPÍTULO – SEGUNDO INCREMENTO

5.1. Diseño

Esta sección continúa con el diagramado creado para este proyecto. Aquí se encuentra el detalle de los casos de uso (definidos en la Figura 11) y los diagramas de secuencia y colaboración para cada uno de estos.

5.1.1. Detalle casos de uso

A continuación se detallarán todos los casos de uso definidos para este proyecto, a partir de la Tabla 3 hasta la Tabla 13.

Tabla 3: Detalle caso de uso 01 – Mover pirata

CU – 01 Mover pirata		
Usuario(s)	:	Jugador
Propósito	:	Mover al pirata en la dirección deseada a través del mundo.
Precondiciones	:	El juego no debe estar en modo pistola ni mostrando cinemáticas, además el pirata no debe estar recibiendo daño, atacando ni saltando.
Postcondiciones	:	El pirata se ha movido unas pocas unidades en la dirección seleccionada.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador desea mover al pirata y presiona el control en una determinada posición.		
		2.- El sistema gira al pirata a la dirección que se presionó en el control y mueve al pirata en la determinada posición según la dirección actual a la que mira la cámara y ejecuta la animación de caminar del pirata.
		3.- El sistema mueve la cámara en la misma dirección a la que se movió el pirata, la misma cantidad de unidades y este caso de uso termina.
SECUENCIAS ALTERNATIVAS		
		3.a.- Si el movimiento fue a la izquierda o a la derecha, la cámara en vez de moverse, se gira a la misma dirección del movimiento del pirata y este caso

		de uso termina.
	2.b.-	El sistema detecta una colisión entre el pirata y algún elemento del mundo y mueve al pirata hasta que este colisiona con el objeto.
	2.b.a.-	La colisión fue contra una caja, por lo que el sistema cambia la animación de caminar del pirata a la de mover una caja.
	3.b.-	El sistema mueve la cámara en la misma dirección a la se movió el pirata, la misma cantidad de unidades y este caso de uso termina.
	3.c.-	El sistema detecta una colisión de la cámara con algún elemento del mundo y se mueve en la misma dirección a la que se movió el pirata, pero hasta que colisione con el elemento y este caso de uso termina.

Tabla 4: Detalle caso de uso 02 – Saltar

CU – 02 Saltar		
Usuario(s)	:	Jugador
Propósito	:	Hacer saltar al pirata
Precondiciones	:	El juego no debe estar en modo pistola ni mostrando cinemáticas, además el pirata no debe estar recibiendo daño ni atacando.
Postcondiciones	:	El pirata ha saltado y se ha movido unas unidades hacia la dirección a la cual estaba mirando.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador desea hacer saltar al pirata y presiona el botón para saltar.		
		2.- El sistema mueve al pirata en la dirección a la que estaba mirando, mueve al pirata hacia arriba y ejecuta la animación de salto. Cuando se llega a la mitad de la secuencia, el sistema mueve al pirata hacia abajo hasta que termina la secuencia.
		3.- El sistema mueve la cámara en la misma dirección a la que se movió el pirata, la misma cantidad de unidades y este caso de uso termina.
SECUENCIAS ALTERNATIVAS		
		2.a.- El sistema detecta una colisión entre el pirata y algún elemento del mundo y mueve al pirata hasta que este colisiona con el objeto.
		3.a.- El sistema mueve la cámara en la misma dirección a la se movió el pirata, la misma cantidad de unidades y este caso de uso termina.
		3.b.- El sistema detecta una colisión de la cámara con algún elemento del mundo y se mueve en la misma dirección a la que se movió el pirata, pero hasta que colisione con el elemento.

Tabla 5: Detalle caso de uso 03 – Atacar

CU – 03 Atacar		
Usuario(s)	:	Jugador
Propósito	:	Hacer que el pirata ataque
Precondiciones	:	El juego no debe estar en modo pistola ni mostrando cinemáticas, además el pirata no debe estar recibiendo daño ni atacando.
Postcondiciones	:	El pirata ha atacado en la dirección donde se encuentra el enemigo seleccionado.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador desea atacar a algún enemigo y lo toca en la pantalla del teléfono.		
		2.- El sistema rota al pirata en la posición a la que está el enemigo y ejecuta la acción de ataque y este caso de uso termina.
SECUENCIAS ALTERNATIVAS		
		2.a.- Si el ataque tocó al enemigo, este recibe daño y ejecuta su acción de recibir daño y este caso de uso termina.

Tabla 6: Detalle caso de uso 04 – Mover cámara

CU – 04 Mover cámara en modo pistola		
Usuario(s)	:	Jugador
Propósito	:	Mover la cámara en el modo pistola, para poder apuntar y disparar.
Precondiciones	:	El juego debe estar en modo pistola.
Postcondiciones	:	La cámara del modo pistola está mirando a la posición determinada.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador desea mover la cámara estando en el modo pistola y mueve el teléfono en cualquier eje de rotación.		
		2.- El sistema rota la cámara del juego en la misma forma que fue rotado el teléfono.

Tabla 7: Detalle caso de uso 05 – Disparar

CU – 05 Disparar		
Usuario(s)	:	Jugador
Propósito	:	Disparar en modo pistola
Precondiciones	:	El juego debe estar en modo pistola.
Postcondiciones	:	Se ha disparado a la posición seleccionada y si ha apuntado a algún elemento del juego que reacciona al disparo, ejecuta su reacción determinada.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador desea disparar y toca la pantalla.		
		2.- El sistema hace vibrar el teléfono y calcula a qué elemento golpea el disparo y este caso de uso termina.
SECUENCIAS ALTERNATIVAS		
		2.a.- El sistema determina que el elemento golpeado por el disparo reacciona ante este y ejecuta la reacción determinada del elemento y este caso de uso termina.

Tabla 8: Detalle caso de uso 06 – Mover cámara detrás del pirata

CU – 06 Mover cámara detrás del pirata		
Usuario(s)	:	Jugador
Propósito	:	Mover la cámara detrás del pirata.
Precondiciones	:	El juego no debe estar en modo pistola ni mostrando cinemáticas.
Postcondiciones	:	La cámara está detrás del pirata mirando en la misma dirección a la cual mira el pirata.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador desea posicionar la cámara detrás del pirata y presiona el botón para mover la cámara detrás del pirata.		
		2.- El sistema posiciona la cámara detrás del pirata y apunta el lente hacia la misma dirección a la que el pirata mira y este caso de uso termina.

Tabla 9: Detalle caso de uso 07 – Activar modo pistola

CU – 07 Activar modo pistola	
Usuario(s) : Jugador	
Propósito : Activar el modo pistola	
Precondiciones : El juego no debe estar en modo pistola ni mostrando cinemáticas.	
Postcondiciones : El juego está en modo pistola.	
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA	
Jugador	**Sistema**
1.- Este caso de uso comienza cuando el jugador desea entrar en modo pistola y presiona el botón para acceder al modo pistola.	
	2.- El sistema cambia la cámara actual del juego a la cámara del modo pistola, que es en primera persona. Además borra la interfaz de juego y dibuja el ícono de salida del modo pistola.
	3.- El sistema activa la captura de la posición y el movimiento del teléfono para mover la cámara.

Tabla 10: Detalle caso de uso 08 – Desactivar modo pistola

CU – 08 Desactivar el modo pistola		
Usuario(s)	:	Jugador
Propósito	:	Desactivar el modo pistola y volver al modo normal de juego
Precondiciones	:	El juego debe estar en modo pistola.
Postcondiciones	:	El juego está en el modo normal de juego.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador desea salir del modo pistola y presiona el botón para salir del modo pistola.		
		2.- El sistema cambia la cámara actual del juego a la cámara normal de juego, borra el ícono de salida del modo pistola y dibuja la interfaz del juego.
		3.- El sistema desactiva la captura de la posición y el movimiento del teléfono.

Tabla 11: Detalle caso de uso 09 – Mover control en pantalla

CU – 09 Mover control en pantalla		
Usuario(s)	:	Jugador
Propósito	:	Mover la palanca cuando se toca el control en una dirección.
Precondiciones	:	El juego no debe estar en modo pistola ni mostrando cinemáticas.
Postcondiciones	:	La palanca del control está en la misma posición en la que está el dedo del jugador.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador mueve la palanca del control de la pantalla.		
		2.- El sistema mueve la palanca a la misma posición en la que está el dedo del jugador.

Tabla 12: Detalle caso de uso 10 – Activar puerta

CU – 10 Activar puerta		
Usuario(s)	:	Jugador
Propósito	:	Activar la acción de una puerta.
Precondiciones	:	El juego no debe estar en modo pistola ni mostrando cinemáticas.
Postcondiciones	:	La acción de la puerta fue activada.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador presionó el ícono para activar la acción de una puerta.		
		2.- El sistema ejecuta la animación correspondiente y ejecuta la acción de la puerta, que es abrirse.

Tabla 13: Detalle caso de uso 11 – Activar switch

CU – 11 Activar switch		
Usuario(s)	:	Jugador
Propósito	:	Activar la acción de un switch.
Precondiciones	:	El juego no debe estar en modo pistola ni mostrando cinemáticas.
Postcondiciones	:	La acción del switch fue activada.
DESCRIPCIÓN DETALLADA, SECUENCIA EXITOSA		
Jugador		**Sistema**
1.- Este caso de uso comienza cuando el jugador presionó el ícono para activar la acción de un switch.		
		2.- El sistema ejecuta la animación correspondiente y ejecuta la acción determinada del switch, que puede ser distinto dependiendo del switch.

5.1.2. Diagramas de secuencia

A continuación se mostrarán todos los diagramas de secuencia generados, asociados a cada caso de uso (definidos en la Figura 11). La finalidad de este diagramado es modelar interacción entre objetos en un sistema.

DS: CU – 01 Mover pirata

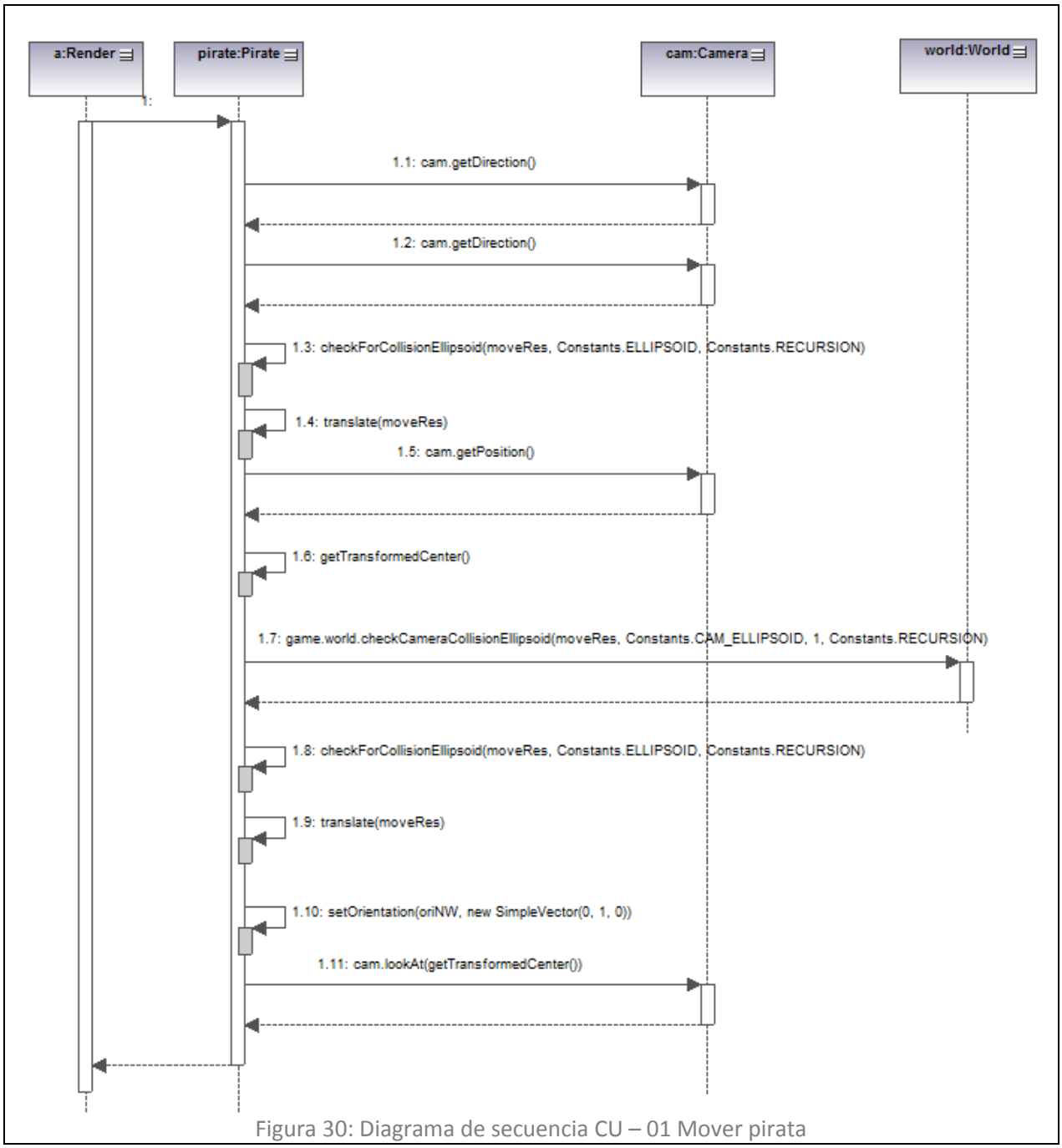

Figura 30: Diagrama de secuencia CU – 01 Mover pirata

DS: CU – 02 Saltar

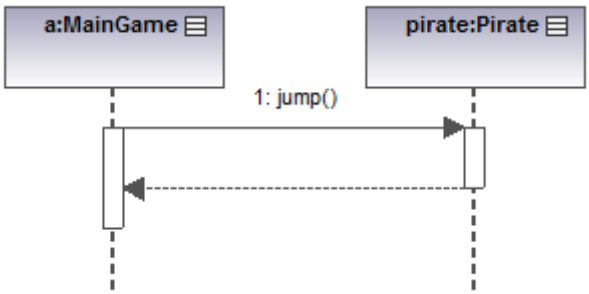

Figura 31: Diagrama de secuencia CU – 02 Saltar

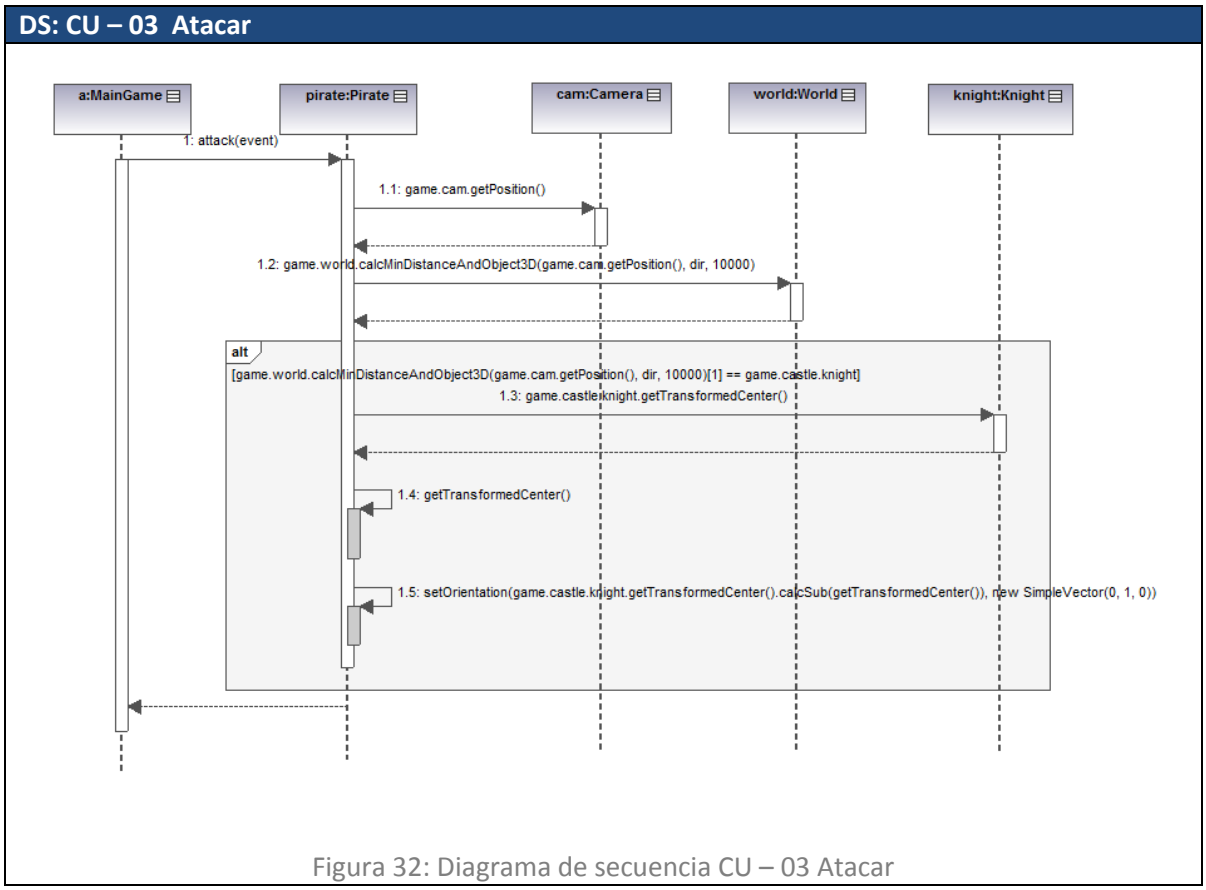

Figura 32: Diagrama de secuencia CU – 03 Atacar

DS: CU – 04 Mover cámara modo pistola

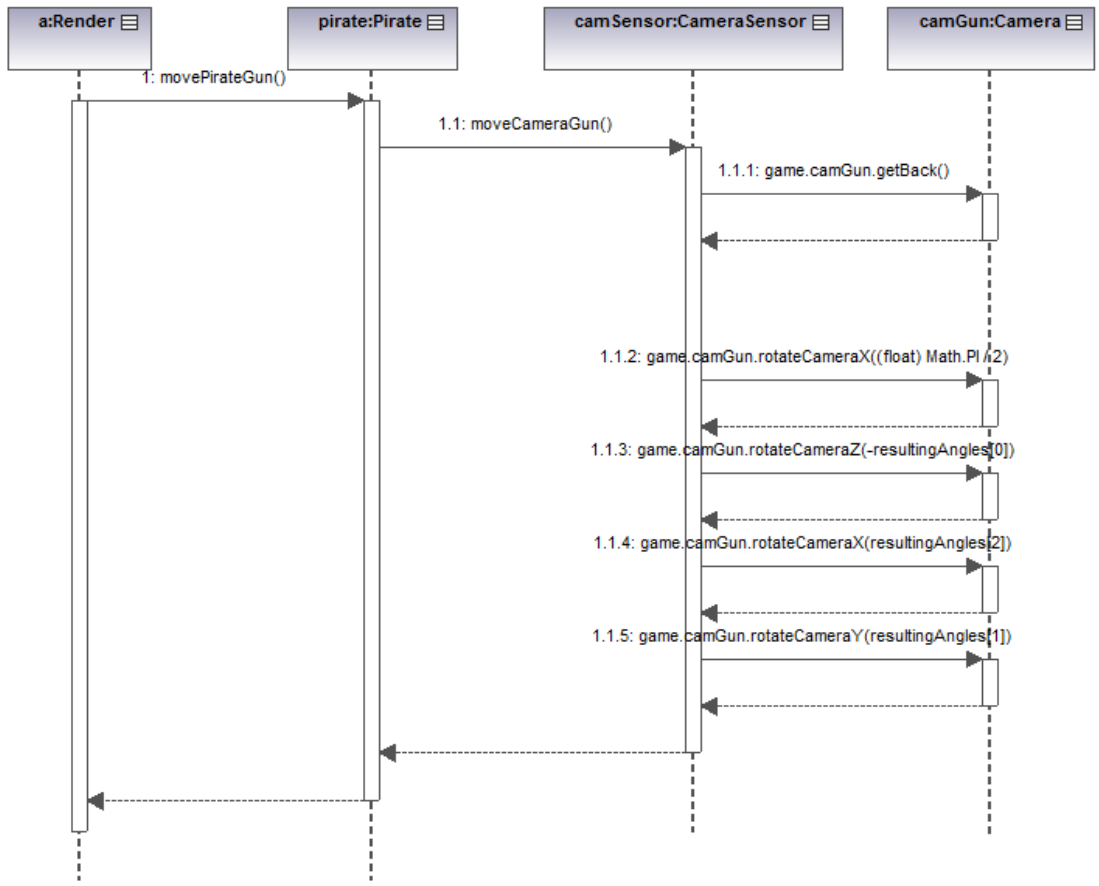

Figura 33: Diagrama de secuencia CU – 04 Mover cámara modo pistola

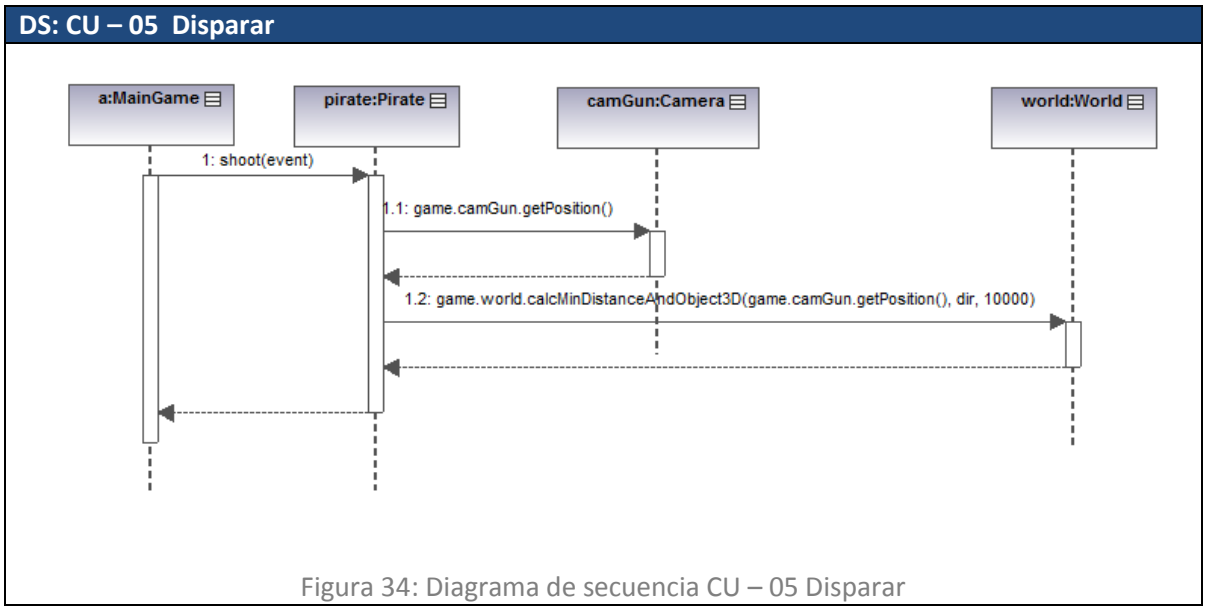

Figura 34: Diagrama de secuencia CU – 05 Disparar

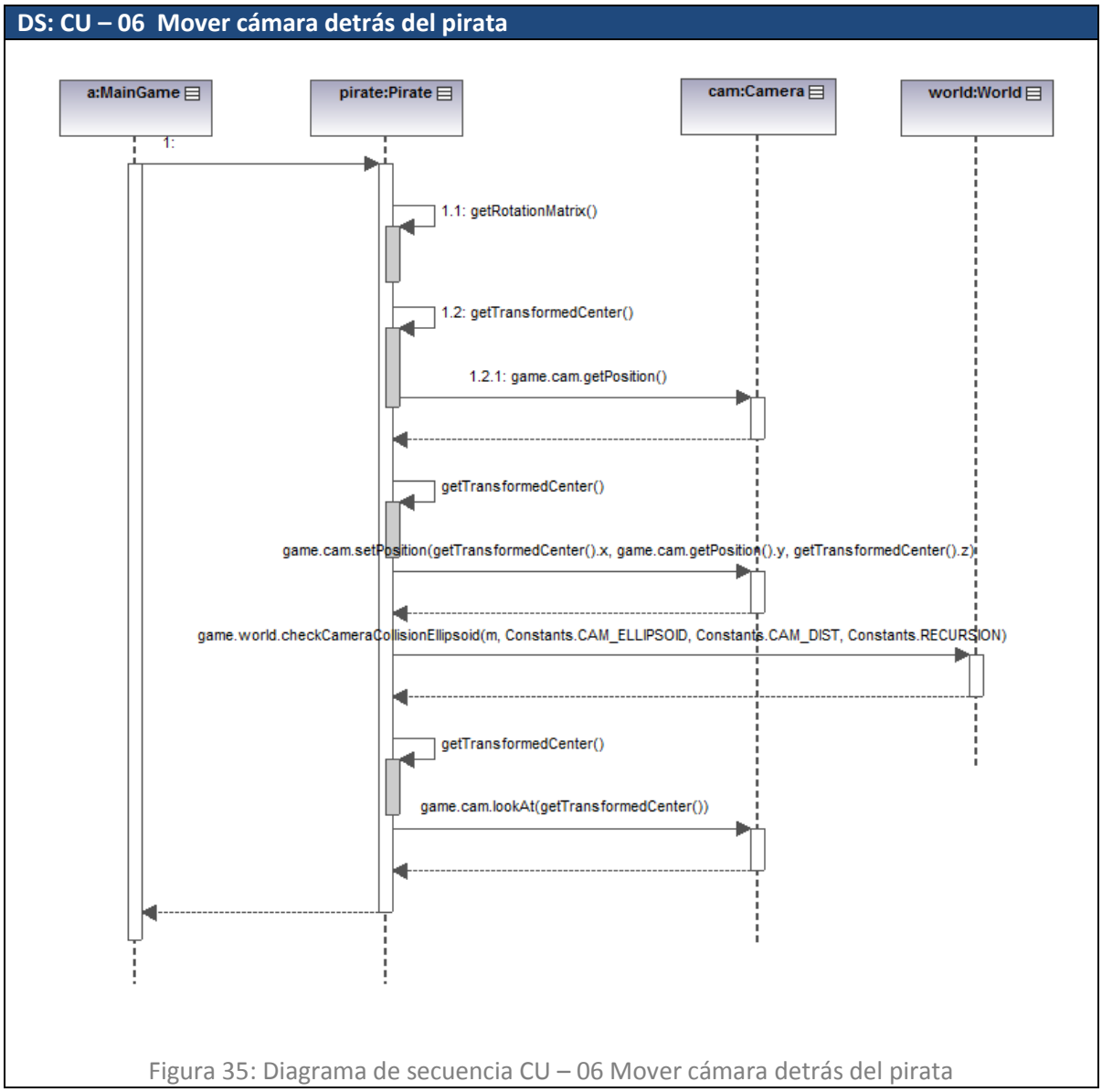

Figura 35: Diagrama de secuencia CU – 06 Mover cámara detrás del pirata

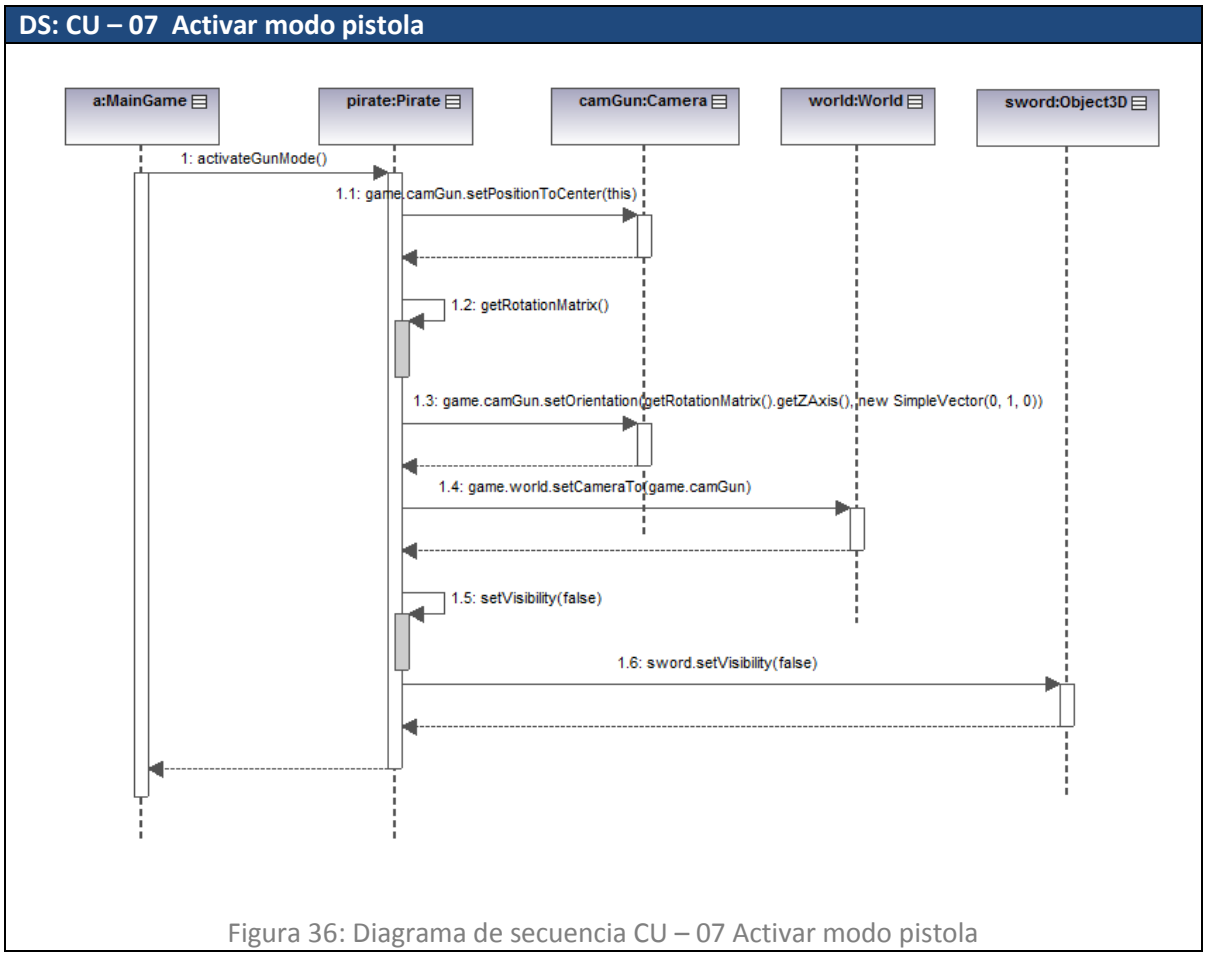

Figura 36: Diagrama de secuencia CU – 07 Activar modo pistola

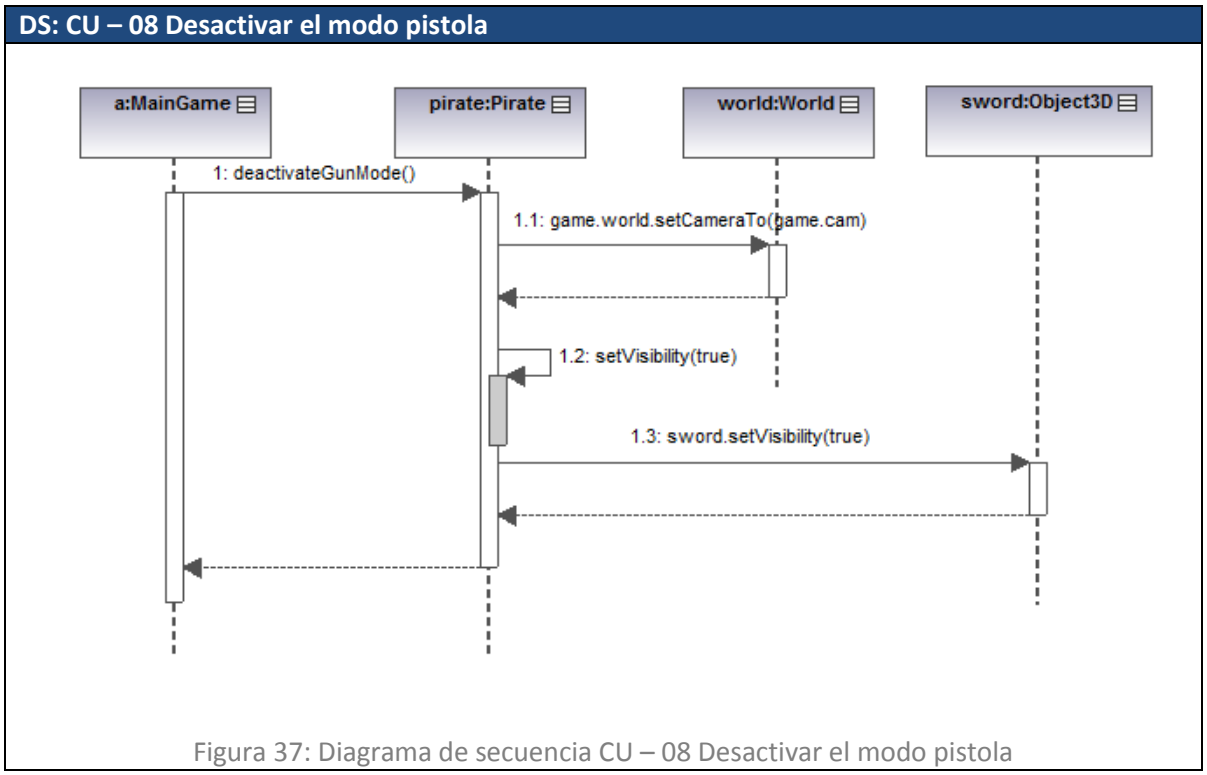

Figura 37: Diagrama de secuencia CU – 08 Desactivar el modo pistola

DS: CU – 09 Mover control en pantalla

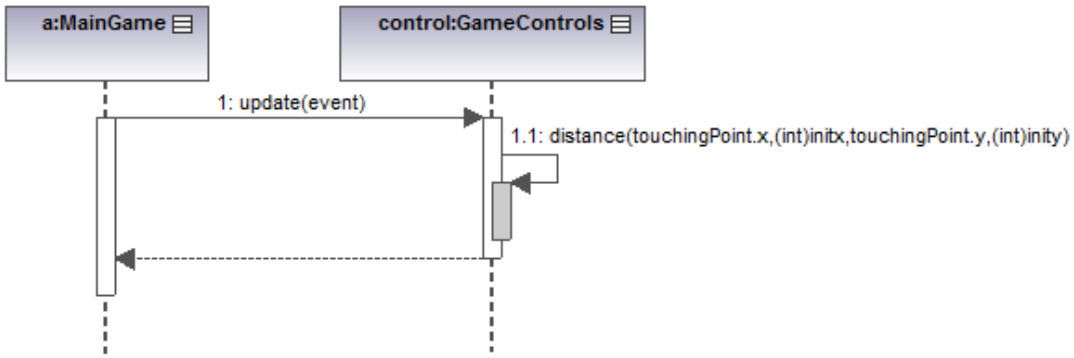

Figura 38: Diagrama de secuencia CU – 09 Mover control en pantalla

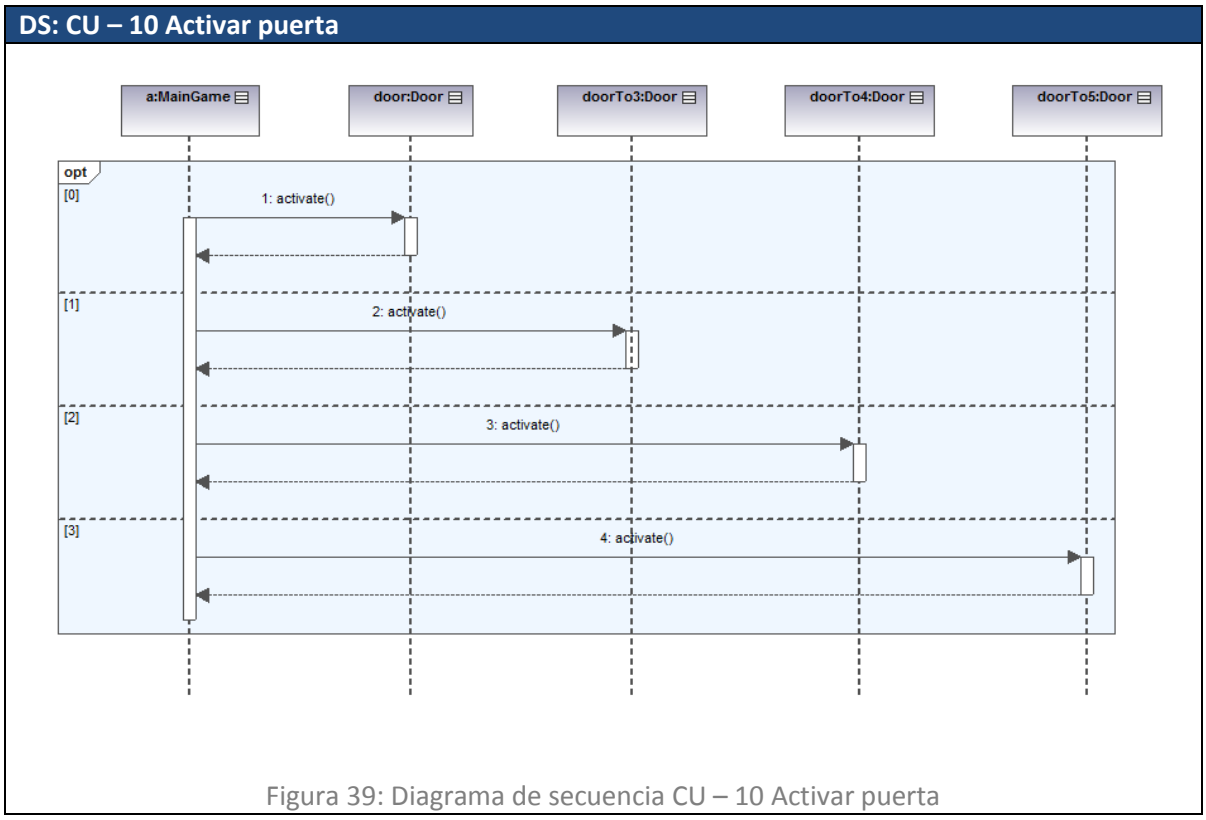

Figura 39: Diagrama de secuencia CU – 10 Activar puerta

DS: CU – 11 Activar switch

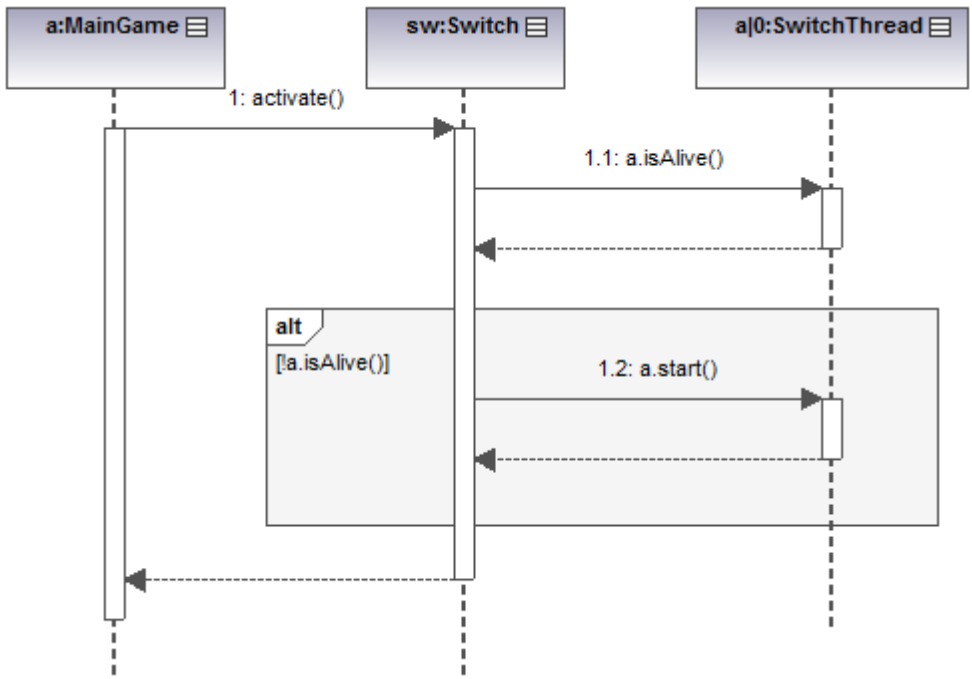

Figura 40: Diagrama de secuencia CU – 11 Activar switch

5.1.3. Diagramas de colaboración

A continuación se mostrarán todos los diagramas de colaboración generados, asociados a cada caso de uso (definidos en la Figura 11). Este tipo de diagrama muestra la interacción entre las clases mediante el paso de mensajes de un objeto a otro.

DC: CU – 01 Mover pirata

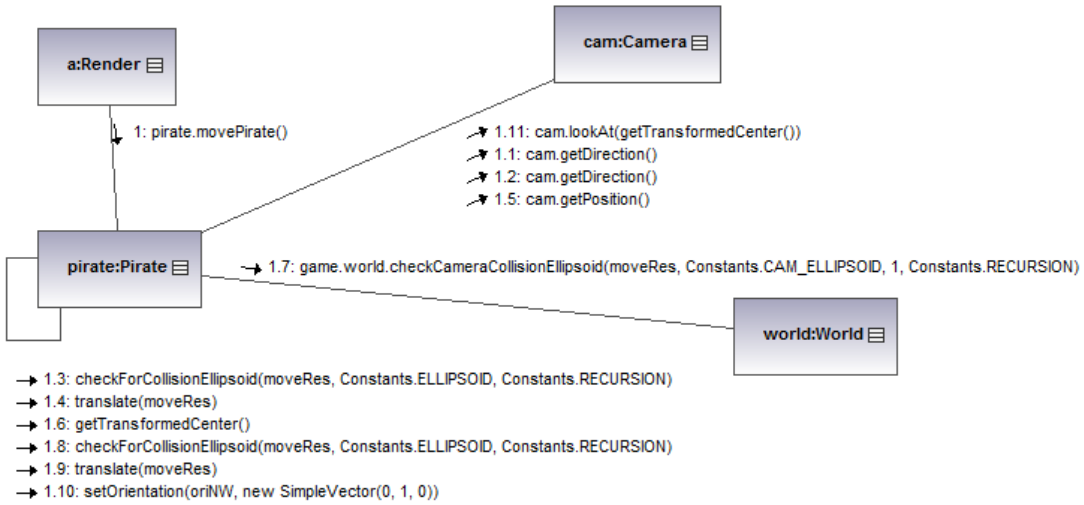

Figura 41: Diagrama de comunicación CU – 01 Mover pirata

DC: CU – 02 Saltar

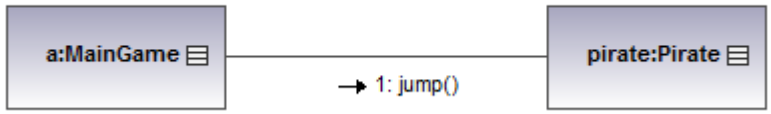

Figura 42: Diagrama de comunicación CU – 02 Saltar

DC: CU – 03 Atacar

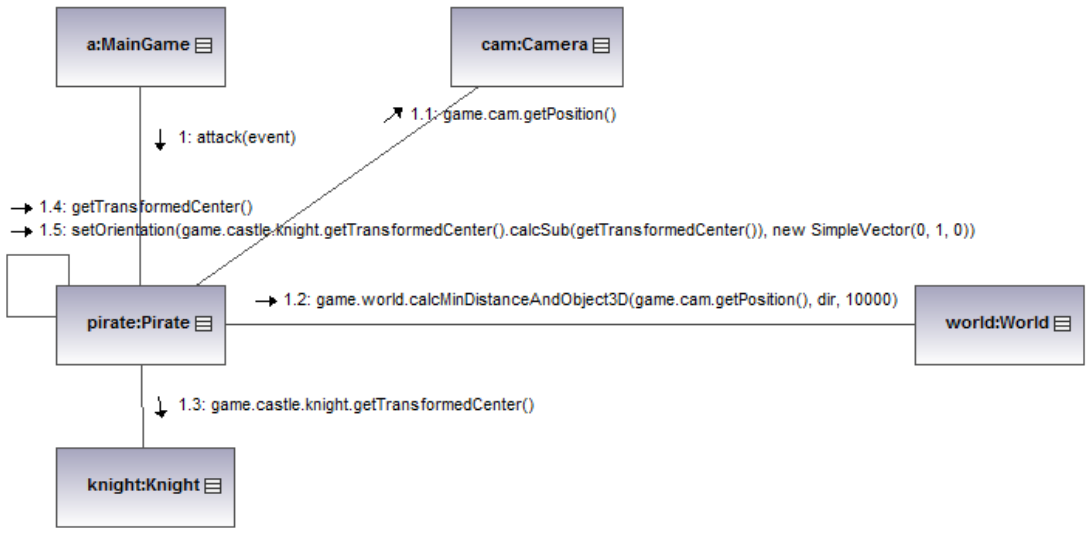

Figura 43: Diagrama de comunicación CU – 03 Atacar

DC: CU – 04 Mover cámara en modo pistola

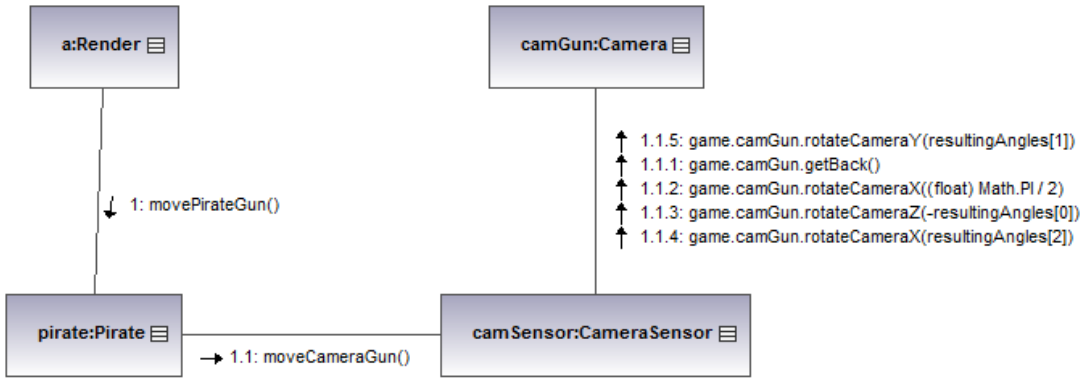

Figura 44: Diagrama de comunicación CU – 04 Mover cámara en modo pistola

DC: CU – 05 Disparar

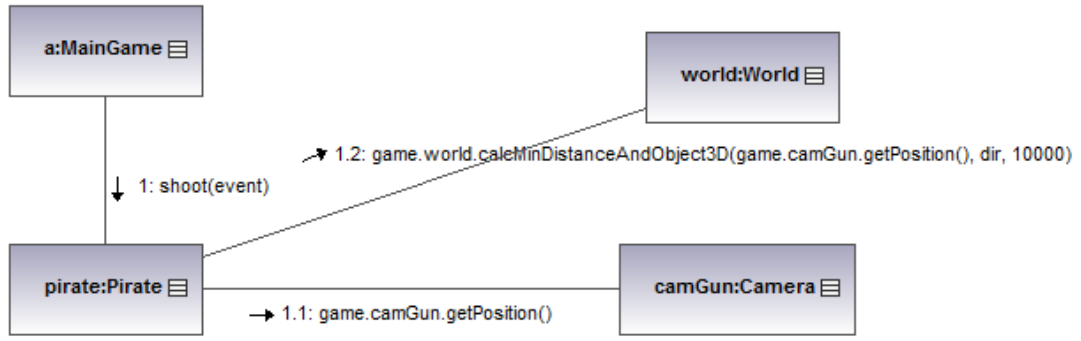

Figura 45: Diagrama de comunicación CU – 05 Disparar

DC: CU – 06 Mover cámara detrás del pirata

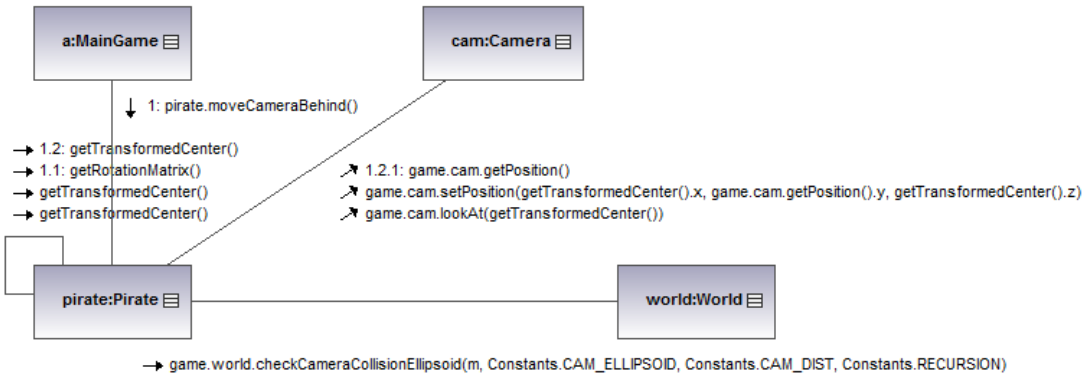

Figura 46: Diagrama de comunicación CU – 06 Mover cámara detrás del pirata

DC: CU – 07 Activar modo pistola

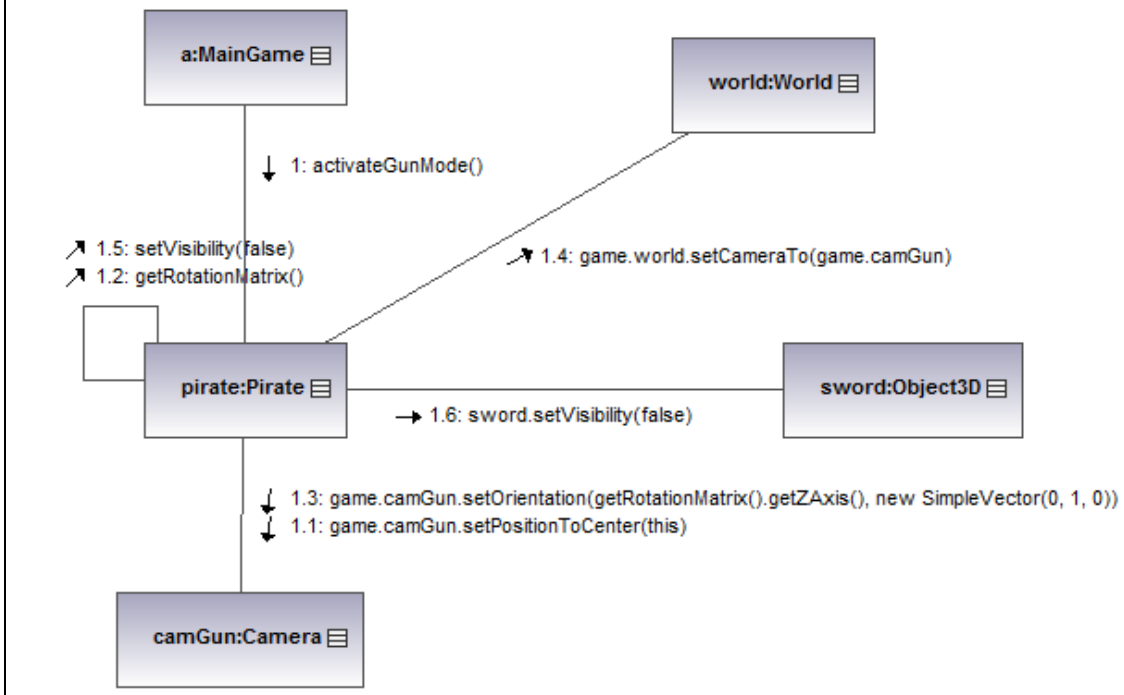

Figura 47: Diagrama de comunicación CU – 07 Activar modo pistola

DC: CU – 08 Desactivar el modo pistola

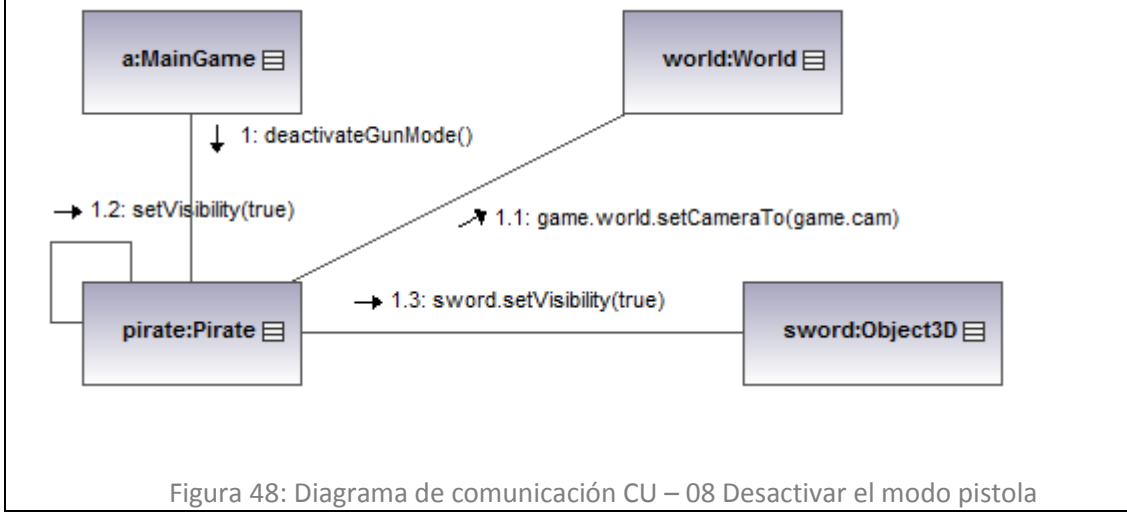

Figura 48: Diagrama de comunicación CU – 08 Desactivar el modo pistola

DC: CU – 09 Mover control en pantalla

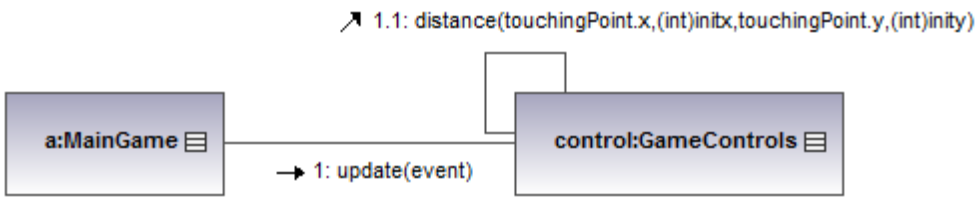

Figura 49: Diagrama de comunicación CU – 09 Mover control en pantalla

DC: CU – 10 Activar puerta

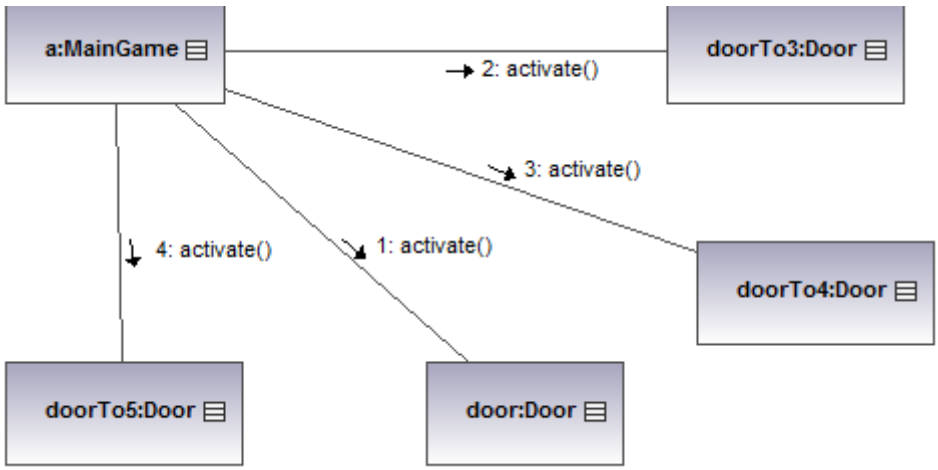

Figura 50: Diagrama de comunicación CU – 10 Activar puerta

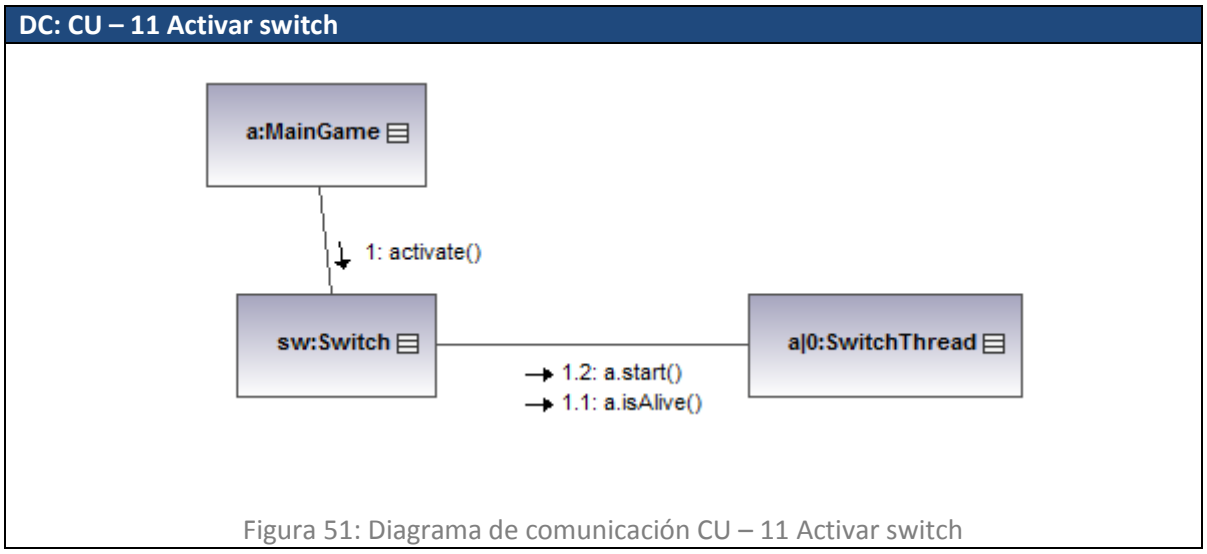

Figura 51: Diagrama de comunicación CU – 11 Activar switch

5.2. Implementación

En esta sección se expondrá la implementación de elementos necesarios para la creación del tipo de juego planeado para el proyecto. Estos son los elementos desarrollados durante la segunda iteración.

5.2.1. Texturas

Las texturas en el software fueron implementadas gracias a las clases *Texture* y *TextureManager* de la librería jPCT – AE, donde ambas extienden de *java.lang.Object*.

La clase *Texture* contiene un objeto bitmap con un alto-ancho potencias de dos y una profundidad de color de 24 BPP con 8 bit alpha[10]. Esta clase posee ocho constructores, donde el utilizado en el proyecto es el siguiente:

- `public Texture (Bitmap imagen, boolean useAlpha)`
    - Este constructor crea una textura usando un *Bitmap*, soportando los mismos formatos soportados por JAVA. Este método además permite activar el canal *alpha* usado para las transparencias. Parámetros: *image*, el *Bitmap* utilizado, *useAlpha*, indica si se habilita o deshabilita el uso del canal alpha de la textura.

La clase *TextureManager* implementa el patrón de diseño singleton[11] utilizado para almacenar y administrar texturas. Cada textura contenida en esta clase debe ser identificada con un nombre único, y cada textura administrada por esta clase debe ser una instancia de *Texture*.

[10] Bits por pixel (BPP) indica cuanta información por pixel existe en una imagen. BPP es la combinación total de todos los bits almacenados en todos los canales de una imagen. Cuando una imagen usa el estándar 24bits RGB, cada pixel almacena 24 BPP, es decir 8bits por cada canal. Cuando una imagen usa 24bit RGB, y además contiene un canal alpha, es decir 32 BPP, cuatro canales y 8 BPP por cada uno.

[11] El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.
Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

Esta clase no posee constructores, ya que se crea por defecto al ocupar esta librería. Dentro de los métodos utilizados, pertenecientes a esta clase tenemos:

- `void addTexture(String name, Texture tex)`
    - Este método añade una textura con el nombre dado al *TextureManager*. Parámetros: *name*, nombre que identifica a la textura, *text*, instancia de *Texture* que contiene el *Bitmap* a almacenar.

En el siguiente trozo de código se muestra el cómo cargar una textura y añadirla al *TextureManager*.

```
Resources res = getResources();
TextureManager tm = TextureManager.getInstance();
      tm.addTexture("piratat",
            new Texture(res.openRawResource(R.raw.piratat)));

      tm.addTexture("espadat",
            new Texture(res.openRawResource(R.raw.espadat)));
```

Estas texturas administradas por el *TextureManager*, son asignadas a los objetos del mundo de la siguiente forma:

```
setTexture("piratat");
```

Donde el método *setTexture(...)*, pertenece a la clase *Object3D* de la librería, y permite asignar la textura al objeto en cuestión.

Creación de texturas

Para la generación de texturas se utilizó el software Adobe Photoshop CS4, cuidando por sobre todo el tamaño (ancho y alto deben ser potencias de dos) y peso de la imagen, ya que imágenes muy pesadas pueden llevar a sobrepasar la capacidad de la maquina virtual de Android (16mb aproximadamente).

En cuanto a la asignación texturas a modelos complejos como por ejemplo el protagonista, se utilizó el software Misfit3D (ver Figura 52), el cual provee una forma sencilla y flexible de administrar la posición y forma en que las texturas de despliegan sobre los modelos (coordenadas).

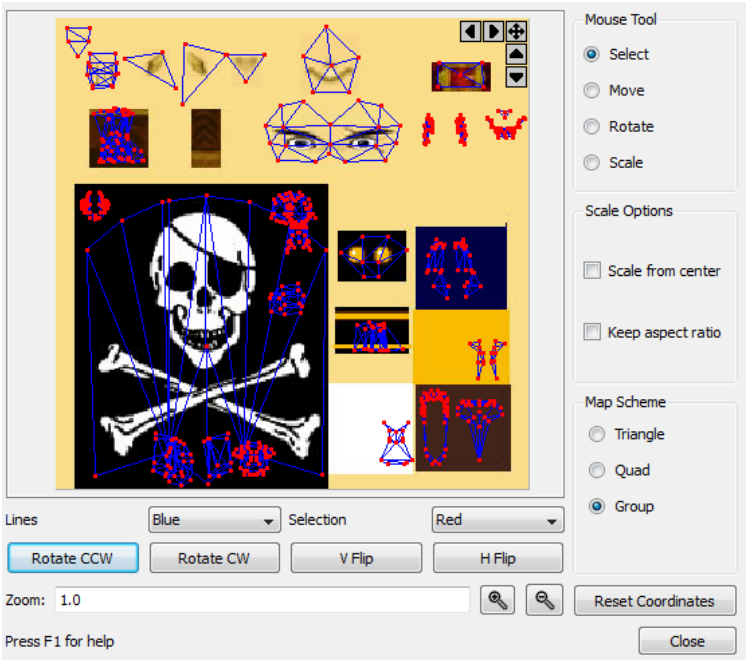

Figura 52: Asignación de coordenadas para la textura perteneciente al Pirata

Al exportar el modelo con las texturas ya asignadas y añadirlos al mundo se conservan las coordenadas de las texturas, e incluso el objeto conoce el nombre de la textura que le pertenece; todo esto gracias a la librería jPCT-AE. Esta ventaja es obvia y para asignar la textura a un objeto sólo basta con llamar al método *setTexture(...)*, anteriormente mencionado.

5.2.2. Iluminación

La iluminación es una parte muy importante en la programación gráfica, ya que está da la sensación realismo, además permite observar el objeto modelado.

En la librería utilizada existe la clase *Light*, la cual extiende de *java.lang.Object* y representa una luz en el mundo. Posee un solo constructor:

- `public Light(World world)`
    - Crea una nueva luz en el mundo. La luz será localizada en el origen (0, 0, 0) y con la intensidad máxima de (255, 255, 255). Parámetros: *world*, que es una instancia de la clase *World*.

Cabe destacar que cuando se crea una instancia de *World*, se crea por defecto una luz ambiental con intensidad de 100 y cuya localización es el origen. Esta luz puede ser modificada mediante el siguiente método de la clase *World*:

- `public void setAmbientLight(int r,int g,int b)`
    - Este método cambia la intensidad para la luz ambiental. Parámetros: *r*, el componente del color que representa al rojo, cuyo valor por defecto es 100; *g*, el componente del color que representa al verde, cuyo valor por defecto es 100; *b*, el componente del color que representa al azul, cuyo valor por defecto es 100.

En el juego se utilizó la luz ambiental que viene dada por defecto y se modificó su intensidad en ciertas ocasiones, por ejemplo al comienzo del juego, utilizando el método *setAmbientLight(...)*, anteriormente mencionado.

5.2.3. Gravedad

La gravedad es fundamental en juego; sin esta no tenemos sensación de peso y por lo tanto poco realismo. Debido a lo anteriormente mencionado es crucial implementar algún mecanismo que simule esta propiedad.

Para implementar la gravedad se utilizó el siguiente método de la clase *Pirate*:

```
public void gravity(){
      if(state != Constants.JUMP){
            SimpleVector gravityVector = checkForCollisionEllipsoid(
            new SimpleVector(0, 1, 0),
            Constants.ELLIPSOID, Constants.RECURSION);
      translate(gravityVector);
      }
}
```

Como puede observarse en el método anterior se utilizó dos contantes. La primera *JUMP*, es debido a que la gravedad no debe actuar cuando el protagonista está saltando, ya que el propio método de saltar realiza la trayectoria que debe seguir el pirata para que el movimiento se vea correcto. La segunda constante *ELLIPSOID* es un *SimpleVector* que representa más bien una esfera en este caso (2, 2, 2) utilizada para encerrar el objeto en cuestión. La última constante *RECURSION* es 5, e indica la cantidad de veces que la colisión del objeto es revisada, entre más alto este valor se obtiene más precisión, pero a un costo de procesamiento muy alto, por lo que se recomienda que este valor esté en el rango de 1 a 5.

En términos generales, este método calcula el vector necesario para que el protagonista choque con algún elemento bajo él, y lo traslada según este vector. Esto da

la sensación de gravedad, ya que el personaje está siendo atraído hacia abajo, simulando el efecto que causa la fuerza de gravedad

5.2.4. Colisiones

Las colisiones son fundamentales en cualquier juego, ya que estas nos permiten interactuar entre objetos, simulando diversas acciones entre ellos, por ejemplo el típico choque en un juego de autos o dar el pie para que ocurra otra acción, como abrir una puerta.

Dentro de la librería utilizada existe un mecanismo muy intuitivo y a la vez eficaz de implementar colisiones; existe una propiedad dada a los *Object3D* que los clasifica según tres categorías:

- `static int COLLISION_CHECK_NONE`

  No se realiza ninguna clase de colisión con este objeto

- `static int COLLISION_CHECK_OTHERS`

  Otros objetos pueden colisionar con este objeto

- `static int COLLISION_CHECK_SELF`

  Este objeto puede colisionar con otros objetos

Así por ejemplo el protagonista es clasificado como un objeto que puede colisionar con otros y el castillo como un objeto que ocasiona que colisione otro.

Para detectar las colisiones se debe emplear un objeto *CollisionListener*, el cual se encarga de recibir el evento en caso de colisión mediante un objeto instancia de *CollisionEvent*. Esta clase *CollisionEvent* genera el evento en caso de que ocurra una colisión.

```
DoorCollision doorCollision = new DoorCollision();
door.setCollisionMode(COLLISION_CHECK_OTHERS);
door.addCollisionListener(doorCollision);
```

Por ejemplo, en el código anterior la puerta se clasifica como un objeto que puede ocasionar que otro colisione con ella. También tenemos el objeto *doorCollision*, que es una instancia de *DoorCollision* que en si extiende de *CollisionListener*; este objeto es asignado a la puerta, y de esta forma se puede detectar las colisiones con este objeto y realizar algún tipo de acción, por ejemplo si el pirata colisiona con la puerta, causar que esta se abra.

5.2.5. Sonido

El sonido en el software fue implementado mediante las clases *AudioManager* y *SoundPool*, proporcionadas por el sistema operativo Android. La primera clase *AudioManager* provee un mecanismo para controlar el volumen y los sonidos. La segunda *SoundPool* es una colección de samples que pueden ser cargados en memoria desde un recurso de la aplicación o desde un archivo en el sistema. *SoundPool* usa el servicio *MediaPlayer* [12] para decodificar el audio en un formato 16 bit PCM mono o estéreo. Esto permite a la aplicación pasar audios comprimidos sin tener que bajar el rendimiento al descomprimir y reproducir al mismo tiempo.

En el software se creó una clase *SoundManager* la cual administra el almacenamiento de los sonidos, mediante un *hashMap*, la reproducción de un sonido en particular mediante una instancia de *AudioManager* y, una instancia de *SoundPool* para crear y reproducir los sonidos que deseemos.

Esta clase *SoundManager*, contiene un método de inicialización que es el siguiente:

```
public void initSounds(Context theContext) {
     mContext = theContext;
     mSoundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);
     mSoundPoolMap = new HashMap<Integer, Integer>();
      mAudioManager =
      (AudioManager)mContext.getSystemService(Context.AUDIO_SERVICE);
}
```

El método anterior se encarga de inicializar los objetos antes mencionados. Además esta clase posee los siguientes métodos cuya finalidad es la administración del sonido en la aplicación:

- `public void addSound(int Index,int SoundID)`
    - Este método se encarga de agregar un sonido al *hashMap* según la clave *Index* y el valor *SoundID*.
- `public void playSound(int index)`
    - Este método se encarga de reproducir el sonido según el *index* dado, obtenido desde el *hashMap* anteriormente descrito.

[12] MediaPlayer es una clase que puede ser usada para controlar la reproducción de archivos de audio, video y streams.

- `public void playLoopedSound(int index)`
    - Este método es similar a *playSound(…)*, sólo que reproduce un sonido indefinidamente.

5.3. Técnicas utilizadas

En esta sección se expondrán las técnicas informáticas aplicadas al proyecto, que fueron integradas para mejorar el rendimiento del juego.

5.3.1. Caché

En informática, una caché es un conjunto de datos duplicados de otros originales, con la propiedad de que los datos originales son costosos de acceder, normalmente en tiempo, respecto a la copia en la caché.

Para el software desarrollado se utilizó una técnica inspirada en este concepto informático. Aplicar esto fue necesario debido a que la carga de procesamiento en ciertos momentos del juego fue demasiada, tornándose injugable.

El modelo completo del mundo (el castillo) del juego fue cortado en escenarios, que corresponden a cada habitación del castillo. Cuando se inicia el juego se carga sólo la primera habitación en un puntero a un *Object3D* llamado *castle*.

Cuando el pirata se acerca a la puerta que accede al siguiente escenario (específicamente si está dentro de un radio de 15 unidades de la puerta) se carga el siguiente escenario en un objeto llamado *castleCache*.

Ahora, si el pirata se aleja de ese radio estando aun en el escenario cargado en *castle* el escenario cargado en *castleCache* se elimina y queda nulo el puntero:

```
game.world.removeObject(castleCache);
castleCache = null;
```

Por otro lado si cruza la puerta y se aleja del radio de la puerta pero ya pisando en el escenario cargado en *castleCache*, se realiza lo siguiente:

- Se elimina el escenario cargado en *castle*:

```
game.world.removeObject(castle);
```

- El puntero *castle* apunta al mismo objeto al que apunta *castleCache*:

```
castle = castleCache;
```

- *castleCache* se hace nulo:

```
            castleCache = null;
```

En otras palabras se realiza una especie de cambio de referencias entre *castle* y *castleCache* para dejar a éste libre de volver a cargar un escenario en caché.

5.3.2. Multithreading

El *Multithreading* es simplemente mantener varios hilos corriendo simultáneamente, con la ventaja que se logra independencia entre los métodos corriendo en los hilos, y esto sirve para que los procesos más pesados no retrasen a los hilos más rápidos.

Esta técnica fue empleada para separar los métodos que animan objetos, que eran llamados en el loop inicial del juego: el método *onDrawFrame* de la clase *Render*. Así, fueron creadas nuevas clases anidadas que extienden de *Thread* en todas las clases que tienen un método de animación y requieren que ese método sea llamado continuamente. Así, cada clase tiene su propio hilo de animación que es inicializado cuando se construye el objeto. Este hilo ejecuta el método en un loop infinito hasta que la aplicación finalice.

Estructura básica de un Thread para animación:

```
public class PirateThread extends Thread{
	//...
	public final static int RUNNING = 1;
	public final static int PAUSED = 2;
	public final static int STOPED = 3;

	public void run() {
		while (state == RUNNING) {
			animate();
			//...
		}
		if(state == STOPED){
			this.finalize();
			//...
		}
	}

}
```

SEXTO CAPÍTULO – RESULTADOS

6.1. Juego

El juego posee nueve habitaciones (ver Figura 53), donde en cada una se presenta diversos obstáculos que deben ser superados para pasar a la siguiente habitación y así finalmente enfrentar al enemigo final.

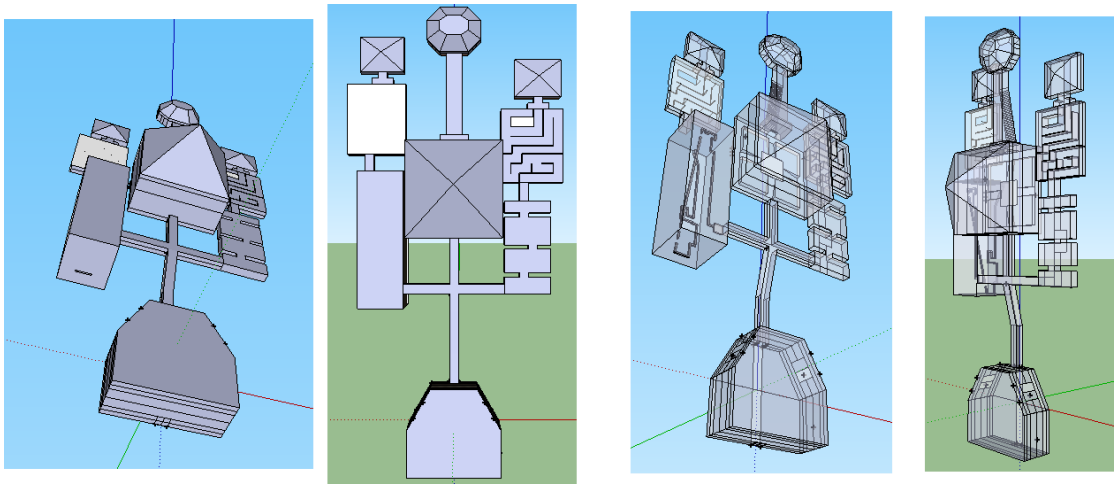

Figura 53: Vistas 3D del plano del castillo

La interfaz del juego aprovecha algunas de las funcionalidades ofrecidas por los dispositivos modernos, en específico pantalla táctil y sensores. Así como puede observarse en la Figura 55, la forma de interactuar con el juego es mediante la pantalla a través del pad virtual y los sensores en el modo pistola.

El juego se desarrollo específicamente para dos modelos de dispositivos Samsung i5500 Galaxy 5 y Samsung Galaxy (ver sección 3.4), aunque pude ser instalado y ejecutado en cualquier dispositivo con sistema operativo Android, pero no se asegura el correcto funcionamiento debido a las características que debe poseer el dispositivo para que se ejecute el juego fluidamente.

Se realizó pruebas en tres dispositivos: Samsung i5500, Samsung Galaxy S y HTC Magic. En los tres dispositivos se instala y ejecuta correctamente la aplicación. En el primer dispositivo, Samsung i5500, el juego corre fluidamente exceptuando cuando se cambia de habitación, ya que se realiza el cambio de los objetos alojados en memoria mediante el caché. En el segundo dispositivo no existe problema alguno de fluidez. En el tercer dispositivo el juego no corre fluidamente, siendo injugable, esto es debido a la poca

memoria con que cuenta el HTC Magic (192mb), muy poca en comparación al Samsung Galaxy S (512mb).

5.3.3. Controles

En esta sección se describe la forma manipular el dispositivo para interactuar con el juego.

Figura 54: Interfaz del juego

1. Barra de vida: indica la cantidad de vida que posee el pirata
2. Pad virtual: se utiliza para mover al pirata dentro del universo virtual
3. Pistola: se utiliza para cambiar a modo pistola, el cual aprovecha el acelerómetro para mover la cámara, simulando estar dentro del mundo virtual.
4. Cámara: se utiliza para posicionar la cámara detrás del pirata.
5. Salto: se utiliza para activar el salto del pirata.
6. Cartel: este espacio de la pantalla se utiliza para mostrar indicadores que activan acciones especificadas por el mensaje escrito en el cartel, por ejemplo abrir una puerta o activar un switch.

- Para atacar a un enemigo se debe tocar su figura en la pantalla.

5.3.4. Enemigos

Hay tres tipos de enemigos "inteligentes", que varían sus movimientos dentro del juego: el fuego, el caballero y el diablo.

- Fuego: pequeña llama voladora que flota por una habitación específica en una trayectoria circular al azar, pero cada vez se acerca más y más rápido al pirata para dañarlo.
- Caballero: el caballero persigue constantemente al pirata y cuando está lo suficientemente cerca, lo ataca con un espadazo. Si golpea al pirata, retrocede unos pasos y vuelve a perseguir a su objetivo. Se requiere dañar con varios espadazos al caballero para matarlo.
- Diablo: su movimiento es similar al caballero, pero cada cierto tiempo este se paraliza convertido en estatua y fuego comienza a caer del techo en la posición donde esté el pirata.

5.3.5. Otros objetos del mundo

Obstáculos

Existen tres tipos de obstáculos en el juego: la guillotina, la sierra y la caja. Cada uno describe un movimiento constante, lo que hace que al pirata se le dificulte el paso sin ser dañado por estos.

Otros objetos

- Botella de ron: En ciertas partes dentro del castillo hay botellas de ron, que el pirata puede tomar y recuperar vida.
- Cruz: Cada vez que se mate a un caballero, una cruz aparecerá en la posición donde se encontraba este. Si el pirata toma esta, se abre la puerta de la habitación donde se encuentra.
- Grim: Estatua que al tocarla se inicia un diálogo.
- Head: Cabeza de un faraón, que al dispararle ocurre una acción determinada. Existen dos cabezas en el juego, al dispararle a una hace subir un puente, y al dispararle a la otra se abre una puerta.

6.2. Distribución de la aplicación

Las aplicaciones para el sistema operativo Android son distribuidas mediante archivos con extensión .apk, el cual es una variante del formato JAR de Java.

Para la distribución del juego se generó un archivo llamado ElTesoroDelMar.apk, el cual se puede obtener mediante la siguiente dirección:

- http://146.83.196.216/~g2chi/ElTesoroDelMar/index.html

En esta dirección se encuentra: información sobre el juego, requerimientos, screenshoots, y una sección de descargas. En esta última, sección de descargas, se puede obtener el juego, descargando el archivo .apk o utilizando el código QR [13].

Para descargar el juego mediante el código QR (ver Figura 55), se debe contar con un móvil con acceso a internet y un programa capaz de interpretar estos códigos. Uno de estos programas, gratuito, es Barcode Scanner el cual lee códigos de barras usando la cámara del teléfono. Para utilizar este programa se debe enfocar el código de barras con la cámara. Si el código es reconocible, dependiendo de qué información este codificada, se puede realizar diversas acciones, desde enviar un SMS hasta redirigir a una página web.

Figura 55: Código QR para descargar el juego

Una vez descargado el juego, en el dispositivo móvil, se debe instalar seleccionando el archivo descargado y autorizando la instalación del mismo.

[13] Un código QR (Quick Response Barcode) es un sistema para almacenar información en una matriz de puntos o un código de barras bidimensional.

6.3. Capturas de pantalla

Las capturas son presentadas en orden secuencial, es decir, siguiendo el avance del juego como lo vería el jugador en una situación normal.

Figura 56: Pantalla de inicio del juego

Figura 57: Presentación de la historia

Figura 58: Pantalla de cargado

Figura 59: Diálogo inicial con la gaviota

Figura 60: Habitación con plataformas flotantes y fuego volador rodeando al pirata

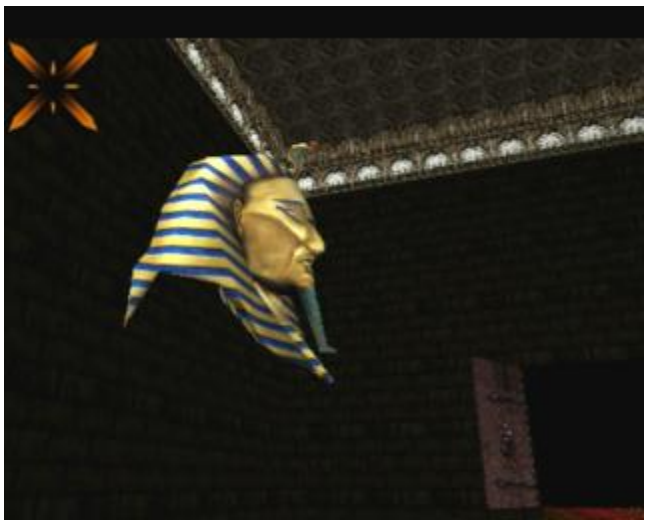

Figura 61: Vista modo pistola, apuntando a una cabeza de faraón

Figura 62: Batalla contra un caballero

Figura 63: Habitación con guillotinas y sierras en movimiento

Figura 64: Habitación con cajas en movimiento

Figura 65: Obtención de una cruz luego de vencer a un caballero

Figura 66: Inicio de diálogo luego de tocar a un Grim

Figura 67: Ingreso a la última habitación del juego

Figura 68: Batalla contra el jefe final, el diablo

Figura 69: Créditos finales, luego de completar el juego

SÉPTIMO CAPÍTULO - CONCLUSIONES GENERALES

7.1. Conclusiones

- A través de este trabajo se demostró en forma práctica la gran capacidad que poseen los dispositivos móviles actuales, capacidad tanto de procesamiento como de interacción con el usuario. También deja de manifiesto las posibilidades que se vislumbran en el ámbito de desarrollo de software para estos dispositivos, tanto en el área de computación gráfica como de software tradicional.
- Mediante este trabajo se aplicó una gran gama de conocimientos adquiridos durante esta carrera, desde ciencias básicas, con cálculos trigonométricos bastante complejos, hasta ciencias de la computación con técnicas de programación avanzadas.
- Además de los conocimientos entregados por la universidad, se debió adquirir conocimientos pertenecientes a otras áreas relacionadas con la elaboración de juegos, como son el diseño gráfico, modelado 3D y composición de música.
- Durante la creación del juego surgieron muchos inconvenientes relacionados con el procesamiento de información, debido a la limitada capacidad de procesamiento de los teléfonos en comparación a los computadores tradicionales. Esto nos llevó a utilizar técnicas de computación gráfica como el Billboard y técnicas de informática como el uso de un caché y el multithreading.
- En el afán de demostrar las capacidades de los teléfonos actuales para interactuar con el usuario, se diseñó una interfaz de juego similar a la de los hardware destinados exclusivamente a los juegos, marcando una diferencia con los juegos para dispositivos móviles actuales en nivel de dinamismo y espectro de opciones.
- Fue difícil comenzar con el desarrollo de este proyecto, ya que el campo de programación de juegos 3D para dispositivos móviles está aun en sus etapas iniciales, por lo que encontrar frameworks destinados a ello y documentación fue prácticamente imposible, teniendo que finalmente optar por utilizar una librería gratuita encontrada en internet que facilitó en gran medida la implementación de elementos fundamentales para un juego de las características requeridas.
- Para finalizar, el objetivo general planteado en el capítulo segundo: diseñar e implementar un juego estilo plataforma en un entorno 3D para dispositivos móviles con sistema operativo Android, fue cumplido en su totalidad, logrando lo expuesto a través de este informe. En cuanto a los específicos, también todos fueron completados y llevados más allá, implementando aun más funcionalidades al juego.

BIBLIOGRAFÍA

Bibliografía

- GRAIG LARMAN, 2002. UML y Patrones, Introducción al análisis y diseño orientado a objetos, Prentice Hall.
- ROGER PRESSMAN, 2002. Ingeniería del Software, un enfoque práctico, Mc Graw Hill.
- VLADIMIR SILVA, 2009. Pro Android Games . Appres.
- SAYED HASHIMI, SATYA KOMATINENI & DAVE MACLEAN, 2010, Pro Android Games 2, Appres.
- ED BRUNETTE, 2009. Hello Android Introduction Google's mobile Development Platform, Third Edition, Pragmatic.

Linkografía

- ANDROID, Wikipedia <http://en.wikipedia.org/wiki/Android_(operating_system)> [consulta 10 de octubre de 2010]
- SISTEMA OPERATIVO, Wikipedia <http://es.wikipedia.org/wiki/Sistema_operativo> [consulta 10 de octubre de 2010]
- VIDEO JUEGO, Wikipedia <http://es.wikipedia.org/wiki/Videojuego> [consulta 10 de octubre de 2010]
- ANDROID HITS TOP SPOT IN U.S. SMARTPHONE MARKET, CNET.COM, <http://news. cnet.com/8301-1035_3-20012627-94.html> [consulta 10 de octubre de 2010]
- WHAT IS ANDROID?, android developers, <http://developer.android.com/guide/basics/ what-is-android.html> [consulta 10 de octubre de 2010]
- OPENGL?, android developers, <http://developer.android.com/topics/graphics/opengl. html> [consulta 10 de octubre de 2010]

ANEXOS

Anexo 1: Ejemplo brazo robot con OpenGL ES

Esta sección tiene como propósito exponer un simple ejemplo de cómo programar con OpenGL ES para dispositivos móviles con sistema operativo Android.

Este ejemplo corresponde a la primera prueba que se realizó previo al desarrollo del proyecto, cuyo fin es realizar una primera aproximación a la programación gráfica para móviles.

El ejemplo en cuestión trata sobre una serie de primitivas, rectángulos, circunferencias y triángulos, que juntos simulan un brazo robot. Además este puede ser rotado en cualquiera de sus articulaciones, tocando la pantalla en una determinada posición.

Esta aplicación consta de dos clases: *Test1* y *Renderer1*. La primera clase es un *Activity* de Android, se encarga de asignar la vista mediante el método **`setContentView(View)`**, como se muestra en el siguiente código:

Clase *Test1*:

```
public class Test1 extends Activity {

    private Renderer1 render;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        render = new Renderer1(this);
        setContentView(render);
    }
}
```

La segunda clase *Renderer1*, es más compleja, ya que aquí se realiza el renderizado de la escena 3D. Para ello, esta clase extiende de la clase propia de OpenGL *GLSurfaceView* e implementa la interfaz *Renderer*, que contiene 3 métodos: *onSurfaceCreated*, *onSurfaceChanged* y *onDrawFrame*.

onSurfaceCreated se ejecuta cuando se crea la superficie para renderizar. *onSurfaceChanged* se ejecuta cuando se modifica esta superficie (por ejemplo cuando se cambia la orientación del teléfono). *onDrawFrame* se ejecuta para renderizar la escena actual en la superficie. *onDrawFrame* se llama continuamente mientras la superficie esté

creada, lo que crea un loop que se puede utilizar para ejecutar todos los métodos que requieran llamadas continuas y hace posible el movimiento de los elementos del mundo.

Primero, este es el constructor de la clase *Renderer1*:

```
public Renderer1(Context context) {
	super(context);
	//Setear esto como renderer
	this.setRenderer(this);
	//Requerir el focus, ya que sin esto el touch no funciona
	this.requestFocus();
	this.setFocusableInTouchMode(true);
}
```

Método *onSurfaceCreated*:

```
public void onSurfaceCreated(GL10 gl, EGLConfig arg1) {
	gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
	initSquare();
	initOval();
	initTriangle();
}
```

initSquare(), *initOval()* e *initTriangle()* son métodos propios que crean primitivas en 2D, tomando arreglos con los vértices que formarán la figura deseada:

```
private float vertices3[] = new float[360*2];

private float vertices[] = {
		-1.0f, -1.0f, 0.0f,
		1.0f, -1.0f, 0.0f,
		-1.0f, 1.0f, 0.0f,
		1.0f, 1.0f, 0.0f};

private float vertices2[] = {
		0.0f, 1.0f, 0.0f,
		-1.0f, -1.0f, 0.0f,
		1.0f, -1.0f, 0.0f};

// buffers para mantener los vértices
private FloatBuffer vertexBuffer;
private FloatBuffer vertexBuffer2;
private FloatBuffer vertexBuffer3;

private void initSquare() {
	ByteBuffer byteBuf = ByteBuffer.allocateDirect(vertices.length*4);
	byteBuf.order(ByteOrder.nativeOrder());
	vertexBuffer = byteBuf.asFloatBuffer();
	vertexBuffer.put(vertices);
	vertexBuffer.position(0);
}

private void initTriangle() {
```

```
        ByteBuffer byteBuf = ByteBuffer.allocateDirect(vertices.length*4);
        byteBuf.order(ByteOrder.nativeOrder());
        vertexBuffer2 = byteBuf.asFloatBuffer();
        vertexBuffer2.put(vertices2);
        vertexBuffer2.position(0);
}

private void initOval() {
        int count=0;
        for (float i = 0; i < 360.0f; i+=(360.0f/360)){
              vertices3[count++] = (float)Math.cos (Math.toRadians
              ((double)i*1.0f));
              vertices3[count++] = (float) Math.sin (Math.toRadians
              ((double)i*1.0f));
               }
        ByteBuffer byteBuf = ByteBuffer.allocateDirect(vertices3.length*4);
        byteBuf.order(ByteOrder.nativeOrder());
        vertexBuffer3 = byteBuf.asFloatBuffer();
        vertexBuffer3.put(vertices3);
        vertexBuffer3.position(0);
}
```

El método *onSurfaceChanged* está vacío, ya que este prototipo no lo necesita:

```
public void onSurfaceChanged(GL10 gl, int w, int h) {}
```

El método *onDrawFrame* se encarga de realizar el constante renderizado de la escena. En este caso, dependiendo del valor actual de los ángulos que contienen la rotación de cada articulación, lo que se muestra en pantalla simulará el movimiento de un brazo robótico.

Para actualizar el valor del ángulo de cada articulación se debe tocar en partes específicas de la pantalla; estos eventos hacen que el ángulo en cuestión aumente o disminuya, provocando la rotación del elemento requerido. Este comportamiento es capturado mediante el método *onTouchEvent*.

A continuación se presenta una parte del método *onDrawFrame*:

```
public void onDrawFrame(GL10 gl) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glLoadIdentity();
        // define el color de fondo
        gl.glClearColor(255.0f, 255.0f, 255.0f, 1.0f);
        gl.glViewport(0, 0, getWidth(), getHeight());
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT);
        gl.glColor4f(0.0f, 0.0f, 100.0f, 1);
        gl.glPushMatrix();
              gl.glTranslatef(0.0f, -0.9f, 0.0f);
              gl.glRotatef(angle, 0.0f, 0.0f, 1.0f);
```

```
            gl.glPushMatrix();
                  gl.glScalef(0.15f, 0.15f, 0.0f);
                  gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer3);
                  gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0,
vertices3.length/3);//vetices3 contiene los vértices de un circulo
            gl.glPushMatrix();
                  gl.glTranslatef(0.0f, 0.4f, 0.0f);
                  gl.glScalef(0.1f, 0.4f, 0.0f);
                  gl.glVertexPointer(3, GL10.GL_FLOAT, 0, vertexBuffer);
                  gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0,
vertices.length/3);// //vetices contiene los vértices de un rectángulo
            gl.glPopMatrix();
//continua…
```

En el trozo de código anterior se expone el cómo se dibuja en cada frame una parte del brazo robótico, en es específico el hombro y el antebrazo. En el ejemplo se puede notar el uso de las sentencias `gl.glPushMatrix()` y `gl.glPopMatrix()`; estas sentencias son utilizadas para simular un comportamiento jerárquico, es decir si el hombro rota el antebrazo debe seguir su movimiento. Este comportamiento no es más que transformaciones aplicadas en segmentos específicos definidos por las sentencias en cuestión, de manera que las tranformaciones de un elemento afecten sólo a las deseadas. Volviendo al ejemplo anterior, la rotación del hombro debe afectar al antebrazo.

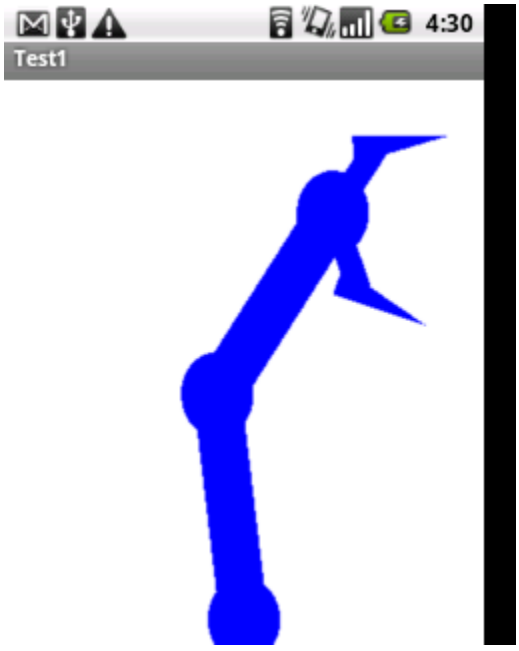

Figura 70: Captura de pantalla brazo robot

Anexo 2: Ejemplo librería jPCT-AE

Este ejemplo trata sobre una sencilla aplicación donde se muestra cómo utilizar la librería jPCT-AE, utilizada en la implementación del proyecto. La idea de este ejemplo es importar un modelo 3ds e incluirlo en la escena de la aplicación 3D.

Para este ejemplo fueron creadas 2 clases, construidas de la misma forma que en el ejemplo del anexo 1.

La primera clase, *JpctDemo*, es un *Activity* exactamente igual al del ejemplo del anexo 1.

La segunda clase, *MyRenderer*, cumple la misma función que *Renderer1* del anexo 1, es una clase extendida de *GLSurfaceView* e implementada la interfaz *Renderer*.

En *MyRenderer*, en el método *onSurfaceCreated* crearemos todos los elementos del mundo, empezando con el mundo mismo:

```
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
      world = new World();
      …
}
```

Después crearemos una instancia de *TextureManager*, que es la clase que se encarga de almacenar las texturas utilizadas en el mundo:

```
TextureManager tm = TextureManager.getInstance();
```

Ya que el importador de modelos de jPCT-AE no importa las texturas asociadas al modelo, éstas deben ser importadas y asociadas al modelo en el código.

Crearemos un objeto *Texture* que tendrá la textura que utilizará nuestro modelo 3ds:

```
Texture rocky = new Texture(getResources().openRawResource(R.raw.rocky));
```

A continuación incluimos la textura en el *TextureManager*, asignándole un nombre. El nombre debe ser estrictamente el mismo que la textura asociada al modelo.

```
tm.addTexture("rock", rocky);
```

Luego se importa el modelo 3ds en un arreglo de *Object3D*:

```
Object3D[] rockArray = Loader.load3DS(res.openRawResource(R.raw.rock),
15f);
```

Object3D es el objeto que representa a un objeto 3D, en este caso, el modelo importado. El método *load3DS* retorna un arreglo de *Object3D*, con todos los componentes de ese modelo por separado. Para unir todo el modelo se utiliza:

```
Object3D rock = Object3D.mergeAll(rockArray);
```

Ya que la textura de *rock* está en el *TextureManager*, es asociada a este.

Lo siguiente es trasladar el objeto hacia adelante en el eje Z, para poder ser visto:

```
rock.translate(0, 0, -90);
```

Ahora obtendremos la cámara actual que está utilizando el mundo, la subiremos un poco en el eje Y y le diremos que mire directamente al objeto *rock*:

```
Camera cam = world.getCamera();
cam.moveCamera(Camera.CAMERA_MOVEUP, 100);
cam.lookAt(rock.getTransformedCenter());
```

Finalmente con estas líneas incluimos el objeto creado en el mundo y lo construimos para poder ser visto:

```
world.addObject(rock);
world.buildAllObjects();
```

Ahora para realizar el renderizado en el método *onDrawFrame*, se utilizan las siguientes líneas:

```
public void onDrawFrame(GL10 gl) {
      fb.clear();
      world.compileAllObjects(fb);
      world.renderScene(fb);
      world.draw(fb);
      fb.display();
}
```

Con *fb* siendo un objeto *FrameBuffer*, que es un buffer que contiene toda la información del frame a ser dibujado en la escena actual.

Figura 71: Captura de pantalla ejemplo librería jPCT-AE