Universidad Nacional Autónoma de México

Facultad de Ingeniería

Diseño e Implementación de Juegos Multimodales: Estrategia y Acción

T E S I S

que para obtener el título de:

Ingeniero en Computación

P R E S E N T A

Jesús Mauricio Andrade Guzmán

Juan Pablo Reyes Altamirano

Director: Ing. Rodrigo Guillermo Tintor Pérez

Ciudad Universitaria, México D.F., 21 de mayo de 2009

Dedicado a la memoria de mi abuela, Alicia Benítez de Altamirano cuyo amor por la vida solo lo igualaba su inmensa curiosidad por ella. Fue su deseo que las generaciones futuras tuvieran la misma oportunidad de conocer a fondo su mundo y enamorarse de él tanto como lo hizo ella. Con esta tesis espero iniciar una jornada que apoya ese deseo, entregando a los niños, de todas las edades, un regalo que muestra una pizca de la verdad mas majestuosa de la creación: lo especial que son cada uno de nosotros.

-Juan Pablo Reyes Altamirano

Quisiera dedicar este trabajo de tesis a todos aquellas personas que de alguna manera me apoyaron en este camino, especialmente a Lesly que ha estado a mi lado desde el comienzo de mi educación en la Facultad de Ingeniería y desde el inicio de una vida que hemos construido juntos.

A mis amigos Rocío, Arturo, Angie, Beatriz, Octavio, Mónica, Marco y afortunadamente a muchos otros con los que puedo contar que siempre han estado ahí para compartir cada experiencia de nuestra vida con una sonrisa o una buena discusión.

A mi Abuelita María† que siempre estuvo ahí para demostrarme su amor y cariño.

A mi Madre Hortensia† que su recuerdo ha sido un apoyo siempre presente.

A mi Padre Armando y mi hermano Adrian que me apoyan siempre que lo necesito.

A toda mi familia que me acepta y respeta por la persona en que me he convertido gracias a ellos.

-Mauricio Andrade

Agradecimientos:

En mi camino he enfrentado a muchas dificultades que no se pudieron haber resueltos con solo mi necedad. Reconozco que debo un especial agradecimiento a quienes me han brindado un mundo de ayuda y, de vez en cuando, un mundo de inspiración.
A mis padres, Octavio Reyes y Alicia Altamirano - por todos sus sacrificios personales y por seguir creyendo ciegamente en mi.
A mi hermano, Eric - por compartir los mejores y los peores de los tiempos y por ayudarme a levantarme cuando no quería.
A mis tíos, tías, primas y primos - por haberme ayudado a descubrir quien soy, a donde voy y de donde vengo.
A mis abuelos, Natal y Alicia† - por presentarme a un universo de curiosidades y deleites que no tiene fin.
A mis abuelos, Otilio† y Mikaela† - por mostrarme lo precioso que es vivir y sentir.
A mis amigos del Centro Universitario México - por brindarme una segunda oportunidad en mi vida joven.
A mi familia del Departamento de Realidad Virtual y de la DGSCA - por dejarme perder entre mis sueños y dejarme brillar.
A mi familia del Programa de Tecnología en Computación - por educarme en el arte de la computación y la amistad.
A todos mis amigos queridos de la facultad de Ingeniería - por hacer de mis 8 años en la carrera los años mas inolvidables de mi juventud.
Al Ing. Santiago Igor Valiente Gomez y al Ing. Rodrigo Tintor - por haberme aguantado todas mis locuras y por todo el apoyo que brindaron.
A mi compañero torturado de la tesis, Mauricio “Chuchin” Andrade - por creer en mi, haberme aguantado y haber compartido la aventura mas grande de mi juventud.
A Rocio - por haber sufrido tantos horrores de edición y todavía poder salir adelante con una sonrisa.
A Linda Rey - por estar ahí cuando mas te necesitaba.
A todos los que me falta aun por mencionar - les agradezco desde el fondo de mi corazón.
-Juan Pablo Reyes Altamirano

Agradecimientos adicionales:
Al Ingeniero Igor Valiente por haberme dado la oportunidad de realizar este trabajo de tesis
A Lesly por su valiosa colaboración en la parte creativa de la idea del video-juego
A Rocío por la labor editorial que aportó para este trabajo escrito
A Beatriz por el modelado de personajes y diseño de la portada
A Israel que me ayudó a aterrizar muchas de las ideas plasmadas en este trabajo
-Mauricio Andrade

Índice general

Prefacio

Desde tiempos remotos, el hombre, dentro de su búsqueda para mejorar su estilo de vida, también ha buscado maneras de mejorar y hacer mas complejas sus actividades recreativas. Casi siempre los grandes logros en esta área son accidentados pero solo debido a que la diversión, a menudo, es una curiosidad insaciable llevada al extremo. De esta forma no se nos hace tan difícil de creer que un simio en la jungla, con su intensa curiosidad por la flora a su alrededor, y un matemático, en frente de su computadora, sean muy parecidos...ambos se divierten curioseando con su ambiente.

Ahora se puede decir que el mono eventualmente evolucionó al punto en el cual podía representar ideas por medio de la pigmentación. Las imágenes y posteriormente la escritura fueron resultados de primero haber aplastado, con las manos, alguna planta, fruta o igual de estar manchado de sangre fresca y haber decidido limpiarse contra alguna superficie - la combinación de una superficie y una sustancia pigmentosa consolida una parte fundamental de las artes plásticas. Sabemos que de igual manera el descubrimiento de la cerámica asentó las bases para lo que posteriormente seria el modelado, la otra mitad fundamental de las artes plásticas. En pocas palabras, le tenemos que dar gracias a este mono, quien por medio de su diversión, inicio una de las tareas mas pesadas que se haya hecho el hombre - el intento por recrear la realidad, sea como se quiera percibir, por medio de la imaginación y del ingenio.

Miles de años después nuestro matemático, con un profundo dominio de las cantidades y sus abstracciones, empieza a divertirse con la representación gráfica de algunas formulas geométricas en su computadora. Se da cuenta de que la combinación de estas formas con otras no tanto matemáticos, sumado a la gran cantidad de memoria que posee su computadora, lo hace capaz de hacer imágenes que son muy atractivos a la vista. Posteriormente esto da lugar a lo que conocemos, hoy, como el arte de modelar y "renderizar" en 3D.

En ambos casos, siempre hubo recreaciones que dan lugar a mas conocimientos que mejoraron la calidad de vida del ser humano. ¿Qué pasa cuando ya hemos llevado este conocimiento a su máxima expresión? Nos volvemos a sentar como si nada nos hubiera pasado? Nos aburrimos? No. La tendencia humana nos dice que es hora de divertirnos otra vez. El mono crea los primeros juguetes a partir de sus conocimientos de las artes plásticas y el matemático le dio una nueva dimensión (literalmente) a lo que ya se conocía como videojuegos.

Este ciclo de diversión-invención-desarrollo-diversión abarca un principio

primordial del ingenio humano - la necesidad es la madre de la invención. La recreación es de las necesidades mas importantes pero de las menos contempladas como tal.

Es gracias a este fenómeno tan peculiar de la sociedad, por la cual la industria de los juguetes y de los videojuegos ha hecho a muchos individuos millonarios en poco tiempo. Hacer videojuegos no se considera una profesión tan importante en nuestra sociedad actual, pero nuestros hijos y los niños grandes no están de acuerdo con esta idea.

Fuera de lo práctico y fuera de lo científico, la tarea mas ardua sobre la tierra es hacer cosas que sean un deleite para los demás... pero no tiene que ser, si nosotros también nos divertimos haciéndolas.

Bien entonces, vamos a abarcar minuciosamente esta tarea, lo que implica y lo que deja.

Capítulo 1

Introducción

1.1. Introducción

La industria de los videojuegos existe desde hace relativamente poco tiempo, siendo que no fue sino hasta la década de los ochenta cuando se consolidó como una industria económicamente activa. Dentro del entretenimiento, los videojuegos han superado en ganancias incluso a la industria del cine.

En México el mercado de videojuegos se limita principalmente a la venta de software importado de diversos países. La creación de videojuegos mexicanos existe, pero no puede ser comparada con el mercado mundial, siendo nuestro mercado un actor pasivo de esta actividad económica. La industria de los videojuegos seguirá creciendo en los próximos años y es muy importante tomarlo en cuenta para poder participar activamente y con productos nacionales de calidad.

Este trabajo de tesis pretende documentar el desarrollo de un videojuego partiendo de herramientas de libre acceso a todo público. El desarrollo de un software de este tipo tiene un gran valor académico, ya que requiere de conocimientos especializados en disciplinas tales como la creación de ambientes virtuales, inteligencia artificial, programación orientada a objetos, modelado de objetos en tres dimensiones, entre otras.

1.2. Resumen de Capítulos

En el capítulo *Estructura general de un videojuego* se presentarán algunas de las disciplinas que sirven de base para el desarrollo de videojuegos. Se examinarán principalmente la programación orientada a objetos, cómputo gráfico e inteligencia artificial. (Capítulo. 2)

En el capítulo *Recursos Utilizados en este Proyecto* se describirán las técnicas utilizadas para el desarrollo de videojuegos, la integración de librerías externas de simulación de física, interfaces gráficas de usuario e inteligencia artificial. Se profundizará en el uso de OGRE 3D y las librerías específicas utilizadas en este proyecto. (Capítulo. 3)

En el capítulo *Sistema desarrollado* se mostrará el sistema desarrollado para este proyecto, describiendo el análisis de los elementos que conforman el videojuego y el motor desarrollados. (Capítulo. 4)

1.3. Planteamiento del Problema

Las metodologías usadas en la creación de videojuegos son tan diversas que no es posible generalizar sobre esta tarea. Prácticamente cada autor o empresa dedicada a esta actividad ha desarrollado una metodología propia.

El trabajo que se requiere para desarrollar un videojuego depende de la complejidad del mismo y de sus reglas. En los principios de esta industria los ambientes que se manejaban eran relativamente simples, sin embargo, el avance en la tecnología, la competencia derivada de la gran aceptación en el público y el ingenio de los programadores, ha hecho posible juegos con reglas y ambientes tan complejos como realidades alternativas.

El hecho de que un juego tenga cierta complejidad no lo hace necesariamente bueno para el jugador, esto depende de diversos factores como el modo de juego, el nivel de dificultad y los ambientes en los que se desarrolla. Un juego de plataformas 2D [1] no necesita de un motor de física o inteligencia artificial, su modo de juego y la dinámica de los elementos que conforman el ambiente es lo que hace a este tipo de juegos que sean tan divertidos.

Por todo lo anterior, el propósito de esta tesis, es documentar la creación de un videojuego, partiendo de herramientas que requieren que el desarrollo se realice desde niveles bajos, es decir, solo lo indispensable para la creación de un ambiente gráfico interactivo con el usuario.

El videojuego que se pretende realizar permitirá al jugador ver el juego desde dos puntos de vista, uno de tipo acción detrás del personaje principal y otro de estrategia, donde se podrán seleccionar varios personajes para ser manipulados.

1.4. Justificación

Programadores de todo el mundo se han dedicado por mucho tiempo al desarrollo de motores de videojuegos, cada uno con el objetivo de realizar juegos para un género específico. En ocasiones estos motores son reutilizados por otros equipos de desarrollo para crear juegos con características similares. Existen alternativas comerciales y de código abierto de estos motores de juegos, que se distribuyen con el fin de promover su uso o de contribuir con el proyecto.

[1] Los juegos de plataforma se pueden identificar con el estilo de videojuego que nos presenta Mario Bros. (el plomero que salva a la princesa en desgracia.. ¨), que no ha cambiando en más de 20 años

Básicamente existen dos tipos de motores de juegos: enfocados a un tipo específico de juego y de propósito general.

- ***Motor de juegos de propósito específico*** Los enfocados a un género específico son fáciles de usar y permiten crear juegos con tramas complejas en tiempos relativamente cortos; los más usados son aquellos que permiten crear juegos FPS[2].

  Para implementar un motor de este tipo, basta con implementar el API [3] del fabricante y crear la lógica (reglas) del juego con lenguajes basados en scripts interpretados.

- ***Motor de juegos de propósito general*** Los motores de propósito general requieren mayor habilidad programando y pueden ser usados para crear juegos de cualquier género, incluso juegos tanto en 2D como 3D.

Este proyecto de tesis propone un motor de juegos basado en uno de propósito general simplificado, que integre características de los géneros: estrategia[4] y acción[5].

La aportación de este trabajo es combinar dos géneros de juegos en el que el jugador pueda elegir cual usar, en un ambiente donde el que los personajes deben cumplir un objetivo y el jugador puede ayudar a su equipo a obtener la victoria.

1.5. Objetivo General

Desarrollar un videojuego que consiste en el diseño del motor que controla las funciones principales del videojuego y la creación del ambiente virtual que contienen las escenas del videojuego.

[2]Los juegos de acción en primera persona, también llamados First Person Shooters (FPS) en inglés, donde el jugador tiene la impresión de estar dentro del escenario virtual (que generalmente se proyecta en tres dimensiones) y tiene el mismo punto de vista que tendría de estar presente en ese escenario.

[3]Application programming interface (Interfaz de programación de aplicaciones)

[4]Un videojuego del género estrategia militar en tiempo real contiene elementos de inteligencia y planeación, para este proyecto se usará el término estrategia para definir un juego que se desarrolla dentro de un tablero con dos equipos rivales, donde se lleva a cabo la simulación de una batalla, en la cual, el jugador ordena acciones a los miembros de un equipo: ataque, defensa y desplazamiento; omitiendo recursos, rangos, construcciones y otros elementos también relacionados con el género

[5]Para este trabajo, un juego de acción es aquel en que se puede ver a un personaje de espaldas y el jugador es quien controla todos sus movimientos

1.6. Alcances y Limitaciones

El producto final de este proyecto será el desarrollo teórico de un motor, y el esbozo de un juego creado a partir de este motor prototipo. Se definirán reglas básicas y un ambiente que permita visualizar los resultados.

El jugador será capaz de interactuar con el ambiente virtual, sin embargo, el control se limitará al movimiento de los personajes dejando para una extensión posterior el comando de órdenes y batallas entre equipos

1.7. Resultados Esperados

Al examinar las técnicas y herramientas para desarrollar videojuegos se espera documentar el proceso que se siguió con el fin de sentar un antecedente sobre el desarrollo de videojuegos.

El prototipo de videojuego que se espera obtener mostrará las capacidades de manejo de gráficos, simulación de un sistema físico en el ambiente virtual, control de personajes, carga de terrenos y personajes virtuales en el ambiente.

Capítulo 2

Estructura General de un Videojuego

2.1. Definición Global

Un videojuego es un producto de software cuyo objetivo es el entretenimiento, basado en la interacción con una o varias personas, que se conocen como jugadores. Esta interacción se desarrolla en ambientes creados por el propio juego siguiendo reglas determinadas para alcanzar el objetivo deseado.

2.2. Primeros Videojuegos

En 1962 Steve Russell y algunos socios programaron lo que sería reconocido como uno de los primeros videojuegos, *Spacewar!*[5]. El programa implementaba simples funciones sinusoidales para simular los efectos de un par de naves disparándose a sí mismas mientras giraban alrededor de un hoyo negro (figura 2.1). Éste había sido precedido por el ancestro directo de *Pong*, *Tennis for Two*[20], de William A. Higinbotham, pero cuya popularidad no se comparó con el de *Spacewar!* hasta 1972, cuando Nolan Bushnell, considerado el "padre de los videojuegos", fundó Atari Interactive y formalmente presentó *Pong* al mundo. *Spacewar!* nació en un equipo DEC PDP-1, el ancestro directo del PDP-11 (donde nació UNIX y posteriormente el lenguaje C). Este mismo juego sería anecdótico para Bill Gates al ser entrevistado en la víspera del lanzamiento de DirectX (haciendo referencia a sus experiencias programando el emulador del Altair 8800 en un PDP-10)[11].

Figura 2.1: SpaceWar! en la pantalla del PDP-1 (Bib. [5])

Partiendo de los ejemplos anteriores podemos identificar elementos usados para crear estos videojuegos:

1. La lógica (hay que disparar y eliminar a las naves enemigas para ganar)
2. Simulación de física (la velocidad de los misiles y de las naves, la atracción gravitacional del hoyo negro)
3. La generación de gráficos (algoritmos al estilo de Ivan Sutherland[1])
4. Integración de sonido.[2]

A grandes rasgos esta misma estructura no ha cambiado en los videojuegos actuales. Hoy en día se agrega a esta lista elementos como Inteligencia Artificial (para complementar a los algoritmos de lógica), manejadores y cargadores de escenas.

De la misma manera que los juegos físicos contribuyeron al conocimiento en general, los videojuegos también contribuyen a la ciencia en computación. Es sabido que los videojuegos han contribuido al avance de la inteligencia artificial y muchos otros campos pioneros de la ciencia en computación, también se reconoce que el desarrollo de uno de los sistema operativo mas populares en la computación, UNIX, fue debido a un videojuego.

A finales de los 60's, AT&T dejo de participar en el programa MULTICS (Bib. [1]), un sistema operativo que reemplazaría al sistema GECOS del GE-645. Ken Thompson, uno de los contribuyentes del proyecto, programó un videojuego llamado *Space Travel* para MULTICS. Al darse cuenta que era extremadamente caro (en horas CPU), lo portó a FORTRAN en el sistema GECOS y finalmente a ensamblador de PDP-7. La experiencia obtenida de portar dicho juego al PDP-7 le sirvió a Ken Thompson para desarrollar UNICS, el predecesor en ensamblador PDP-7 de UNIX. Aunque no se concibió como tal, algunos programadores UNIX consideran a *Space Travel* como la primera aplicación UNIX. (Bib. [4])

2.2.1. Estructura de un Videojuego

Partiendo de la estructura mencionada en el punto 2.1, para este proyecto se utilizó una librería para render en tiempo real, librerías de simulación de

[1] Ivan Sutherland es considerado por muchos como el creador de los gráficos por computadora - http://es.wikipedia.org/wiki/Ivan_Sutherland

[2] El cual hubiera sido una serie de ondas cuadradas debido principalmente a que en esa época no existían las tarjetas de sonido como las conocemos actualmente.

física y de interfaces gráficas de usuario (*GUI*). Estos elementos conforman un motor de juegos, actualmente existen opciones como el motor Delta3D[3] o el motor de juegos de Quake o Unreal, ambos de videojuegos comerciales.

Se eligió solo usar un motor de render en tiempo real ya que esto permite explorar diferentes opciones para la implementación de los demás elementos. Al usar librerías separadas se obtienen ciertas ventajas básicas de implementación, esto es, libertad en el desarrollo del tipo de juego.

Un videojuego es un equilibrio de muchas disciplinas. De estas disciplinas la inteligencia artificial es un punto importante, pues en conjunto con la simulación de física, son los elementos que el jugador percibe con mayor frecuencia ya que son los aspectos que atribuyen la mayor cantidad de realismo en el juego (después del visual). Esta percepción de realismo en los videojuegos se conoce como grado de inmersión, el cual, si es parte del objetivo, tiene prioridad sobre otros elementos del desarrollo, como sucede en videojuegos de simulación de combates militares que reproducen con gran fidelidad elementos del ambiente.

[3]Motor de código abierto de videojuegos y simulación - http://www.delta3d.org

2.3. Disciplinas Relacionadas

Crear un videojuego partiendo de librerías de desarrollo separadas requiere de un análisis basado en ingeniería de programación y algunas técnicas de programación de bajo nivel, esto debido a que no existe una metodología general para el desarrollo de videojuegos por la complejidad y variedad de este tipo de software.

A continuación se presentan las disciplinas del cómputo a las que se recurrió para el diseño y desarrollo de este proyecto. Cada uno de estos apartados podría ser examinado más profundamente, sin embargo, las herramientas que se utilizaron permiten simplificar el uso de estas disciplinas y explotar al máximo sus ventajas para obtener resultados en un menor tiempo.

2.3.1. Programación Orientada a Objetos

La programación orientada a objetos permite abstraer los objetos de entidades del mundo real con acciones y atributos. La interacción de los objetos, dentro de algoritmos especiales, es lo que conforma una aplicación que toma ventaja de este paradigma. Finalmente se ofrece una visión más modular y portable, además de una facilidad inherente de reuso de código, que solo era posible en la programación estructurada a través de funciones que no siempre podían ser usadas en otros programas con requerimientos particulares.

Sin duda la metodología orientada a objetos es vital en el desarrollo de este proyecto, puesto que la mayoría de las librerías empleadas se basan en esta filosofía. La programación orientado a objetos es solamente uno de los paradigmas que han surgido para reemplazar el uso de la programación estructurada (otros son el uso de módulos o el uso de lógica, como son *Modula* y *Lisp*). De la misma manera en que la programación estructurada mejora y cambia la programación en ensamblador, la programación orientado a objetos mejora y cambia a la programación estructurada. Para que se entienda mejor esta idea, por favor lea acerca la historia de la programación en el capitulo *"Historia de la programación y su importancia en los videojuegos"* del anexo.(Capítulo. A.3)

Un objeto se puede entender de mejor manera si se plantea la manera en cómo se implementaría en lenguaje C. En C existen tipos de datos especiales llamados estructuras y uniones que sirven como un conjunto de datos primitivos (o incluso de otras uniones y estructuras). Si se necesitara asociar funciones a estas estructuras o uniones en lenguaje C, se debe

crear una referencia por medio de apuntadores. Las *Clases* son un tercer tipo de conjunto de datos que permiten asociar funciones además de ofrecer otras características especiales que hacen posible la POO. Las clases tienen permisos y no necesariamente hacen referencia a una función por medio de apuntadores. De esta manera vemos que una clase engloba tanto a tipos de datos como a funciones, lo cual define a un *Objeto*. Un objeto es una implementación de una Clase, una instancia o referencia a la clase a partir de la fue creado

Una clase no es un objeto, sino el plan de construcción de uno, por lo tanto una clase siempre debe tener un método/función llamado *constructor* y (para el caso de C++) un *destructor*. Cuando se quiere crear o declarar un objeto, se llama al método constructor de éste, y cuando se deja de usar, se llama a su método destructor.

2.3.2. Computación Gráfica

En un principio el problema de hacer un videojuego era que la mayor parte del tiempo no se dedicaba al desarrollo del mismo sino en la generación de gráficos (porque rara vez un juego se basaba en interfaces no-gráficos). Hoy en día, la mayor cantidad de tiempo se emplea en el desarrollo del motor del juego. Esto es posible debido a que se generalizó una solución para la generación de gráficos que debe integrarse con los fines y requerimientos del motor del juego. ¿Pero cuál es esta solución general?

A finales de la década de los setenta, inicio la revolución 3D. Empezó con la llegada de gráficos 3D a las estaciones de trabajo[4], logrando con esto que ya no estuvieran relegados a *Mainframes*[5]. El contraste era enorme - un *Mainframe* era, en comparación, más costoso de mantener y difícil de administrar, mientras que las estaciones de trabajo podían ser dedicadas a una sola tarea, con menos recursos y sin afectar a muchos usuarios (si llegasen a fallar). Los participantes en esta revolución se convertirían luego en los representantes del cómputo de alto rendimiento: IBM, DEC, Sun microsystems, CDC, InterGraph, Evans & Sutherland y Silicon Graphics.

A principios de la década de los ochentas se diseño PHIGS, una plataforma común para la generación de gráficos en 3D, sin embargo no era muy flexible, estaba basada en modo retenido[6] y también se basaba, ligeramente,

[4]Categoría de equipo de cómputo cuyo tamaño es comparable a un PC y con la potencia de una mini o supercomputadora

[5]Equipo de Cómputo de mayor tamaño que llenaba un cuarto entero y que normalmente servían a varios usuarios a la vez

[6]Esto significa que el render de la escena se queda residente en memoria

en grafos de escena (que se explicará en el punto 2.3.3). En particular Silicon Graphics desarrolló una solución lo más sencilla y optimizada posible, dejando que los programadores se preocuparan por cómo estructurar el dibujado de la escena. Así es como surgió IrisGL o como mejor lo conocemos hoy, OpenGL. En OpenGL no había objetos, solo primitivas; se ejecutaba en modo inmediato [7] y otorgaba mucha libertad en la forma de dibujar en la pantalla.

Años más tarde y con una proliferación de aplicaciones y soluciones basadas en OpenGL, se estimaba que esta sería la solución dominante en la industria del software para el desarrollo de aplicaciones 3D. Hoy en día esto es solamente una verdad parcial, debido a la llegada de otra solución, por parte de Microsoft, más usada por programadores de videojuegos. El nombre de esta solución es DirectX, siendo Direct3D el producto similar a OpenGL, ofrece casi las mismas ventajas que este. Entre las desventajas de Direct3D respecto de OpenGL se encuentra la exclusividad que tiene para las plataformas de Microsoft: Windows y Xbox. OpenGL por su parte, ya sea su última versión (3.0) o la versión embebida (OpenGL ES), se encuentran en prácticamente todas las demás plataformas; desde celulares, consolas de videojuegos, PCs y Macs, hasta Clusters de Computadoras.

Ahora se puede hablar de las dos soluciones (que desde ahora haremos referencia a ellas como APIs[8]), bajo los mismos términos cuando hablemos de gráficos, porque como se ha anteriormente mencionado, se basan en los mismos principios, pero con enfoques distintos. Las dos trabajan con primitivas y las dos funcionan en modo inmediato[9].

El elemento fundamental de los gráficos en 3D, es el vértice: una estructura de datos que contiene 3 números, ya sean enteros o flotantes. A partir de estos vértices se puede definir su conexión y si entre ellos se forma un plano ya que esta es la información necesaria para dibujarlo en pantalla. Generalmente están estructurados en triángulos o en cuadriláteros pero existen también planos con mas vértices. Dibujarlos en la pantalla, o el rendereo, se hace por medio de matrices de transformación particulares a dos sistemas de coordenadas - proyectado e isométrico. Desde el procesamiento de vértices hasta que se dibuja en la pantalla, pasa por una serie de estados que en conjunto se llama el *Pipeline Gráfico*.

Hasta hace algunos años, este *Pipeline Gráfico* estaba determinado por los arquitectos de los APIs o (de manera limitada) por quienes implemen-

[7] No se quedaba nada en memoria residente
[8] Application Programming Interface
[9] Direct3D implementó el modo inmediato a partir de la versión 5.0 de DirectX

taban las librerías de OpenGL. Recientemente, con Direct3D 10 y OpenGL 3.02 (con extensiones de NVIDIA), se ha vuelto totalmente reprogramable este pipeline por medio de shaders.

Más adelante se explicará como funcionan algunos aspectos del pipeline y los *shaders* (pag. 24)

Transformaciones

Lo relevante para entender la importancia de los vértices, es la forma en como OpenGL y otros APIs 3D, han manejado sus transformaciones en el espacio: traslado, rotación y escalamiento. Resulta natural pensar que los vértices se componen por ternas en el espacio, pero en efecto se usan cuaternas para facilitar su transformación por medio de matrices (y por la multiplicación de matrices se logra facilitar su procesamiento al CPU/GPU).

Traslación Partiendo de una cuaterna arbitraria

$$A = (x, y, z, 1)$$

se puede trasladar un vector en cualquier sentido por medio de la siguiente multiplicación:

$$A * \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{pmatrix} = (x + tx, y + ty, z + tz, 1)$$

Rotación La rotación depende mucho de que API se este usando (Direct3D usa coordenadas de la mano izquierda mientras que OpenGL de la mano derecha). Las rotaciones se realizan respecto a un eje y en secuencia (X primero, Y segundo, Z tercero) debido a que la multiplicación entre matrices no es conmutativa. En el caso de OpenGL, las matrices de rotación son las siguientes (donde ϕ es el ángulo de rotación):

Rotación sobre el eje x

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotación sobre el eje y

$$\begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotación sobre el eje z

$$\begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para tener nuestra matriz final de transformación, simplemente multiplicamos cada componente:

$$matrizDeTransformacionFinal = matrizx * matrizy * matrizz$$

Finalmente esta matriz resultante la postmultiplicamos por nuestro vector.

Escalamiento Partiendo de una cuaterna arbitraria

$$A = (x, y, z, 1)$$

, se puede escalar un vector por medio de la siguiente multiplicación:

$$A * \begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = (x \times sx, y \times sy, z \times tz, 1)$$

Shaders

Una tecnología que ha recobrado mucho auge en el 3D interactivo son los *shaders* que descienden directamente de los *shaders* del 3D fotorealístico.

Un shader, en su definición original, es un algoritmo que define el relleno de planos entre vectores o el dibujo de varios planos conformando una superficie. Puede ser tan simple como un algoritmo de iluminación plana (flat shading) hasta un algoritmo que calcule modelos de iluminación que simulan fotones y que incluye efectos visuales como agregar detalles en las texturas que simulan mayor complejidad geométrica (bump-mapping). Este proceso de control tiene su origen en los métodos de dibujo fotorealísticos (REYES, RenderMan, MentalRay) donde cada método definía su propio lenguaje de programación para estos algoritmos[10]. Subsecuentemente llegó a los métodos de dibujo en tiempo real y debido a la flexibilidad del hardware para el procesamiento de los algoritmos, se ha extendido al control de vértices y geometrías[11]. En lo que sigue se explicaran los *shaders* en este último contexto.[13]

Los *shaders* de tiempo real se realizaban, en un principio, en los mainframes gráficos (los SGI Onyx, por ejemplo) por medio de la biblioteca OpenShaders de SGI que se basaba en la generación de texturas en tiempo real y su almacenamiento temporal en memoria para luego encimar varias texturas una sobre la otra (con efectos de transparencia). Este método era extremadamente ineficiente y caro, por lo que se dejaron de utilizar las tarjetas dedicadas al pipeline a favor de GPU's[12] más versátiles. Ésto último dio por resultado la creación de varios lenguajes para programar *shaders*, dando lugar a la actual incompatibilidad de los programas *shaders* entre cada plataforma.

Como se había mencionado anteriormente, los *shaders* hacen posible cambiar al Pipeline Gráfico, ya sea de Direct3D u OpenGL. El pipeline original era muy rígido, fijo en su implementación y optimizado para los limites del hardware de SGI (ver figura 2.2). El uso de *shaders* cambia este esquema y se da mayor flexibilidad en la transformación de los vértices, las luces e incluso los fragmentos (antes de enviar todo al *framebuffer*[13]). (ver figura 2.3)

[10] Shaders fotorealísticos
[11] Shaders en tiempo real
[12] Unidad de procesamiento gráfico o GPU (acrónimo del inglés Graphics Processing Unit)
[13] Memoria de la pantalla o de la tarjeta de video.

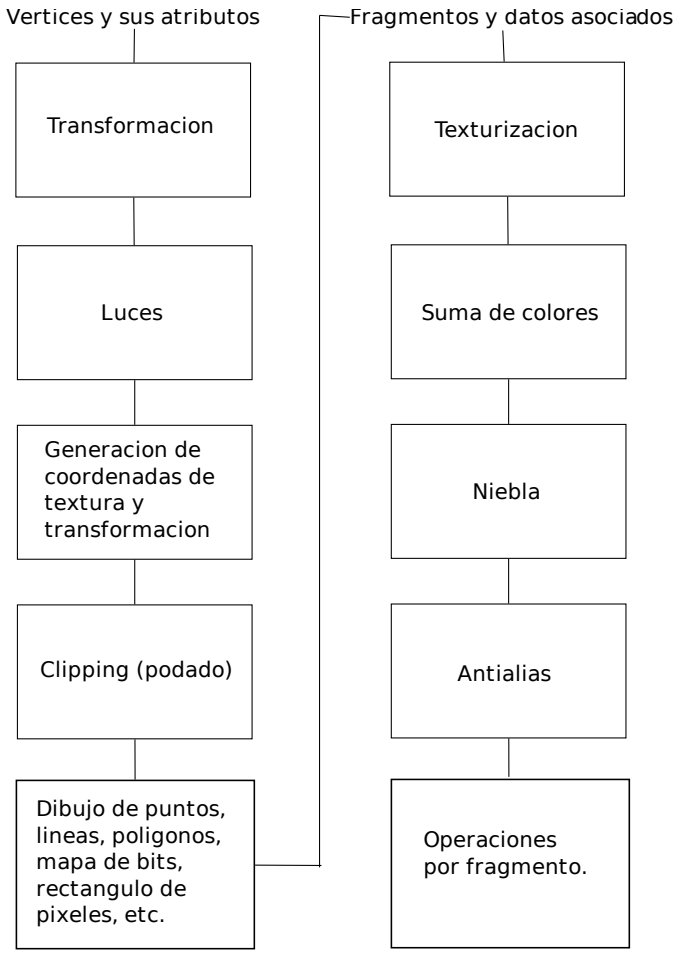


Figura 2.2: Pipeline Tradicional de OpenGL

Inicialmente para incluir código extra en el pipeline gráfico se recurría al ensamblador del GPU para incluir procesos extras, pero desde OpenGL 1.5, se ha podido programar en dos niveles: bajo (como ensamblador para gráficos) y alto (un lenguaje procedural tipo C). Ambos niveles tienen sus ventajas y desventajas, pero hoy en día, casi no se programa a bajo nivel.

En OpenGL, este lenguaje de *shaders* se llama GLSL[14]. En Direct3D se llama HLSL[15]. La empresa de Hardware 3D, nVidia, nos presenta con un tercer candidato que promete unificar los dos estándares, CG[16]. CG, para lograr unificar los estándares de shader, se construye a partir de los dos lenguajes y ofrece un lenguaje parecido al HLSL. En OpenGL se presenta como una biblioteca *wrapper*, mientras que en Direct3D aparenta una extensión o modificación de su lenguaje nativo.

Al poder editar el pipeline en un lenguaje universal (CG) se pueden hacer efectos especiales en los videojuegos que antes no se podía contemplar, debido a la gran demanda de recursos y la exclusividad para una sola plataforma.

[14] Graphics Library Shading language
[15] High Level Shading Language
[16] C for Graphics

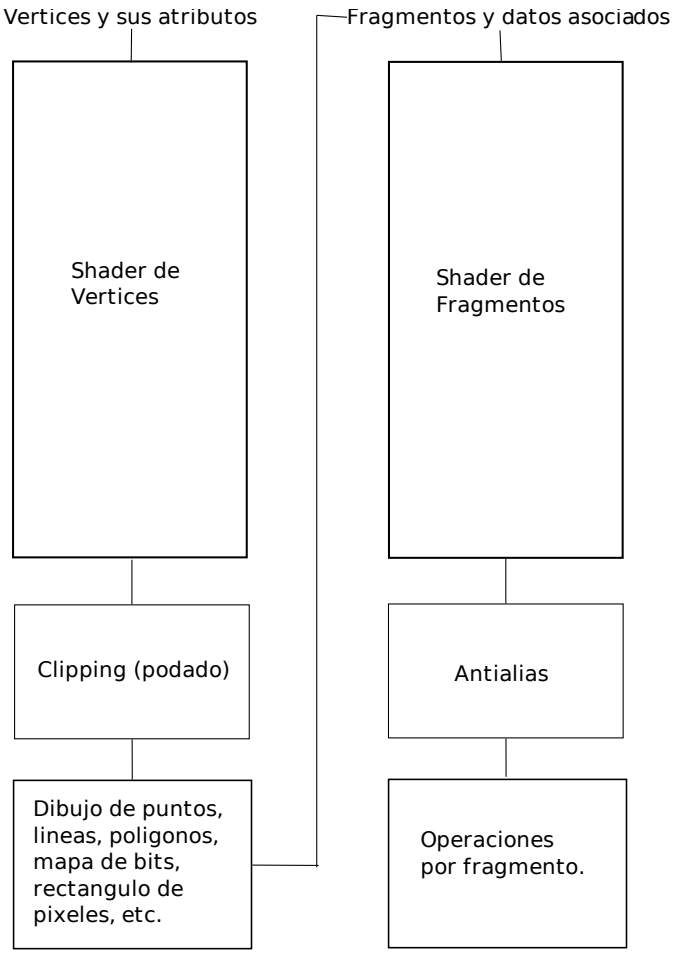


Figura 2.3: Pipeline con *shaders* de OpenGL

2.3.3. Grafos de Escena

Un grafo de escena es una estructura de datos que contiene descriptores de la geometría y apariencia de un escenario virtual así como la organización espacial de la escena. Los grafos de escena almacenan información en *nodos* que describen la geometría de dicha escena y otros nodos hijos que contienen

descriptores que afectaran a los que jerárquicamente descienden de ellos.

Para el cómputo gráfico, los grafos de escena definieron un nuevo paradigma para organizar la información de lo que se dibuja en pantalla.

La librería OGRE no es un grafo de escena, sino una implementación de este, para poder llevar a cabo las mismas operaciones en máquinas con menos recursos. La idea de los grafos de escena es establecer una cierta jerarquía a las geometrías y que sus transformaciones y forma de dibujarse sea lo más eficiente. Esto se logra organizando a los objetos en un árbol de n niveles de profundidad y n cantidad de hojas en cada nivel.

OGRE es un motor de render gráfico que utiliza las funciones de bajo nivel de manejo de memoria de forma que resulte más sencilla y portable su manipulación, permitiendo utilizar internamente la especificación de OpenGL o Direct3D a elección del programador (o dependiendo de la plataforma utilizada), aunque no es tan versátil o complejo como OpenSceneGraph, que sí es propiamente un grafo de escena.

Retomando la idea de un árbol, cada una de las hojas es lo que se dibuja en pantalla bajo diferentes contextos (principalmente dependiendo de la posición de la cámara y su campo de visión). Una de las características de los grafos de escena es que permiten eliminar del procesamiento a cualquier geometría que no este siendo visualizada por el usuario de manera eficiente (esto es, todavía existe en memoria, pero ya no se dibuja).

Los nodos que maneja OGRE, excepto el de la raiz, contienen la siguiente información acerca de los elementos que alojan:

- Transformación - Tiene su propia matriz de transformación.
- Grupo - Puede tener n número de nodos hijos.
- Gráfico - Puede tener n número de geometrías asociadas.

La forma en que se organizan estos elementos es análoga a las pilas en OpenGL. Con las pilas es posible definir una asociación espacial entre varias geometrías, ya sea de manera individual o de grupo, por medio de matrices de transformación. Las geometrías, a su vez, son arreglos de vértices asociados a un arreglo de índices. Los índices describen, en OpenGL, la manera de dibujar las caras que posteriormente generará una figura en tres dimensiones. El árbol de grafos puede verse como la evolución lógica de esta estructura de la pila, ya que brinda las mismas facilidades para crear y manipular geometrías además de presentar una forma eficiente para el dibujo de la escena.

A continuación se ejemplifica un sistema solar en un API 3D genérico (usando pilas):

```
/*Iniciar la pila*/
PushMatrix()
/*Camara/vista del usuario*/
Rotar(Rot\_Mouse)
/*Rotamos todo lo que esta dentro según las coordenadas del mouse*/
Dibuja\_Sol()
/*Función que crea una esfera que representa al Sol, con una textura amarilla*/
PushMatrix()
/*Creamos otra Matriz de Transformación que esta asociada al Sol
y gira cuando hacemos girar la cámara con el mouse*/
Rotar(x)
/*Rotamos x grados cada cuadro*/
Desplazar(10)
/*Desplazamos 10 unidades para dibujar al planeta Mercurio*/
escalar(.25)
/*Escalamos a un cuarto del tamaño de la tierra*/
Dibuja\_planeta(mercurio, rojo, 0)
/*Pasamos los parámetros para dibujar a Mercurio*/
PopMatrix()
/*Cerramos esta matriz pero aún estamos bajo la influencia de la
matriz anterior*/
...
... /*Dibujamos a los otros 7 planetas*/
...
PopMatrix()
/*Hasta aquí ya se ha finalizado el sistema solar, entonces se sale de la
influencia de la última matriz*/
```

A continuación se presenta el mismo modelo pero implementado con grafos de escena.

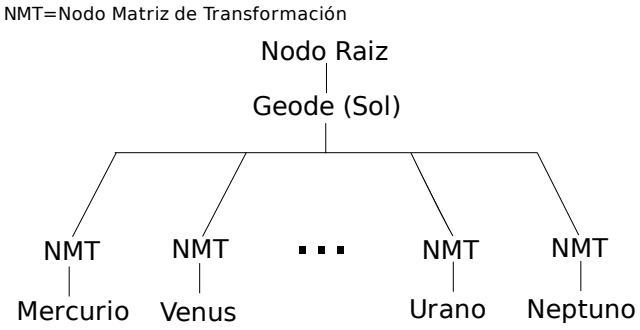


Figura 2.4:

La diferencia aquí es que cada traslación, rotación y escalamiento son métodos o funciones de los nodos NMT. Cada nodo también tiene un método para insertar un nodo hijo. Los nodos hijos, al igual que el nodo del Sol, tienen un método para asociarse a una geometría. Los grafos de escena engloban en objetos todas las operaciones que tradicionalmente se usan en un motor gráfico.

Dentro de las ventajas de emplear grafos de escena es la manera en como se *podan*[17] datos. Por un lado se asocian los objetos en el árbol por cuestiones de mecánica[18]. Por otra parte, se asocian dependiendo del punto de vista del usuario, para poderse llevar a cabo un podado eficiente. Según el campo de visión que tiene el usuario, es mas fácil de eliminar las raíces de varios nodos que no se observen, en vez de ir eliminando a cada nodo.

En lo anterior es en lo que se basa la filosofía de los grafos de escena y normalmente su optimización es lo que consume más tiempo del desarrollo de una escena.

2.3.4. Inteligencia Artificial

No es posible definir a la inteligencia de forma única pues incluye procesos psicológicos que es complicado describir. La inteligencia humana se relaciona con la capacidad de razonar sobre un problema, la palabra inteligencia es de origen latino, *intelligentĩa*, término compuesto de *intus* "entre" y *legere* "escoge", por lo que, etimológicamente, inteligente es quien sabe escoger, psicológicamente inteligencia es una capacidad cognitiva de aprendizaje y relación. La cognición se refiere a la facultad de los seres de procesar

[17] Se conoce como *podar* al proceso de eliminación de un subárbol

[18] Al crear un objeto, sus partes se ligan jerárquicamente, como sucedería al modelar un brazo, por lo que, los objetos dependen del movimiento de otro objeto, etc.

información a partir de la percepción.

La percepción esta relacionada al concepto de conocimiento. En el sentido filosófico se define como un modelo de la realidad en la mente. En el área de las ciencias técnicas y entre ellas la ingeniería, se puede definir como un conjunto de datos e información destinados para resolver un problema. El concepto de conocimiento nos lleva entonces al aprendizaje que se trata de un término asociado al ser humano y se refiere al proceso de adquirir conocimiento.

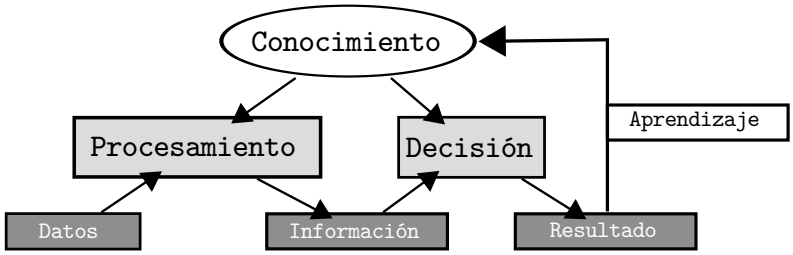


Figura 2.5: Esquema sobre el conocimiento desde el punto de vista de las ciencias de la información, como se genera y como se aplica[17].

La idea de implementar Inteligencia Artificial o *IA* [19] surge de hacer que el usuario o jugador pueda competir contra la computadora. En los primeros videojuegos no se tenía esta interacción debido a que se trataba de dos jugadores sin intervención de la computadora como contrincante.

Con la evolución de los videojuegos y de los recursos computacionales de los que se disponía, gradualmente se empezaron a utilizar técnicas de IA en videojuegos de un solo jugador. Un ejemplo de esto es *Pong*, en el que era posible jugar contra la computadora que usaba la información de posición de la pelota para realizar su movimiento.

Una técnica muy usada por estos primeros videojuegos era crear la ilusión de IA al implementar algoritmos aleatorios para alcanzar el objetivo deseado, el resultado era aparentar *intención* por parte de la computadora por obtener la victoria durante el juego, un ejemplo de esto lo podemos ver con los fantasmas enemigos en el videojuego *Pac-Man*, donde se puede percibir que los fantasmas buscan al personaje principal.

Un aspecto fundamental al implementar IA en los videojuegos, independientemente de la tecnología que se use, es diseñarla para maximizar la experiencia de interacción del jugador y brindar un nivel apropiado de *reto*; la computadora no siempre tiene que ganar o perder.

[19] Del las siglas en inglés *AI*, "Artificial Intelligence"

Otro aspecto importante es que el comportamiento de los personajes controlados por la computadora sea difícil de distinguir de un jugador humano; la *credibilidad* en el comportamiento de los jugadores por lo general es el objetivo principal en la IA de un videojuego, más importante incluso que simular ambientes reales.

A continuación se presentan los algoritmos utilizados comúnmente en la implementación de Inteligencia Artificial para los videojuegos actuales.

Búsqueda de Rutas con A*

El uso de algoritmos de búsqueda en videojuegos está relacionado con obtener la ruta más corta, si es que esta existe, entre dos puntos. De forma general, una búsqueda intenta solucionar problemas espaciales más abstractos, por ejemplo usando grafos de puntos conectados, entre los cuales existen distancia de unos a otros, donde un punto representa la solución deseada y otro el inicio de la búsqueda. A* es una de las técnicas más comunes en la mayoría de los videojuegos. El algoritmo A* (y muchas de sus variantes) utiliza heurística para conducir la búsqueda hacia el destino objetivo, resultando de ello un mejor rendimiento de otros algoritmos como los de búsqueda por profundidad (DFS[20]) o por anchura (BFS[21]).

Minimax

Minimax fue uno de los primeros algoritmos de Inteligencia Artificial que se usaron específicamente para el desarrollo de videojuegos, usado en juegos como damas o ajedrez por muchas décadas. Funciona haciendo que el personaje no jugador evalúe todos los posibles movimientos y seleccione el de mayor puntaje de acuerdo con con alguna métrica específica.

Máquinas de Estado Finito

Una máquina de estado finito[22] consiste en varios estados conectados entre sí, con varias posibles transiciones entre los estados y funciones que causan la transición entre éstos. Son comúnmente usados dentro de un videojuego en personajes que cambian entre comportamientos predefinidos,

[20] Del término en inglés, *Depth First Search*. Este algoritmo recorre todo el árbol de búsqueda o grafo de forma ordenada, pero no uniforme.

[21] Del término en inglés, *Breadth First Search*. Algoritmo de búsqueda sin información, que expande y examina todos los nodos de un árbol sistemáticamente para buscar una solución

[22] Del término en inglés FSM, Finite State Machines

por ejemplo, con guardias patrullando que escuchan algún sonido cercano y dejan su recorrido para investigar y posteriormente regresan a patrullar cuando se lanza una función de que ha pasado un tiempo definido.

Las máquinas de estado finito permiten definir lógica en un videojuego, pero a medida que esta lógica se complica en el desarrollo del juego, se vuelve más complicado controlar el comportamiento de los personajes debido a que esta técnica involucra programar muchos tipos de comportamiento. Si se pretende reusar el código que los generó (que es el objetivo común al utilizar esta técnica), las máquinas de estado deben ser sensibles al contexto, dependiendo de los objetivos del personaje en un momento dado y adaptándose al estado actual del personaje (por ejemplo, su nivel de energía), lo que vuelve complicada la lógica que debe producirlos.

Capítulo 3

Recursos Utilizados en este Proyecto

3.1. Requerimientos Generales

A continuación se presentan los requerimientos generales que se utilizaron para el desarrollo de este proyecto y las herramientas particulares que se emplearon para satisfacerlos.

- Equipo de Computo con tarjeta aceleradora de gráficos 3D
- Amplia documentación de como se puede hacer un videojuego
- Compilador de C++
- Modelador 3D
- Motor de render gráfico
- Motor de Inteligencia Artificial

El compilador de C++ elegido es GCC, que puede ser usado para cualquier plataforma, basada en sistemas UNIX, Windows. La ventaja principal de este compilador respecto a otros como: Visual C++, Intel o Borland; es que, siendo software de código abierto y disponible para todas las plataformas, el código del proyecto puede ser portado con mayor facilidad.

Un modelador 3D, es un software que permite crear mallas tridimensionales[1] que luego deben ser incluidas dentro del motor de juegos para controlar los eventos donde participan. Para este trabajo de tesis se usaron los modeladores 3D Studio Max, Blender 3D y Maya, que cuentan con extensiones o plugins adecuados para exportar las mallas creadas a las escenas desarrolladas por el motor de juegos.

Para la realización de este proyecto se requieren modelos tridimensionales de los personales, una historia bien estructurada así como un compilador de C/C++ en alguna de las siguientes plataformas: Windows, Linux o Mac

Existen muchas alternativas de motores gráficos disponibles, entre ellas:

- Crystal Space 3D. Conjunto de herramientas de desarrollo de software de visualización 3D, enfocado al desarrollo de juegos de video de código abierto, que puede usarse con el modelador Blender 3D.
- XNA. Microsoft ofrece este conjunto de herramientas integradas a su entorno de desarrollo .Net para juegos de video.

[1] El proceso de modelado de personajes tiene mayor relación con la parte artística de este proyecto y será explicada más adelante

- OGRE. No es un motor de juegos propiamente, esta enfocado a escenas 3D. Es un conjunto de librerías escritas en C++ con el propósito de administrar los recursos gráficos de la computadora.

Debido a que OGRE no es un motor de juegos, puede ser manejado con mayor versatilidad, pudiéndose utilizar para resolver problemas generales involucrados en el desarrollo del motor de juegos. Su versatilidad es una ventaja y desventaja al mismo tiempo, debido a que, siendo un software dedicado solamente a los gráficos, deben de usarse librerías externas para cualquier otra característica que se desee utilizar, como la entrada y salida de comandos, inteligencia artificial o interfaz gráfica de usuario. Esto dificulta el uso de OGRE, ya que se debe aprender a usar estas librerías externas e investigar sus características de forma individual. Pero estas características permiten mayor versatilidad a la hora de desarrollar un motor de juegos personalizado, como es el caso de este proyecto.

OGRE se eligió también por la amplia documentación disponible, tanto en fuentes impresas y principalmente en Internet, lo que facilita la implementación de las características que se planearon.

Para este trabajo de tesis se eligió una plataforma basada en UNIX, debido a que las librerías gráficas para esta plataforma se pueden utilizar en otros sistemas operativos, a diferencia de las librerías gráficas nativas de Windows por ejemplo. Particularmente se utilizó el sistema operativo Mac OS X, debido al software de desarrollo disponible para esta plataforma y tomando en cuenta que la mayoría de las librerías de desarrollo pueden ser instaladas en él.

3.2. Motor de Render Gráfico

3.2.1. OGRE

OGRE (Motor de Render Gráfico Orientado a Objetos)[2] es un motor de 3D flexible, orientado a escenas, escrito en C++. Su propósito es el de facilitar a los desarrolladores la producción de aplicaciones utilizando gráficos acelerados por Hardware. Esto lo logra por medio de una librería de clases que abstrae todos los detalles del uso de las librerías del sistema como Direct3D y OpenGL que proporciona una interfaz basada en objetos del ambiente y otras clases intuitivas.

OGRE no es un motor de juegos, aunque frecuentemente se le ha hecho referencia como tal, pero está diseñado deliberadamente para proporcionar una solución de gráficos solamente; para otras características como: sonido, redes, inteligencia artificial, colisiones, física, etc, se deben integrar con librerías externas.

Manejador de Escenas

Todo lo que aparece en la pantalla es controlado por un Manejador de Escenas (SceneManager). La clase del SceneManager se encarga de rastrear y mantener localizado cada uno de los objetos de esa escena. Cada vez que se crea una cámara, el Manejador de Escenas está al tanto de su posición en el espacio así como de entidades como planos, billboards, luces, etc.

Existe una variedad de Manejadores de Escenas para diversos propósitos, como la manipulación de terrenos, mapas BSP[3], entre otros.

Entidades

Una entidad para OGRE es uno de los tipos de objeto que se pueden dibujar en la escena. Se puede pensar en una entidad como cualquier objeto que puede ser representado por una malla 3D. Un robot, un pez son entidades; un terreno es una gran entidad; pero las luces, billboards, partículas y cámaras no lo son.

OGRE separa los objetos de su posición y orientación, lo cual quiere decir, que no se puede colocar una entidad directamente en la escena. Las entidades se deben asociar a un nodo de escena (objeto tipo SceneNode), el cual contiene la información relativa a su posición y orientación.

[2] Por sus siglas en inglés *Object-Oriented Graphics Rendering Engine*
[3] Binary Space partition - formato original de las escenas en el videojuego *Quake*

Nodos de Escena (SceneNodes)

Como se había mencionado anteriormente, un nodo de escena almacena la posición y orientación de todas las entidades que se asocian a él. Cuando se crea una entidad, no se dibuja en la pantalla hasta que se asocie a un nodo de escena. De la misma manera, un nodo de escena no es un objeto que se despliega en la pantalla. Solo cuando se crea un nodo de escena y le asocia una entidad (u otro objeto) puede llegarse a dibujar en la pantalla.

Los nodos de escena pueden tener cualquier cantidad de objetos asociados a ellos. Suponiendo un ejemplo en donde se quiere dibujar un personaje caminando por la calle con una luz iluminándolo, se procedería a crear primero un nodo de escena, luego la entidad del personaje y asociarla a este nodo y posteriormente se crea un objeto *Light* (de luz) y se asocia al mismo nodo. Los nodos de escena se pueden asociar a otros nodos de escena lo que permite crear toda una jerarquía entre nodos.

Otro concepto importante acerca de los nodos de escena es que su posición es siempre relativa a la de su nodo de escena padre y en cualquier manejador de escenas existe un nodo raíz al que todos están asociados.

3.3. Motor de Física

3.3.1. ODE - Open Dynamics Engine

ODE es una librería de alto rendimiento, libre y de código abierto, para la simulación de física de cuerpos rígidos. Es un API maduro, estable y muy intuitivo, independiente de cualquier plataforma (dado que esta escrito en C/C++). Integra facilidades para fricción, colisión y uniones ('joints')[4]. Es útil para simular vehículos, objetos en una escena de realidad virtual y criaturas virtuales. Actualmente se utiliza en varios videojuegos, aplicaciones de modelado 3D y de simulación.

3.3.2. OgreODE

OgreODE es un "wrapper"[5], originalmente escrito por "monster"[6]. La librería soporta las primitivas de ODE y agrega objetos prefabricadas para facilitar la implementación de vehículos y cuerpos muertos (ragdolls).

[4] Articulación entre varios cuerpos rígidos

[5] Interfaz que sirve de intermediaria entre dos API's o librerías distintas, comúnmente se considera como un *traductor* entre funciones de ambas librerías

[6] Autor contribuyente al proyecto de OGRE Addons - http://www.ogre3d.org/developers/addons

Originalmente esta librería estaba disponible para plataformas Windows y linux solamente. Afortunadamente durante el desarrollo de este trabajo de tesis se logro portar al sistema operativo Mac OS X, bajo su sistema de Frameworks, con el propósito de usarla en el entorno de desarrollo de este sistema operativo.

3.4. Motor de Inteligencia Artificial

3.4.1. Game::AI++

Game::AI++ es una librería de control de decisiones basada en la ejecución de árboles de comportamiento y está escrita en C++. El sistema de control en la toma de decisiones en Game::AI++ esta desarrollado para que múltiples árboles conectados entre sí representen comportamientos en los que cada nodo es una tarea específica que realiza búsquedas en los nodos hijo, regresando los nodos de solución y terminación de la tarea.

3.5. Modelador 3D

3.5.1. Blender

Blender es una aplicación integrada que facilita la creación de contenido 2D y 3D. Es un software que nos brinda una gran cantidad de facultades para modelado, texturizado, iluminación, animación y postproducción. Al ser de plataforma abierta puede funcionar en una amplia gama de sistemas operativos y procesadores, brinda además interoperabilidad entre sus diferentes versiones y posibilidad de expansión a través de plugins, ocupando poco espacio en memoria.

Desarrollado inicialmente para profesionistas de medios audiovisiales y artistas, se usa actualmente para la creación de visualizaciones en 3D usadas, por ejemplo en televsión o en producciones cinematográfica, también incorpora un motor de 3D en tiempo real como auxiliar en la creación de contenido interactivo.

3.5.2. Maya

Maya es una solución integrada de modelado, animación, efectos especiales y render. Es posible crear código interpretado personalizado por su entorno de desarrollo integrado. Maya tiene varias opciones adicionales en

forma de complementos (plugins) que le dan la capacidad de integrar modelos creados en esta herramienta a librerías como OGRE.

3.6. Entorno de Desarrollo

3.6.1. Xcode

Xcode es un entorno de desarrollo, desarrollado por Apple, para el sistema operativo Mac OS X. Disponible como un paquete opcional en el DVD de instalación de Mac OS X, también se encuentra la ultima versión disponible para miembros del ADC[7] e incluye todas las herramientas necesarias para crear, depurar y optimizar aplicaciones para Mac OS X. Cuenta con un IDE[8] que incluye un editor de código fuente, un sistema de vinculación, un depurador y un compilador basado en GCC para las plataformas Intel y PowerPC.

[7] Apple Developer Connection

[8] Del término en inglés "Integrated Development Environment", entorno de desarrollo integrado

Capítulo 4

Sistema Desarrollado

4.1. Análisis del Problema

El desarrollo de videojuegos es una tarea que se debe abordar desde dos puntos de vista (en ocasiones muy distintas) ya que se trata tanto de un producto de software como una obra de entretenimiento. Esta integración de visiones distintas requiere de la colaboración de dos equipos de trabajo, uno dedicado a crear la lógica con herramientas de desarrollo de sistemas y otro encargado de orientar este producto de software al público. Así se crean las reglas del juego, se diseñan los personajes, las escenas y la historia en la que se van a desarrollar. Esta es la razón principal por la cual existan diferentes géneros de videojuegos.

Cuando se inicia el desarrollo de un videojuego es común utilizar ideas de productos conocidos, partiendo muchas veces de los videojuegos más populares. Sin embargo, basar el desarrollo de un nuevo producto en un videojuego existente que está dirigido a un sector específico con características particulares, puede causar que el nuevo producto no aporte la originalidad necesaria para tener éxito.

Sí es válido basarse en productos existentes, solamente es necesario considerar que cada videojuego es una solución a un problema particular de sus diseñadores. Es responsabilidad de los diseñadores de nuevos juegos identificar sus objetivos y las diferencias que tienen con respecto a los videojuegos existentes para poder crear un producto único que se identifique como derivado del estilo de los primeros juegos.

Este videojuego está basado en un motor de juegos que mezcla dos géneros distintos. Se explicará el proceso que se siguió para el diseño de las escenas y el desarrollo de los objetivos para crear el motor de juegos separándolas del diseño de la historia.

El desarrollo de este proyecto comenzó analizando la separación de los elementos que conforman el videojuego, es decir se organizarán los recursos que se piensa utilizar. Entre estos elementos encuentran: el manejo de escenas y sus transiciones, la secuencia en el tiempo de éstas, la carga de memoria de los procesos, el diseño de la lógica y los elementos utilizados en el ambiente.

Lo que sigue, después de una breve explicación de la diferencia entre un motor y un videojuego, es un repaso de los elementos que se definieron para el análisis.

Un videojuego es la fusión de dos grandes elementos principales: el engine o motor de juegos y el videojuego propiamente que incluye la creación de los modelos y escenas. Partiendo de estos elementos se define la forma de jugar (lógica del juego) y una historia que da origen al diseño artístico de las escenas.

La distribución de tareas para este proyecto consistió en separar las actividades relacionadas con la lógica del juego y la construcción de las escenas usadas. Esto dio como resultado la creación de dos módulos principales que luego se integraron en una solución única.

Los personajes y los niveles (escenas en donde se desarrolla la historia) forman parte del módulo que desde este momento será denominada "Diseño de la Escena del juego" ya que consiste en elaborar artísticamente los aspectos visuales del videojuego. Por otro lado el sistema la base y el diseño de la lógica del juego se integran en el módulo que denominaremos "Motor de Juegos".

4.2. Diseño de la Escena del Juego

La idea de las escenas, en este caso, va encaminado a lo que se entiende en el cine por una escena. Existen en estas escenas, actores, edificios, acciones, luces, efectos especiales, etc. En el contexto de OGRE la mayoría de estos elementos son nodos y atributos de ellos.

Así como un director de cine se pregunta como se rescata la esencia de una historia, sin involucrar todos los detalles de él, también los diseñadores del videojuego. Se divide la historia o el tema propuesta de un videojuego en escenas que proyectan lo mas representativo de él y así poder minimizar la cantidad de trabajo necesario. De particular importancia es la ubicación de espacios físicos en un contexto temporal, esto es: ¿Dónde? y ¿Cuándo?. Se presenta el borrador de una historia donde, a falta de detalles finos, fue necesario inventar transiciones entre las escenas que se habían escogido. Para auxiliarse en esta tarea se hizo un guión ilustrado (storyboard o comic, vea la figura 4.1) y un diagrama de flujo. El diseñador tiene que tomar en cuenta los limitantes de su motor. A diferencia de una escena real filmada se tiene que dar prioridad al rendimiento en tiempo real de la escena virtual. Entre más cuadros por segundos se logra dibujar, con todo y efectos especiales, más convincente queda la ilusión.

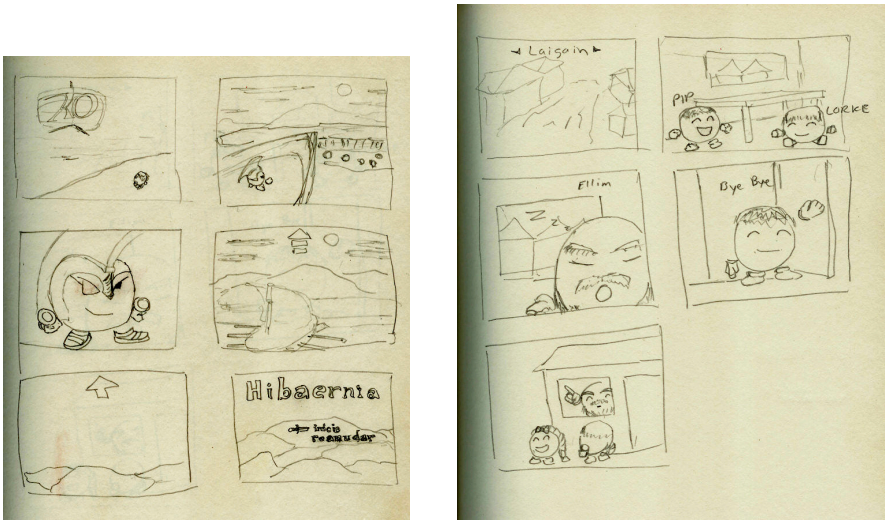


Figura 4.1: Bocetos iniciales del videojuego

Para el juego entonces, el diseño se tuvo que limitar a usar menos polígo-

nos (entre menos mejor) y usar texturas de tamaño estándar (128x128 hasta 1024x1024). Muchas de estas técnicas son las mismas que se usaron hace 10 años en el diseño de videojuegos 3D, otras de las técnicas son bastante nuevas (generación de partículas, terrenos, etc.), pero al final se pretende tener una serie de objetos optimizados para el que se tuviera un proyecto limitado en memoria. En la carga de escenas se recurrió a una tecnología poco convencional para un videojuego pero ideal para facilitar la edición de escenas, el XML[1]. Se usan documentos, 'páginas', en XML para describir el contenido de las escenas y dejar a un lado la carga en código.

4.2.1. Segmentación de Elementos de la Escena

La forma que toma la escena tiene una evolución propia. Para crear la escena, lo más cercano a la visión original se debe conocer a detalle los limites y las ventajas que ofrece el motor sobre el cual se trabaja. Para este videojuego, el motor se basa en OGRE y es preciso conocer que es lo que ofrece para la realización de sus escenas. Consecuentemente este proceso obliga a cualquier diseñador a familiarizarse con los conceptos de luces, cámaras, entidades, terrenos y como es que se incorporan en un árbol de nodos.

Se tienen claramente identificados a lo siguientes elementos de una escena:

Personajes Son modelos de personajes interactivos (y no interactivos) con animación.

Terrenos Geometrias extremadamente grandes producidas a partir de mapas de escala de grises.

Luces Por defecto no existe iluminación por lo que será necesario calcular sus formas y sus posiciones dentro de la escena.

Cámaras Son nodos que tienen que ver específicamente con el interfaz y con el manejo de eventos. Entender como se deben manipular es entender como se quiere observar y navegar a la escena.

Junto con el cargador XML se contó con efectos ambientales (el cielo), billboards o mamparas (para plantas y postes), geometrias con varios nodos anidados (particularmente los edificios) y efectos de animación generados

[1]Del ingles, EXtensible Markup Language

en tiempo real (el agua del mar). Todo lo antes mencionado son los elementos primordiales conformantes de una escena. A continuación se da una explicación de como se trabajan.

4.2.2. Aislamiento de los Elementos Conformantes

Personajes.

Los personajes del videojuego fueron de los elementos mas optimizados dentro de la escena. Se busco que fueran de fácil y rápido modelacion con el fin de hacerles modificaciones eficientes en caso de ser necesario. Utilizando mayores recursos se puede exigir la creación de personajes con más detalles y trabajar los modelos de tal manera que sean las geometrías que mejor aprovechan el tiempo de procesamiento de la escena (sin olvidarse de los limites que presenta el hardware). Para este juego fue necesario trabajar con la mayor austeridad en detalle.

Es importante recalcar la importancia de una preproducción (lo cual en muchos juegos queda ausente y puede llevar a que un desarrollo quede en un ciclo infinito). Los personajes, cuya visión original en la historia debieron ser antropomórficas, con el paso del tiempo fueron tomando formas mas simplistas y minimalistas hasta que se volvieron, literalmente, frutas y verduras (ver figura 4.2).

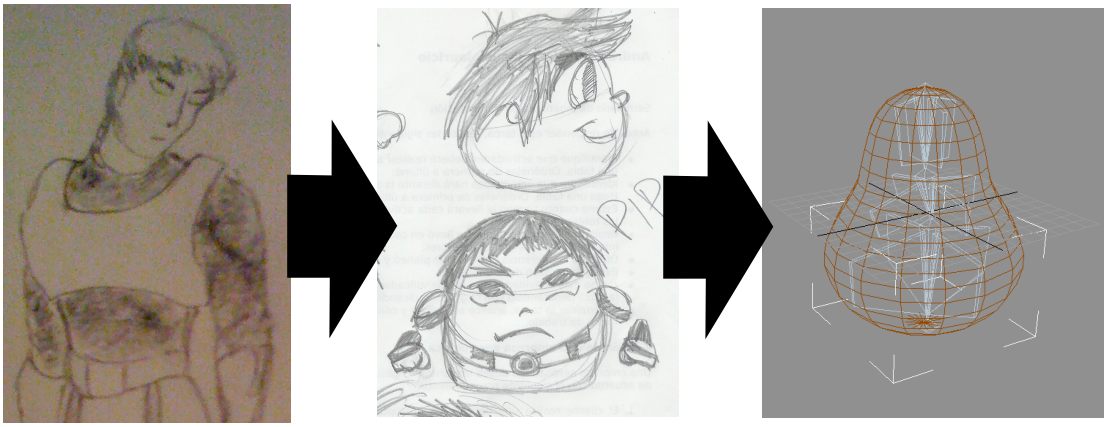
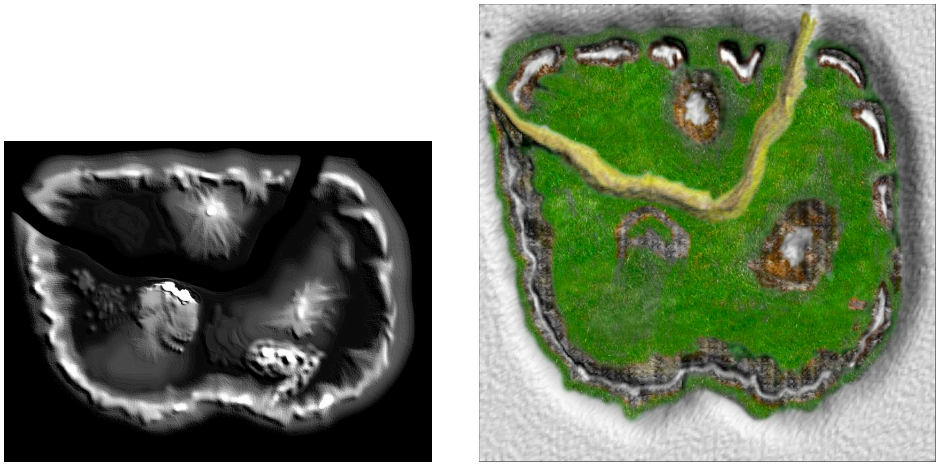


Figura 4.2: Evolución del diseño del personaje Pip

Terrenos

Dentro de las múltiples facilidades que ofrece OGRE, se tiene la posibilidad de generar geometrías de dimensiones exageradamente grandes con el fin de representar paisajes o terrenos. Antes, en los inicios del 3D interactivo, estos terrenos se generaban, al igual que cualquier otro nodo en el grafo de escena, en un modelador con sacrificio de detalle geométrico. Eran optimizadas en cuanto a cantidad de vértices pero se le podía atribuir mapas de textura muy grandes (o de hecho, varios mapas de texturas encimadas en capas). Para este caso particular se permite ser negligente con la optimización de la geometría, ya que para terrenos el motor gráfico recurre a algoritmos de generación en tiempo real. En otras palabras es posible tener terrenos muy detallados debido a que la geometría se genera no solo al momento de cargar la escena, sino al momento de atravesarla. El secreto esta en el uso de una mapa de grises (ver figura 4.3(a)).



(a) Mapa de Grises de un terreno (b) Textura del terreno

Figura 4.3: Elementos generadores de terrenos tradicionales

Los mapas de grises son herramientas muy comunes, incluso en la topografía, debido a que los diferentes valores de gris, de 256 niveles entre el blanco y el negro, representan alturas en un mapa de bits de dimensiones 2nx2n (el tamaño lo define cada motor gráfico, pero para el caso de OGRE es de $(2^n + 1) \times (2^n + 1)$).

También en las primeras épocas de los videojuegos 3D se usaba este método exclusivamente para la generación de un terreno en un modelador

(figura 4.4) donde posteriormente se optimizaba para las constricciones de memoria de la plataforma (lo cual, en los primeros años de la revolución 3D, era considerablemente poco, alrededor de los 4 a 16 MB). La memoria sigue siendo un factor muy importante en la generación de estos terrenos, pero ahora su tamaño se ve determinado por la posición del visor dentro de la escena.

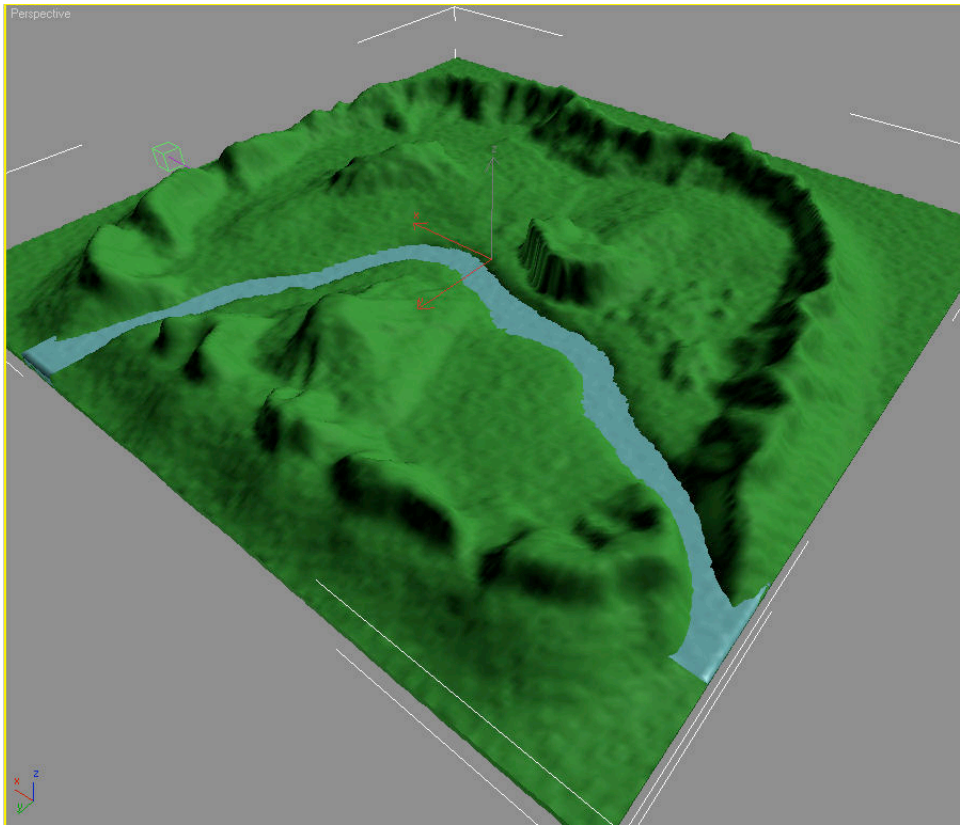


Figura 4.4: Terreno final en el modelador

Para explicar mejor esto ultimo, se describirán brevemente las curvas de Bezier. Casi desde los inicios de la computación gráfica, los ingenieros se veían forzados a desarrollar formulas mas precisas para describir curvas, ya que no existía una forma de representación exacta, matemática, de las curvas hechos a mano o por la curva francesa. La curva Bezier es una solución desarrollada por Pierre Bezier cuando su compañía, Renault, comenzo a modelar sus autos en la computadora. A lo largo de los años, la fórmula ha

comprobado ser uno de los algoritmos más precisos para preservar las curvas de forma digital (otros derivados de este algoritmo son los famosos B-splines y NURBS). Una curva Bezier se genera a partir de una serie algebraica, en donde una curva se ve más nítida y más preciso entre mas puntos de control se le define. También entre cada punto de control se puede especificar cuantas iteraciones (de segmentos rectos) se requieren para que la curva resultante se aproxime a la curva original.

Extendiendo la idea de a una maya de curvas Bezier, tenemos lo que se llama una superficie Bezier. Finalmente la optimización de esta superficie (para el caso del terreno) se realiza usando el siguiente algoritmo:

1. Entre mas lejos son los puntos de control a la cámara, menos iteraciones de segmentos se generan entre ellos (hasta el punto en que ya sean rectas entre cada punto de control). También a distancias ya mas grandes, se reduce la representación de estos puntos de control a simples pixeles (ya no hay necesidad de procesarlos más en los pipelines de OpenGL o Direct3D).

2. Los puntos de control más cercanos a la cámara, incrementan en el número de iteraciones, haciendo mayas más suaves. Incluso el terreno puede generar una serie de artefactos en la superficie (truco visual del pixel-shader) para hacer un efecto de bump-mapping (relieve).

OGRE genera los puntos de control a partir de la mapa de grises, esto es, cada pixel del mapa de grises es efectivamente un punto de control desde la cual se interpola la curva por medio de las series de Bezier. Entre más grande el mapa de grises, mejor definido queda el terreno, pero también llegan a ocupar mucha memoria. Al usar superficies Bezier se ha incrementado un tanto la cantidad de cálculos que se llevan a cabo en el procesador central pero no lo suficiente para afectar su rendimiento en otras áreas como la lógica del juego y la inteligencia artificial.

Billboards

Literalmente traducido quieren decir carteleras pero se entiende mejor el concepto si se ven como mamparas. Este sistema de mamparas es uno de los trucos más antiguos del 3D interactivo que existen para engañar al usuario. Su uso hoy en día se relega a situaciones donde el usuario no se percata (o no debería de percatarse) de los elementos que conforman el fondo de la escena (e.g. *Need For Speed Carbon*). Es muy utilizada para representar vegetación debido a que es ésto uno de los elementos mas difíciles de generar (debido

a los cálculos excesivamente pesados). En el contexto del juego son un poco más obvios debido a la falta de tiempo para implementar algoritmos para la generación de árboles y plantas.

La implementación de Billboards en OGRE existe pero se ha documentado muy poco. Fue uno de los elementos al cual se le tuvo que invertir mucho tiempo de desarrollo para poder implementarse en el cargador de XML.

Los Billboards son, desde la perspectiva geométrica, planos rectangulares que tienen asociado un material (textura estático/fijo o incluso un shader) y que tienen la particularidad de siempre estar posicionados paralelamente a la cámara (es decir nunca se da cuenta el usuario de que es un plano).

Un ejemplo clásico del uso de Billboards es en el juego de Mario 64 (tanto para el Nintendo 64 como para DS) en donde una gran cantidad de objetos y personajes son generados por este ilusión. Todos los arbustos, los árboles y los Bob-ombs (incluyendo a su rey) son Billboards con las cuales se puede interactuar. El usuario nunca se parcata de que es un billboard hasta que se acerca demasiado y se da cuenta que la textura sufre un efecto de antialias[2]. Lo que ayuda a la ilusión, en el caso de Mario 64, es la posibilidad de 'chocar' con los objetos (Mario puede subirse al 'árbol' a pesar de ser un Billboard, figura 4.5(a)) e incluso manipular los Billboards (Mario puede tirar a los Bob-ombs, figura 4.5(b)).

4.2.3. Manejo de eventos dentro de la escena

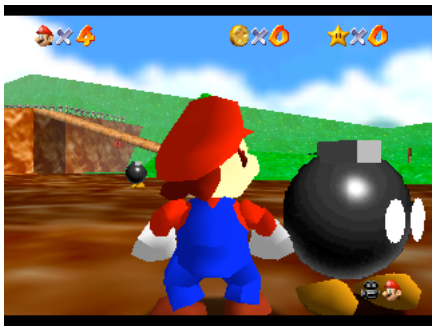
El manejo de eventos dentro de la escena es una parte esencial del diseño visual pero esta íntimamente ligado al motor. En el pasado algunos videojuegos recurrían a usar internamente un metalenguaje para describir tanto la lógica de la escena como la interacción del mismo juego.

Un ejemplo famoso, pero poco conocido, es el metalenguaje del motor de Unreal, llamado UnrealScript[21]. Este lenguaje se asemeja al lenguaje Java, pero a diferencia de él no es posible sobrecargar métodos ni distingue entre mayúsculas o minúsculas. En UnrealScript, todo el código describe a una escena en término de objetos. Todos los miembros o nodos de la escena tienen propiedades particulares, específicos del engine de Unreal, que le permiten un control más nítido sobre los eventos de los nodos. Los nodos/objetos tienen atributos que permiten saber si hubo colisión, si cambió de estado de reposo o si cambió de textura, etc. Se le llaman estados a estos atributos y son esencialmente lo que separa a OGRE de un verdadero motor de juegos.

La idea de usar scripts para la lógica y control de secuencias del video-

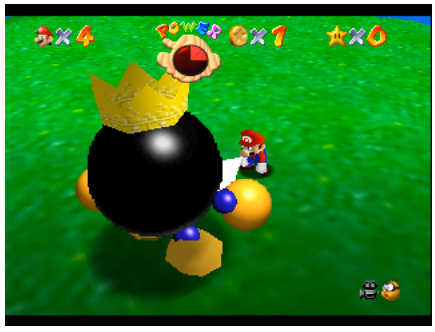
[2]

(a) Mario sube a un Billboard



(b) Los Bob-ombs son también Billboards

(c) El cuerpo, aparentemente esférica, del rey Bob-omb...



(d) ...es un Billboard.

Figura 4.5: Billboards (Bib. [13])

juego, no es nada nuevo. Cuando IBM lanzo el PCjr en 1983, donaron un motor para videojuegos, llamado AGI (Adventure Game Interpreter)[16], a la compañía de videojuegos Sierra Online. Este motor fue mejorado por Sierra e implementado en casi todos sus primeros videojuegos. Consistía en un motor que, por medio de un lenguaje parecido a C, cargaba los elementos gráficos, los sonidos y la lógica de un juego a memoria. Una vez acabado el script se 'compilaba' (en realidad se convertía a un formato binario nada mas, similar al bytecode de Java) y se corría agregando un argumento al motor indicando la dirección de tales datos. Lo mismo sucedía con su sucesor, el motor SCI (Sierra's Creative Interpreter)[3] y más tarde con el famoso motor de LucasArts. SCUMM[2]. Al igual que el motor de Unreal, la intención fue quitar todo lo frágil, vulnerable y patentable del motor y dejar a los demás elementos lo mas flexibles y editables posible para no tener que modificar directamente al motor (y evitando el desarrollo de código espagueti[3]).

[3]Código que incluye saltos GOTO o que son muy difíciles de interpretar para otros programadores

4.3. Motor de Juegos

Un motor de juegos es un producto de software encargado de administrar los recursos que componen un videojuego, esto se logra con la definición de funciones encargadas de procesar las entradas de información tanto del jugador como del propio ambiente virtual, para lograrlo hacen uso de una cantidad enorme de recursos computacionales, que incluyen el cálculo de las reacciones de factores como la física del juego que influye en la forma de moverse y actuar sobre el ambiente; la inteligencia artificial que toma la información del entorno y la convierte en conocimiento para los personajes que decidirán las acciones que realizarán en consecuencia.

Este trabajo aborda el desarrollo de un motor de juegos especializado en juegos 3D de estrategia y acción, cada uno de estos podría abordarse de forma independiente. Para integrar estos dos géneros se propone diferenciarlos por el control del personaje y por la posición de la cámara activa.

El motor de juegos se puede ver como un sistema cuya salida es el comportamiento de los elementos que conforman el ambiente virtual. El ambiente virtual en un juego posee propiedades en las que los elementos se desenvuelven, estos elementos se pueden dividir en gráficos y no gráficos.

Un motor de juegos de acción generalmente se muestra al personaje desde atrás a una distancia determinada siguiendo al personaje. En un motor de estrategia la vista es aérea con la cámara en posición variable y el control de personajes se basa en ordenes que se dan a uno o varios miembros del equipo del jugador.

En ambos casos la inteligencia artificial tiene un papel muy importante pues en todo momento los personajes que no son controlados por el jugador deben de tomar decisiones para sobrevivir o para cumplir el objetivo del nivel específico del juego.

En este trabajo se abordará el desarrollo de este motor desde ambos enfoques, primero el de acción ya que para realizar este se necesita tener listo el modulo inteligencia artificial para los personajes no jugables, que en este caso pueden ser enemigos o aliados.

4.3.1. Personajes

Los personajes son entidades que actúan en el ambiente con comportamiento propio, el cual puede o no, recibir la influencia de parte del usuario del sistema o jugador. Los personajes serán llamados personajes controlados y autónomos dependiendo de su pueden actuar independientemente de la interacción del jugador y los que deben tomar decisiones propias sin inter-

vención del jugador, respectivamente. A su vez, estos personajes se dividirán en personajes aliados y personajes rivales que serán descritos mas adelante.

Los personajes autónomos se verán con mayor profundidad ya que obtienen información directamente del ambiente para poder tomar decisiones por sí mismos, esta clase de personajes puede estar dentro del grupo de personajes aliados o rivales. Los personajes obtendrán la información del ambiente independientemente de la influencia del jugador, ya que las acciones del jugador pueden cambiar las condiciones del ambiente en todo momento. Los aliados o rivales actuarán entonces de manera similar pero con objetivos diferentes y en ocasiones opuestos.

Personajes aliados

El objetivo de los personajes aliados es seguir las ordenes del jugador para alcanzar un objetivo que contribuya a ganar la partida. El método para ingresar órdenes depende del modo de perspectiva que haya elegido el usuario, ya sea tercera persona o vista aérea.

En el modo de vista aérea los comandos se introducen después de seleccionar uno o varios personajes, en la interfaz gráfica de usuario se presentan las posibles acciones a realizar. Luego de que los personajes reciben las ordenes del jugador, los personajes aliados harán lo posible por cumplir la orden. Si por ejemplo el comando fue de desplazamiento de unidades y al moverse el grupo de personajes se encuentra con personajes rivales, los personajes aliados atacarán a los rivales con el fin de poder cumplir el comando pedido, ya que si son atacados no podrán alcanzar la posición requerida. Si el comando solicitado es de atacar a un personaje o grupo de personajes rivales, los personajes aliados que reciban la orden intentarán derrotar al grupo de personajes rivales a los que se les ordenó atacar, pero si están siendo derrotados y solo resta el 10 por ciento de los personajes aliados enviados al ataque, intentarán retirarse para evitar perder mas unidades.

Este modo de juego es utilizado en juegos de estrategias, sin embargo, en este tipo de juegos factores como los recursos utilizados y producidos son claves para el juego. En este proyecto de tesis no haremos énfasis en este punto ya que requeriría otro tipo de lógica por parte de los personajes tanto aliados como rivales, lo que se considera adicional al cubrir el objetivo principal que es crear un juego con perspectivas seleccionables por parte de jugador.

En el modo de vista en tercera persona, el ingreso de comandos al personaje aliado seleccionado es muy semejante a un juego de plataformas, en el cual el jugador toma el papel del personaje y las acciones que realiza, son

como si se tratara del propio personaje.

La interfaz de comandos utilizada en este caso, es el ratón y en menor medida el teclado, al presionar el botón principal del ratón y arrastrar el ratón, se podrán realizar la mayor parte de las acciones, si el ratón entra al área donde esta dibujada la interfaz gráfica, el ratón se usa como en cualquier otro programa, para seleccionar las opciones disponibles, como cambiar de modo, o por ejemplo el arma que se va a utilizar al momento del ataque.

Finalmente en la perspectiva de tercera persona se podrán elegir diferentes distancias de la cámara principal, utilizando la barra espaciadora.

Personajes rivales

El objetivo de los personajes rivales es ganar la partida, ya sea derrotando al jugador o cumpliendo un objetivo determinado.

Tratar de ganar la partida significa que los personajes rivales tendrán objetivos similares que los personajes aliados, solo que para beneficio de la entidad rival que será análoga al jugador para los personajes aliados. La entidad rival, sin embargo, no tendrá que enviar ordenes a los personajes rivales como lo hace el jugador con los aliados, en este caso los personajes rivales tienen completa autonomía de acciones para poder cumplir con el objetivo final que es ganar la partida. La tarea del jugador rival es ser el representante del equipo, almacenando los puntos obtenidos de experiencia, detectando el estado del equipo del jugador con el fin de informar a los personajes rivales, y estos puedan tomar decisiones sobre ataque o defensa de su territorio o unidades.

Personajes autónomos

Un personaje autónomo es uno que durante el curso de un juego de computadoras o animación, puede decidir como comportarse por si mismo. Para hacer un personaje autónomo, necesitamos un modelo computacional del comportamiento del personaje. Un personaje autónomo además necesita representar explícitamente un poco de conocimiento de su entorno. En el caso más simple, este conocimiento pueden ser unas pocas variables que indiquen el estado actual del mundo del personaje.

Una alternativa mas interesante y elaborada es que el personaje pueda mantener una representación explícita de como su mundo puede cambiar. Este conocimiento de la dinámica del mundo del personaje se le llama Conocimiento de su Dominio. Para poder distinguir entre estos términos, diremos que los personajes tiene conocimiento de su dominio como personajes autónomos.

4.3.2. Descripción de la escena

Elementos de la escena

Los elementos de la escena son los componentes que intervienen en el desarrollo de juego, los cuales pueden ser gráficos o no gráficos. En la figura 4.6 se pueden apreciar todos los elementos gráficos que intervienen en el videojuego, por ejemplo, se puede observar el campo de visión propuesto para los personajes así como un perímetro que es el límite espacial para poder comunicarse con personajes cercanos.

Un elemento gráfico dentro de la escena puede ser el tablero si se trata de un juego en dos dimensiones o un terreno tridimensional si se trata de uno en 3D. El tablero del juego (figura 4.6) a su vez aloja a los personajes que intervienen en el juego, así como los elementos propios del ambiente, este proyecto de tesis hará énfasis en los elementos activos del juego, es decir, los personajes.

Como ejemplo de elemento no gráfico se puede mencionar la física del juego. Se trata de cálculos de las fuerzas que actúan sobre los personajes y el ambiente virtual, el procesamiento de estos cálculos y la implementación de los modelos físicos sobre los elementos del juego le dan realismo al ambiente.

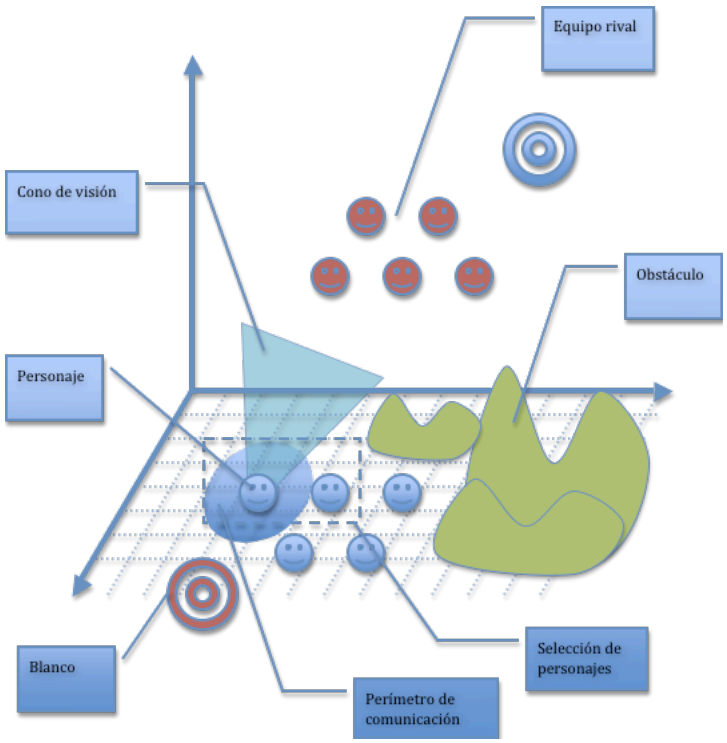


Figura 4.6: Tablero de juego

Tablero

El tablero de este videojuego es un espacio tridimensional creado a partir de tablas de altura y mallas estáticas que son los límites en los cuales los personajes basan su movimiento, aquí adquiere importancia el control de colisiones y la visión para la demostración del funcionamiento del motor de juegos, el tablero será representado con un plano y los obstáculos con cajas o piedras que no pueden ser atravesadas por los personajes.

La manera más sencilla de crear un tablero de juegos es dividir el espacio disponible en una cuadricula definida la cual separa a los elementos del ambiente, de esta forma los personajes pueden calcular la ruta a un destino especificado dentro del tablero u obtener la información de la posición de un personaje enemigo por ejemplo.

El motor gráfico OGRE ofrece la posibilidad de usar un plano infinito con el que se calcularán las colisiones con los otros elementos gráficos. Se usará sin embargo una malla simple de dimensiones finitas para visualizar el terreno, que para este caso será un área de 100x100 unidades utilizadas por

OGRE que pueden ser interpretadas como metros, centímetros o cualquier otra unidad.

Datos de entrada

Los personajes obtienen datos tanto del ambiente del juego como de otros personajes ya sean rivales o aliados. Estos datos permitirán que los personajes decidan que acción realizarán para cumplir con el objetivo.

- **Visión**: Intersección de objetos con el cono de visión del personaje.
- **Comunicación**: Solicitud y envió de datos a los personajes aliados en el perímetro de comunicación, estos datos se refieren a la localización de obstáculos y enemigos, basados en su visión.

Las siguientes son propiedades de cada personaje. Estos parámetros contribuirán en la toma de decisión al tratar de conservar la mayor cantidad de energía por ejemplo.

- **Velocidad**: valor inicial, entre 1 y 20 (aleatorio) (valor aplicado al movimiento del personaje).
- **Ataque**: valor inicial, entre 1 y 20 (aleatorio) (valor que se aplica al momento de ejecutar un ataque durante una pelea).

  Energía: valor inicial, entre 1 y 20 (aleatorio) (escalar aplicado al personaje utilizado en las peleas).

De forma predeterminada se tiene una velocidad, energía y un nivel de ataque igual para todos los personajes.

Posibles acciones para un personaje. Estas acciones se aplican para ambos bandos.

- **Movimiento**: El personaje calculará una ruta hacía un punto del tablero
- **Ataque**: El personaje iniciará una batalla para detener el avance del equipo rival.

Batalla

Sucede cuando se encuentran dos personajes de bandos rivales. Al aproximarse a cierta distancia inician una batalla (4.7), la cual consiste en lanzar

ataques al personaje rival quitando puntos de energía aleatoriamente dependiendo de su nivel de ataque (el valor aleatorio entre 0 y 1 es multiplicado por su nivel de ataque y esa cifra es la cantidad de energía restada al personaje enemigo).

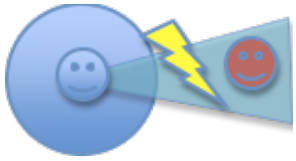


Figura 4.7: Batalla entre personajes de equipos rivales

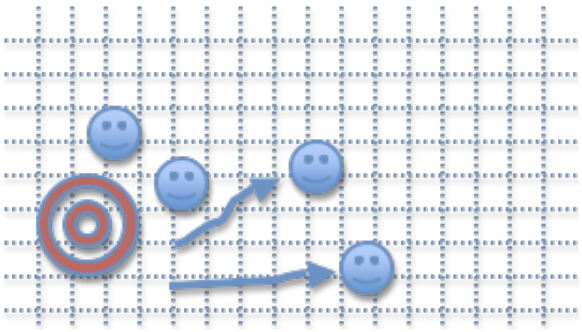


Figura 4.8: Elección de roles dentro del equipo.

Al iniciar la partida, los personajes reciben valores aleatorios para velocidad, energía y ataque; el valor asignado a la velocidad determina el rol del personaje, si es mayor a la velocidad del personaje con el que se está comparando, este personaje recibirá entonces el rol de atacante, de otra manera, su rol será defensor; el equipo deberá tener aproximadamente el mismo número de atacantes que defensores, favoreciendo el rol de defensor.

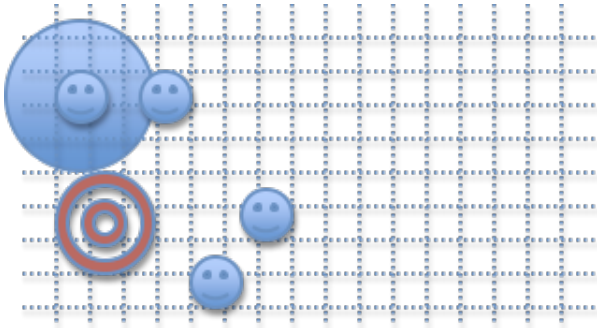


Figura 4.9: Los miembros del equipo se comunican y eligen su rol dependiendo de sus atributos.

El proceso de elección de roles se realizará con cada personaje que se encuentre en el perímetro de comunicación, de esta forma, si su valor de velocidad es mayor que el personaje con el que se está comunicando, su rol tenderá al de atacante, pero si se encuentra con otro personaje con un valor mayor de velocidad, su rol cambiará a defensor; inicialmente todos los miembros del equipo tienen rol de defensor, contribuyendo esto a que los personajes se puedan comunicar (pues el rol de defensor tenderá a estar a una corta distancia del objetivo que defienden).

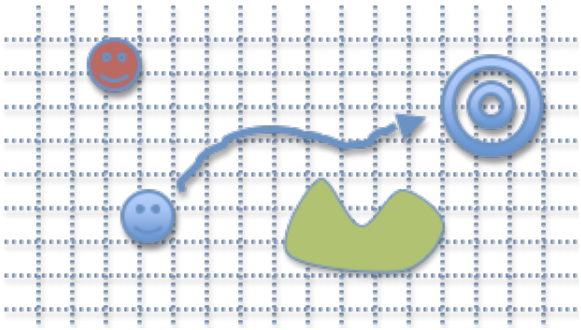


Figura 4.10: El equipo buscará una ruta hacía el objetivo contrario

El personaje debe encontrar la ruta hacía el objetivo (figura 4.10), evitando obstáculos y enemigos; si encuentra a un enemigo de frente que obstaculice la ruta hacía el objetivo sin posibilidad de evitarlo, se produce una batalla (figura **??**).

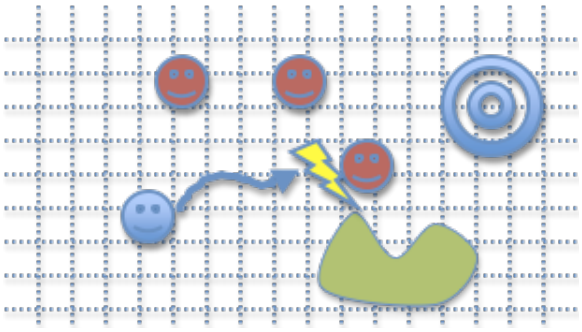


Figura 4.11: Si los personajes que intentan llegar al objetivo contrario encuentran enemigos y no pueden evitarlos, se produce una batalla

Defender su objetivo (figura 4.12) . Interponerse en la trayectoria del personaje rival hacía el objetivo; Iniciar una batalla con los personajes que se aproximan.

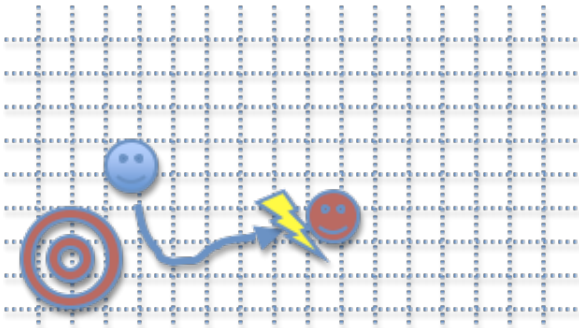


Figura 4.12: Si los personajes encargados de defender el objetivo no pueden evitar que los enemigos se acerquen, inicia una batalla para defender el objetivo

4.3.3. Implementación de Visión para los Personajes

Los personajes del juego obtendrán información que recibirán del ambiente para poder tomar decisiones sobre el siguiente movimiento que harán, la visión será su principal fuente de datos sobre tablero.

La aplicación de la información obtenida del ambiente a través de la visión se discutirá mas tarde.

Cono de visión

La visión de los personajes estará limitada por dos planos perpendiculares al plano del escenario separados entre sí por un arco de entre 150 y 170 grados (figura 4.13) con vértice atrás del personaje para dotarlo de una visión realista, la profundidad de su visión estará determinada por el ambiente, siendo como máximo el equivalente a unos 10 metros de longitud.

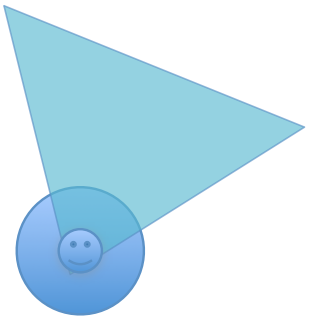


Figura 4.13: Representación del cono de visión de los personajes

Tercera Persona

Usando esta perspectiva el jugador obtiene la identidad del personaje que esta controlando, al colocar la cámara a espaldas del personaje para que el usuario pueda observar las características y los elementos cercanos al personaje.

Un cambio en la perspectiva de visión hace que las acciones del jugador se traduzcan en comportamiento, creando la sensación de identidad sobre el personaje controlado. Esta sensación dependerá entonces de la reacción del personaje que se controle, por tanto, mientras mas reactivo sea el personaje a las acciones de jugador, la inmersión del jugador en el ambiente virtual del juego será mayor. El control directo sobre los personajes y el cambio de perspectiva son elementos que no afectan sobre el poder que tienen los personajes autónomos para tomar decisiones, por lo que las reglas de adquisición de conocimiento para generar comportamiento son las mismas en esta perspectiva que las usadas para la de vista aérea.

Controlar a un personaje en el ambiente virtual de un juego incluye introducir comandos desde los dispositivos de entrada/salida de la plataforma,

que puede ser el teclado, ratón o algún control de juegos. La tendencia actual en la definición de controles de juegos es simplificarlos, las plataformas de juegos actualmente usan elementos de interacción sencillos como lápices con pantalla táctil o controles con sensores de movimiento además de los convencionales como pueden ser el teclado de computadora o ratón. La propuesta que hacemos esta basada en el uso del ratón exclusivamente, es decir, que el jugador no necesite usar el teclado de la computadora, o lo haga moderadamente.

Llamaremos a los dos tipos de control como: control en tercera persona y control de vista aérea. Cada uno tiene su propio sistema de instrucción para los personajes. El control para tercera persona funcionará como un juego común de acción/aventura, en el que se ve de espaldas al personaje controlado y un jugador controla al personaje como si se tratará de este. El control de vista aérea funciona como en un juego típico de estrategia, donde se necesita seleccionar a un personaje o grupo de personajes para darles acciones que realizar. Para los dos tipos de control, los personajes controlados interactúan con los autónomos, de forma que puedan tomar sus propias decisiones si no están recibiendo órdenes del jugador en un momento dado.

4.3.4. Control de Juego

El control para el motor de juegos tiene características particulares para cada modalidad o perspectiva. En la perspectiva de tercera persona el elemento principal es el uso de los periférico de la plataforma. En la perspectiva aérea el elemento principal es el terreno del ambiente, aquí el jugador debe ser capaz de controlar a un equipo de personajes y apreciar el efecto de sus acciones en el terreno.

Control en Tercera Persona

Durante este modo de juego el jugador verá de espaldas al personaje elegido, caminando en dirección de la vista del jugador, es decir, moviéndose con la misma orientación de la cámara.

El elemento principal de control es el ratón, arrastrándolo en la dirección del movimiento y dependiendo de la distancia del cursor al personaje, la aceleración del movimiento, el botón de acción será el botón principal del ratón (izquierdo), para un desarrollo futuro se espera poder implementar gestos para el ratón.

Se podrá elegir entre varios ángulos para visualizar al personaje elegido,

con la tecla espaciadora, permitiendo que el personaje elija entre dos o tres ángulos diferentes para visualizar mejor al personaje cuando así lo requiera.

Este tipo de control es el llamado de *juego de acción*, donde se controla a los personajes desde un plano separado del personaje, que surgió de los juegos de plataformas donde se podía apreciar el ambiente donde el personaje se mueve.

Control de Vista Aérea

El jugador verá a los personajes desde un punto más elevado en el ambiente virtual, necesita seleccionar a los personajes que desea comandar, este modo de juego es muy parecido al que se acostumbra en la mayoría de los juegos de acción.

De nuevo el elemento de control principal es el ratón, en este caso, para poder ordenar a uno o mas personajes que ejecuten cierta acción, se deben seleccionar el botón principal del ratón encerrándolos en una caja al arrastrar y soltar el ratón sobre los personajes elegidos.

Para este modo de juego la interfaz gráfica de usuario juega un papel protagónico, ya que, el jugador elegirá de ahí los comandos disponibles para los personajes seleccionados, algunas de las opciones disponibles son: moverse (que se puede ahorrar la interfaz gráfica de usuario al dar clic con el botón secundario del ratón sobre el punto elegido como destino), atacar (el personaje o personajes seleccionados intentarán atacar el elemento o personaje del ambiente elegidos después de seleccionar este comando) y cancelar (cancela la selección del personaje o personajes).

Este tipo de control es del tipo *estrategia* donde el jugador puede ver a todo su equipo aliado para dar instrucciones e intentar cumplir con el objetivo del juego.

Lógica de Juego

Se conocerá como lógica de juego a la forma en como los personajes que no están controlados por un jugador humano reaccionan ante los personajes que si lo están, así como las acciones que realizan los personajes dándoles un objetivo, el cual puede ser a favor del jugador o en contra, con lo cual se definen los personajes, aliados o rivales respectivamente.

Nuevamente este problema se puede dividir en dos partes fundamentales, la lógica correspondiente a los personajes aliados, y la lógica de los personajes rivales. Aunque en este caso se tratará del mismo algoritmo, simplemente cambiarán los objetivos definidos para cada tipo de personaje. Los persona-

jes aliados seguirán los objetivos dados por el usuario, es decir, obedecerán las ordenes del jugador e intentarán satisfacerlas. Los personajes Rivales, por otro lado tendrán objetivos definidos por su perfil que podrá ser creado desde la programación de la lógica o leído desde un archivo de configuración que podría estar en formato XML por ejemplo; los objetivos rivales obviamente serán contrarios a los del usuario, y principalmente intentarán hacer que el jugador pierda la partida.

Reglas para Ganar una Partida

- Que el número de integrantes de cada equipo sea el mismo.
- Cada equipo se dividirá en dos grupos, uno para defender su objetivo y otro para alcanzar el objetivo del equipo enemigo.
- Los personajes no podrán atravesar obstáculos por lo que deberán encontrar una ruta para evitarlos.
- Cuando algún miembro del equipo llegue al objetivo enemigo, el juego termina.

Antes de poder definir las acciones que realiza cada tipo de personajes se debe especificar como se gana una partida, ya que los personajes aliados y rivales tienen ese objetivo final.

Los personajes rivales integran un equipo, al igual que los aliados, cada equipo tendrá un numero variable de integrantes, que tienen sus propios atributos. A continuación se listan los atributos de los personajes.

La victoria en el juego llegará cuando la totalidad de los personajes de uno de los equipos sean derrotados, o el equipo se retire del campo de batalla (tablero). Una partida se define entonces como un periodo en el que se enfrentan los personajes rivales y aliados, esto es, de las acciones del jugador depende si se abre una partida.

Los personajes aliados y rivales son autónomos a niveles diferentes. Un personaje aliado es capaz de tomar las decisiones necesarias, basadas en las ordenes dadas por el jugador, tal que, al realizarlas se llegue al objetivo de ganar la parida. Un personaje rival, no necesita de ordenes establecidas, cada personajes rival tendrá la tarea de realizar acciones individuales y en equipo que ayuden a que su equipo (el equipo rival) alcance la victoria.

4.4. Inteligencia Artificial

En esta sección se presentarán de forma detallada las acciones que tienen que ver con la inteligenia artificial del juego, primero se definirán los requisitos para tener los personajes autónomos que ya se describieron en la sección anterior, a continuación, se presentará el análisis para conseguir esto.

4.4.1. Inteligencia Artificial en Videojuegos

La inteligencia artificial dentro de un juego controla el comportamiento de los personajes que no reciben órdenes del jugador, este comportamiento define el momento y la acción que realizan basados en un objetivo.

El comportamiento de los personajes depende del tipo de videojuego que se esté diseñando. Actualmente los tipos de videojuego son muy variados, tanto en su forma de jugarlos, como los temas que proponen. Existen género de juegos como: acción, deportes, estrategia, rompecabezas, aventuras, de rol, entre otros.

El motor de juegos que se propone en este trabajo permite elegir al jugador dos perspectivas, pudiendo escoger entre modo acción o modo de vista aérea. Esto se logra cambiando la posición de la cámara dentro del ambiente, cambiando el modo de control de los personajes y la visualización del mundo virtual.

En este capítulo, se presentará la estructura que elegimos para resolver el problema de crear un juego que tuviera dos puntos de vista para el jugador, lo que da la libertad a este de interactuar en el ambiente virtual del juego en dos perspectivas diferentes, una con la cámara a espaldas del personaje y otra con la cámara elevada para poder elegir entre varios personajes aliados y comandarles acciones.

A este tipo de separación, la llamados multi-modos o multi-perspectiva. Existen dos puntos claves a la hora de implementar esta idea, la construcción de la escena y la definición de reglas para los personajes. Estos dos conceptos aunque muy relacionados entre sí a la hora de construir la estructura de un juego, deben separarse, ya sea para poder tener una base de código fuente para futuros juegos o para poder expandir las capacidades del juego que se esta realizando. De esto surge la idea de crear un Motor de juegos y una escena específica para el juego de forma separada. Los elementos de estas dos partes de un mismo juego se deben relacionar de manera que resulte sencillo para un programador crear de forma sencilla su propia implementación, es decir, su propio juego, que es la meta final de este proyecto de tesis.

A continuación se presentan los elementos que se consideraron para

construir la lógica diseñada para implementar la inteligencia artificial en el videojuego.[10]

Conocimiento de Dominio

El problema aquí es que representar conocimiento en un personaje computacional es complicado debido a que basados en el comportamiento de un humano que tiene cierto sentido común, es complicado decidir si algo afecta al mundo o no, pensando en que el personaje autónomo no tiene sentido común y debemos dotarlo de ciertas nociones de conocimiento de su mundo acerca de la dinámica del mundo que lo rodea.

Instruir a los personajes

El comportamiento de un personaje está completamente determinado por una secuencia de acciones que se ejecutan. Esta es una simplificación común debido a que el resultado de estas acciones puede depender de factores ajenos al control del personaje. Por ejemplo, en un video juego, dos personajes pueden decidir correr hacía una montaña, pero si uno recibe un disparo en la pierna cuando va en camino, el comportamiento resultante de cada personaje se verá diferente.

Para personajes autónomos normales, que no poseen conocimiento de su entorno, necesitamos decidir de antemano que acciones ejecutarán para cada diferente situación que se les presente. En este caso decimos que el comportamiento del personaje es determinístico o predefinido.

Cuando el comportamiento del personaje no está completamente determinado de antemano nos referimos a un comportamiento no determinístico. El caso mas simple de comportamiento no deterministico es el que esta completamente no definido, pero un comportamiento aleatorio no es particularmente útil. En lugar de eso, el personaje necesita escoger su comportamiento para una tarea específica.

Un personaje consciente que sigue este comportamiento no predefinido puede elegir la acción más adecuada dependiendo de la tarea que tenga que realizar. Podemos decirle a un personaje los efectos de cada acción al darle "objetivos". El personaje puede entonces buscar la secuencia de acciones adecuadas para cumplir con el objetivo que le dimos, este comportamiento se conoce como "basado en objetivos".

Las ventajas y desventajas de estos tipos de comportamiento son variadas, pero siempre se debe encontrar un punto medio entre un comportamiento predefinido completamente y uno basado en objetivos para obtener un

rendimiento óptimo. La idea aquí es darle al personaje tanto conocimiento sobre su entorno, como un objetivo a realizar.

$Conocimiento + Instrucciones = Comportamiento$

El conocimiento del entorno puede usarse para planear objetivos y las instrucciones proporcionan un "bosquejo de plan" de como alcanzar esos objetivos. Técnicamente este sigue siendo un comportamiento basado en objetivos, y por tanto es fácil de especificar, pero si se agrega información de heurística definid, tiene el potencial de acelerar el tiempo de ejecución.

Un personaje que puede elegir acciones no deterministicamente significa si puede escoger una acción y además recordar las otras elecciones que ha hecho. Si, después de pensar acerca de esas elecciones, el personaje puede darse cuenta si el resultado será favorable o no, entonces puede reconsiderar cualquiera de las alternativas que resultarían de elegir otras acciones. Este tipo de personaje es fácil de ordenar, pero es mas lento en su tiempo de respuesta, si damos objetivos a los personajes, estos pueden elegir la acción adecuada entre sus posibilidades para alcanzar estos objetivos.

Adquisición de conocimiento

El tipo más simple de conocimiento que un personaje puede adquirir por sí mismo es el estado actual de su mundo. Este tipo de adquisición de conocimiento se conoce como "detección".

Discretización

Un personaje consciente traduce la información de fenómenos continuos de su entorno, como datos discretos, por ejemplo la posición de un objeto, normalmente la posición de un objeto dentro de un mundo virtual varia muy poco con el paso del tiempo, pero en el modelo de inteligencia de un personaje, podemos elegir representar la posición en un tablero discreto. Entonces, por ejemplo, si un objeto se mueve dentro de una de las celdas del tablero, su posición para el personaje no cambiará. Solo cuando la posición del objeto pase de una celda a otra del tablero será cuando el personaje observe el cambio de posición.

Incertidumbre

El resultado mas importante de que el personaje solo posea un modelo simplificado del mundo es que, desde su punto de vista, el mundo es ahora, de alguna forma, impredecible. Obviamente, si el modelo de conocimiento es demasiado simplificado, el mundo para el personaje parecerá demasiado

impredecible que no será capaz de usar el modelo de conocimiento para poder tomar ventaja de él.

Las personas y robot que interactúan en el mundo real, tienen que lidiar con cierto grado de incertidumbre todo el tiempo. Esto es, cuando una persona decide realizar cierta acción, no es completamente seguro de como le afectará el realizarla, entonces debe observar que ha pasado. Por tanto, para mantener esta correlación al modelo del mundo real, el modelo de conocimiento necesita ser actualizado regularmente con información del modelo del mundo real, a esto se refiere el término "sensado o detección".

4.4.2. Implementación en el Videojuego

Para lograr este nivel de abstracción en los personajes, se implementó un árbol de comportamiento jerárquico utilizando el concepto de "Máquina de estados finitos jerárquica"[4][12] el cual esta compuesto de estados que pueden representar acciones o transiciones del comportamiento. Con este tipo de máquinas de estado que estan basadas en la jerarquía de los elementos, secuencia de ejecución y comunicación. Con este tipo de diagramas se puede representar un comportamiento complejo con diagramas sencillos.

Este tipo de máquina de estados permite agrupar elementos para crear comportamientos más complejos.

El problema de crear una máquina de estados finitos[5] para la resolución de este tipo de problemas es que este tipo de estructura es muy específica para el tipo de problema al que se aplica. Para este caso se requiere que sea una máquina de estado con una lógica reusable de forma que se pueda adaptar después para propósitos particulares.

Árbol de comportamiento

A continuación se presenta el árbol de comportamiento creado para solucionar las acciones que realizarán los personajes. La lógica que se creó en este árbol está basada en la funcionalidad de la aplicación demo que se preparó para este trabajo de tesis.

La aplicación que se creó para demostrar las capacidades en este proyecto de tesis consiste en crear dos equipos, cada uno con cinco integrantes.

- Cada miembro del equipo recibe un rol al inicio del juego, uno de defensa y otro de búsqueda.

[4] Hierarchical finite state machines
[5] Finite state machines

- Los miembros que reciben el rol de defensa deben patrullar el objetivo de su equipo
- Los miembros que reciben el rol de búsqueda deben buscar el objetivo enemigo

Patrulla del objetivo (rol de defensa)
La patrulla del objetivo consiste en rodear el objetivo propio, buscar enemigos cercanos, acercarse a ellos e iniciar un ataque.

1. Identificar el objetivo propio Buscar enemigos, si encuentra, atacar
2. Definir una ruta alrededor del objetivo Buscar enemigos, si encuentra, atacar
3. Moverse a través de esa ruta Buscar enemigos, si encuentra, atacar

Buscar el objetivo enemigo (rol de búsqueda)
Los miembros del equipo que reciben el rol de búsqueda, deben encontrar la ruta hacia el objetivo enemigo y moverse a él. Esto debe realizarse buscando enemigos en el camino y equivarlos como si fueran obstáculos en el camino.

1. Identificar el objetivo enemigo Buscar enemigos, si encuentra, esquivar
2. Iniciar el movimiento hacia el objetivo Buscar enemigos, si encuentra, esquivar
3. Detenerse al encontrar el objetivo (Fin del juego)

4.5. Propuesta de Solución e Implementación

En este capitulo se presentaran los desarrollos y resultados obtenidos en el intento de implementar el juego. En muchos casos se podrá apreciar la dedicación de los autores al trabajar estos resultados, no obstante, también se apreciara lo que aun falta por desarrollar en futuros alcances. A continuación se dará un resumen de los prototipos de algunas áreas mencionadas en el capitulo anterior.

4.5.1. Escena

Se hicieron cuatro escenas que ejemplifican los inicios del juego "Lia fail". Como son la presentación final del diseño artístico del juego, sirvieron para visualizar el desarrollo de sus elementos conformantes que se mencionaran posteriormente.

Hibernia

El primero de las escenas que se trabajo fue hecho pensando en una animación para la introducción del juego. Se puede apreciar en esta escena a varios elementos: el uso de shaders en el agua, el uso prominente del terreno, el uso de niebla y el uso de objetos móviles (el barco).

Figura 4.14: Vista del barco romano llegando a Hibernia

Playa de Hibernia

Aquí se puede apreciar una de las primeras escenas que se tenia en mente desde el principio del desarrollo del juego. Vemos a uno de los primeros personajes que se hicieron para el proyecto (el nativo anciano, o cubo). También se aprecia el uso de shaders en el agua y el uso de terreno. Faltaria en esta escena geometrías que representan arboles y montañas en el fondo (a falta de tiempo, ya no se integro este fondo a la escena).

Figura 4.15: Playa de Hibernia con un personaje

Camino a Laigain

Es la ultima escena en crearse. Se puede notar a un personaje enemigo. Aquí se tiene estipulado que se desarrollaría la primera batalla en el juego. De nuevo se nota la falta de un fondo con arboles.

Figura 4.16: El Camino a Laigain con un enemigo

Laigain

El pueblo de Laigain, en donde se llevo la mayor cantidad de trabajo en modelar. Se aprecian nuevos elementos como chozas, Billboards y el ultimo de los personajes (Pip). También se le tenia destinado un fondo con montañas y arboles pero, al igual que las escenas anteriores, ya no se pudo incorporar por falta de tiempo.

Figura 4.17: El pueblo de Laigain

4.5.2. Personajes

Fuera de los personajes que se tenían ya hechos en la preproducción, se agregaron algunos mas para representar a los malvados del juego y los soldados constituyentes de los equipos de batalla. En breve daremos una descripción de cada uno.

Pip

Pip, aparte de ser la personalidad principal del juego, también fue el modelo con el cual se hicieron la mayoría de las pruebas de modelaje en 3D Studio Max. A partir de el se dejo de usar el modelador 3dsmax y se modelaron los siguientes en Blender. El personaje también posee una animación para caminar.

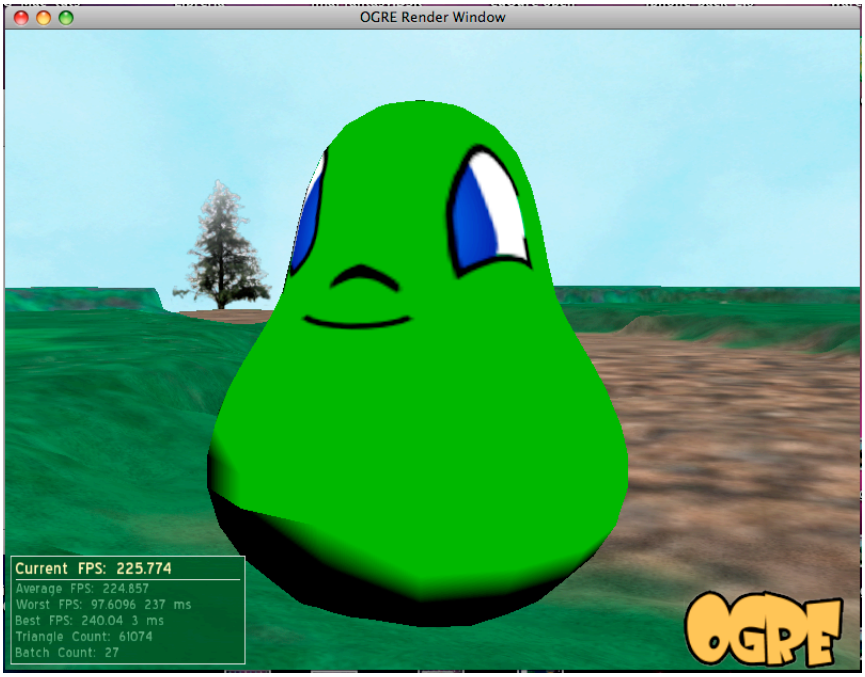


Figura 4.18: Pip

Soldados bueno (azules) malos (rojos)

Básicamente parten de la misma maya en 3D Studio Max. Sus diseños se basaron en dibujos conceptuales alternativas de Pip.

Figura 4.19: Soldado bueno enfrentando a soldado malo

Cubo/Anciano Nativo

Primer personaje hecho en Blender. Demuestra una gran cantidad de animaciones y técnicas de texturización (a partir de wrapping UVW). Con este personaje se representa uno de los primeros personajes imaginados para la historia (aunque menos sarcásticamente). Se espera que, junto con Pip, sea uno de los personajes que se pueden controlar en modo estratégico en un futuro.

Figura 4.20: Cubo/Anciano Nativo

Pepito/Enemigo

En realidad este personaje es la mascota oficial de uno de los proyectos del Dep. de Realidad Virtual en DGSCA (Ve3D), pero su personalidad bélica lo hizo víctima de una aparición cameo en el juego, para finalmente asentarse como uno de los personajes malévolos. También se modelo en Blender y no presenta animaciones por falta de tiempo.

Figura 4.21: Pepito/Enemigo

Barco Romano

Propiamente no es un personaje pero posee un rasgo singular en todo el juego, ya que no es un inmueble que forma parte del escenario ni tampoco es un personaje con los mismos atributos de formar un elemento estratégico. Es el único móvil que se contempla en todo el juego por el cual es un elemento interactivo. Se modelo a partir del esquema de un barco romano mediterráneo. Presenta rasgos parecidos a un barco explorador vikingo, por lo que solo se puede imaginar si de veras lo hubieran usado los romanas (de los primeros años del imperio) para alcanzar Hibernia.

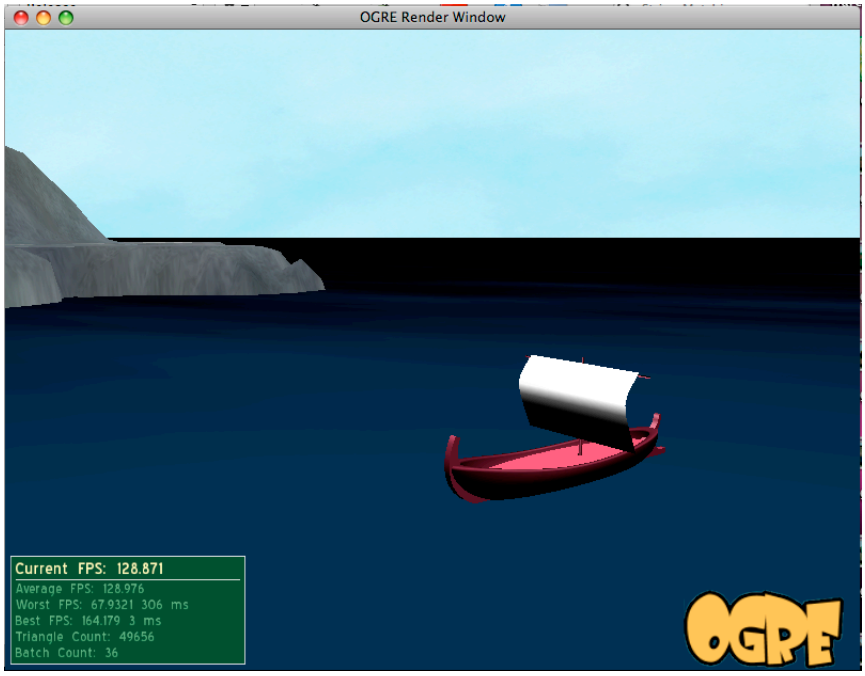


Figura 4.22: Barco Romano

4.5.3. Terrenos

Se hace hincapié en que el diseño de estos terrenos no fue un asunto trivial. A pesar de que OGRE sí dispone de todas las ventajas que ofrece OpenGL 2.0 (para ser preciso, OpenGL 1.5 + extensiones), se remite a usar los pixel-shaders lo menos posible (a menos que a alguien se le ocurre programar el uso de shaders en el modulo de terrenos). Siendo así entonces, ya los terrenos generados no se ven afectados por las luces ni por ciertas características del ambiente. Esto da, como consecuencia, la falta de sombras en los terrenos, lo que implica el uso de un viejo truco visual, que antes se usaba en los juegos, llamado Texture Baking (literalmente dorar una textura). En seguida se explica este proceso.

Texture Baking

En un modelador se usa otro tipo de pipeline de rendering para llegar a obtener resultados convincentemente reales o fotorealisticos (los cuales ya no sirven para dibujarse en tiempo real). Estos pipelines prestan sus cálculos intensivos y tardados para hacer una 'pos-producción' a las texturas que se van a usar en la escena generado en tiempo real. Los pipelines de rendering fotorealistico derivan de un algoritmo (en su mayoría) que fue usado por primera vez en LucasFilm, en Point Reyes California. El pipeline Reyes (desarrollado por la misma división LucasFilm que mas tarde se llamaria Pixar) simula casi todos los efectos reales y estocásticos de la luz natural, por medio de Ray-tracing. El Raytracing tiene sus orígenes en un programa que originalmente calculaba las trayectorias (y energías) de partículas radioactivas en un reactor nuclear. Debido a la naturaleza tan cercana que tienen estas partículas a los fotones, se integro al rendering fotorealistico. A consecuencia se tienen reflejos, refracciones, filtros y sombras muy parecidos a la vida real.

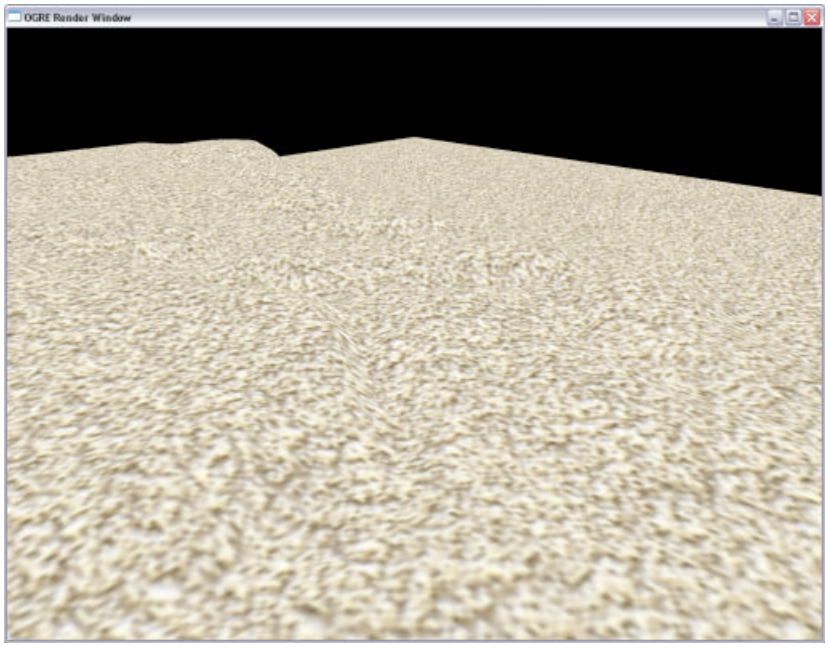
Se puede apreciar en la figura 4.23 como se ve el terreno de una playa en OGRE al corregirle gradualmente la textura y otro efecto visual que se llama bump-mapping (otra vez relieve). El programa que se uso para el texture baking de los terrenos fue 3D Studio Max.

4.5.4. Billboards

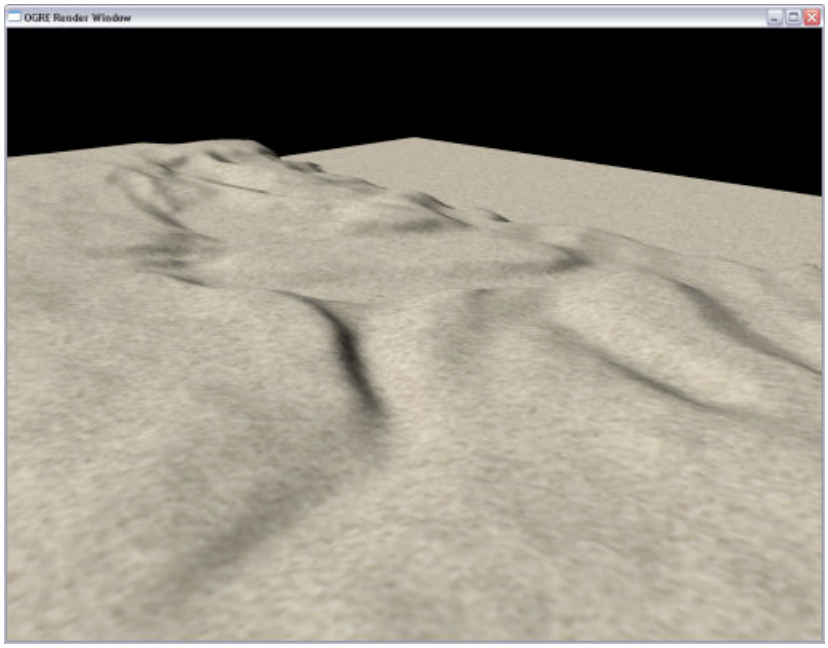
En los escenarios de Lia Fail se contemplaba usar Billboards para arboles y plantas principalmente. Su diseño involucraba mucha mas habilidad con cámaras y editores de bitmaps que con el propio código de OGRE o el cargador de XML.

Tomando como ejemplo a uno de los Billboards, el primero que se diseño, una planta que se encontró a lado de la rampa bicipuma (entre el anexo de ingeniería y la facultad de contaduría, en Ciudad Universitaria). Es un ejemplo de Billboard de los mas ideales debido a varios factores que mencionaremos a continuación. Siendo que estaba totalmente aislado de las otras plantas se elimino la necesidad de filtrar que partes eran de la misma planta y cuales eran de alguna otra planta.

Otro factor igual de importante fue que la planta no presentaba detalles muy finos. A un profesional que se dedica a trabajar texturas no le convendría tener que invertir mucho tiempo limpiando los bordes de una planta cuyas partes resultan ser de tres a un pixel de grosor. Se sugiere encontrar un objeto con detalles de gran grosor, a menos que sea totalmente necesario

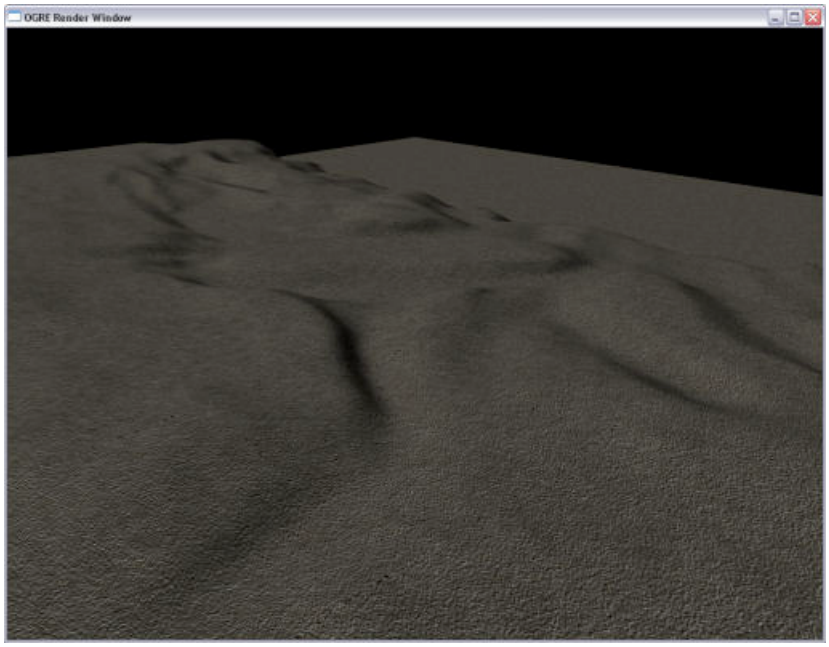


(a) Imagen sin texture baking

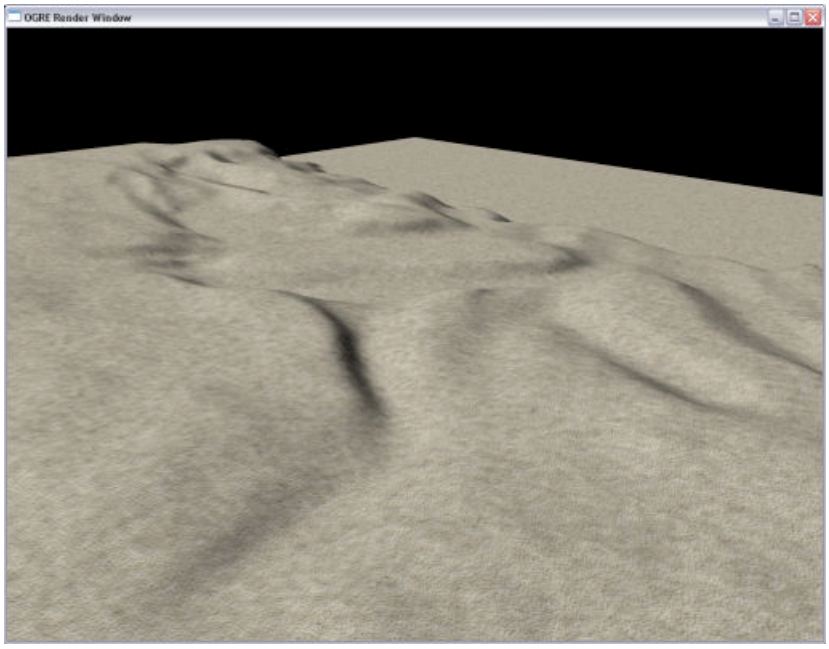


(b) Con texture Baking

Figura 4.23: Desarrollo de un terreno con Texture Baking



(a) Hasta aquí se ve fantástico la playa, pero ahora se va a experimentar con el bump-map de OGRE para observar cuanto mejora



(b) Después de hacer mas brillante al bump map (150 veces mas)

Figura 4.24: Desarrollo de un terreno con Texture Baking 2

para el escenario.

El siguiente factor importante fue que el fondo estuviera monocromático, esto es, que el fondo solo tuvo un color o tono lo cual también facilita enormemente la separación de la planta del fondo. Si se pregunta uno, en las películas donde no tuvieron una pantalla azul (o verde) de fondo y nada mas tienen la película original (con una variedad de fondos), ¿Porqué sale el proceso de cambiar el fondo o transportar al actor, tan caro y minuciosa? Es por lo mismo pero multiplicado por la cantidad de cuadros involucrados en esa escena. El fondo de la foto fue concreto blanco, lo que deja que nada mas se limpiara partes con distintos tonos de luz o que se veían sucios.

Un ultimo factor para escoger un Billboard ideal es el efecto de la luz sobre el objeto. Esto se nota en la figura 4.5 (pag. 51), donde el Rey Bob-Omb se ve en 3D debido a un reflejo del sol que nunca se mueve pero engaña lo suficientemente bien al usuario para hacerlo pensar que es una esfera en 3D. En la planta se observa un efecto parecido lo cual lo hace muy ideal para que se situé en medio de un campo abierto o al borde de un bosque, pero no dentro de uno. En dado caso que se quisiera situar la planta dentro de un bosque se auxilia de unas paraguas, antes de tomar la foto, para que no le pegue el sol.

Viene en seguida el pos-procesamiento de la imagen digital, donde se limpia la imagen de todo lo innecesario usando canales alfa. Dentro de las definiciones de materiales para OpenGL (y de manera derivada, en OGRE) se usan Mapas de bits de 3 o 4 bytes y de dimensiones que siempre se rigen en potencias de 2 para la fácil manipulación de la textura (normalmente sufre mutaciones al encimarse a la geometría). En este caso no se desea que el material sufra distorsiones al trasladarse al plano del Billboard. Para lo anterior la imagen se manipulo de tal forma que se viera bien siendo cuadrado y con el canal alfa se eliminaron todos los detalles y elementos demás que lo hiciera parecer como un mapa de bits cuadrada. Al final se guarda la imagen en un formato que soporta 32 bits (es decir, que soporta canales alfa) que pueden ser formato TARGA, PNG o TIFF. GIF es un formato opcional por si no se requiere de una gran gama de colores (En su caso se usa uno de sus colores indexados para efectos de transparencia).

Se puede apreciar los Billboards en la escena de Laigain en las figuras 4.25 y 4.26.

Una de las cosas mas importantes al usar Billboards es su ubicación y la cantidad de veces que se usan. Como se había mencionado antes se usan con menos frecuencia en los nuevos videojuegos debido a que se tiende a explotar mas la capacidad de las nuevas tarjetas gráficas para hacer escenas mas realistas y convincentes, disminuyendo su utilidad. OGRE tiene un sistema

Figura 4.25: Billboards 1

Figura 4.26: Billboards 2

de Billboards pero por lo mismo anterior, ha tenido poca popularidad y por lo tanto no se implemento en el cargador XML original. Gran parte del tiempo invertido en el desarrollo del cargador de escenas es debido a la programación de Billboards (al igual que los shaders).

Los Billboards en OGRE llevan una gran cantidad de opciones debido, mas que nada, al uso de arreglos. Los Billboards no se pueden definir individualmente en OGRE pero si se pueden crear arreglos con un solo Billboard, lo que complica su definición en XML. Gran parte de su declaración en XML ya estaba hecho y nada mas faltaba programar la correspondencia entre la etiqueta en XML y la clase Billboard. Visto desde un punto de vista, los Billboards eran arboles dentro del árbol de nodos en el grafo de escena, entonces la programación fue pensado en términos de como redefinir una etiqueta que actúe como si fuera una etiqueta ¡node¿.

4.5.5. Shaders

OGRE tiene soporte para tres variantes de Shaders: HLSL, GLSL y Cg. Para la necesidad particular del juego, en donde el único caso en donde se contemplaba el uso de shaders era la generación de agua, se decidió usar el lenguaje mas portátil de entre los tres - Cg. Todos los lenguajes de shaders tienen al menos 2 áreas donde se aplica la programación de shaders, estos son vértices y fragmentos (o pixeles en HLSL). En OGRE se llega a distinguir entre estas dos áreas pero se manejan como materiales. Como se menciono anteriormente el único shader que se llego a implementar en el juego, fue el shader del agua. Este shader tiene su propia etiqueta XML para incluirlo en casi cualquier escena. No se puede apreciar muy bien en las figuras 4.14 y 4.15, pero en esas escenas se llegan a usar. El shader de este elemento calcula en tiempo real una serie de ondulaciones en el agua que afecta tanto a su geometría como a su textura.

4.5.6. Manejo de escenas por XML

En un principio se tenia la idea de programar toda la carga de escenas por medio del engine. Se llego a la anterior conclusión por la forma en como estaba codificado la mayoría de los juegos (o se pensó). En realidad es muy distinto tener un metalenguaje, como UnrealScript, a tener una librería que se encarga de cargar todo a memoria, aunque se puede ver a uno como el inicio del otro (a pesar de que el XML no es un lenguaje Turing completo). Se puede decir que el cargador de XML es un intento de scripting provisional.

La librería de cargas de escena del juego principalmente es un derivado

de otra librería que no tuvo licencia pero sí una serie de autores quienes dejaron su documentación en las siguientes paginas:

http://www.ogre3d.org/wiki/index.php/DotScene_Loader_with_User_Data http://www.ogre3d.org/wiki/index.php/DotScene_Loader_with_User_Data_Class http://www.ogre3d.org/wiki/index.php/New_DotScene_Loader

A la hora de escribir, esta librería ya presenta ciertas adiciones, necesarios para el juego de Lia Fail, que ya se han mencionado: El manejo de Billboards en escenas y el manejo de un plano con efectos de agua (por medio de shaders).

En un desarrollo posterior se tiene contemplado incluir los siguientes elementos:

- Manejo de música de fondo y atributos de control sonido para los nodos (esto para tener un efecto de sonido al realizarse una acción en un objeto).
- Manejado de Escenas pre-animadas (películas).
- Manejo de Física y colisiones en la escena.
- Manejo de Atributos de los personajes.

4.6. Integración

La integración de algunos de los elementos anteriormente mencionados, específicamente los que conforman a la escena de Laigain, con el motor del juego desarrollado sera el objetivo final de la aplicación de demostración. Se espera que con la aplicación de demostración para plataforma Macintosh se logre representar la visión original del videojuego de Lia Fail.

El prototipo del juego se logró integrando el uso de recursos como colisiones, gravedad, interacción con el teclado, seguimiento de cámaras en la escena junto con el cargador de escenas directamente en el código fuente del programa demo. Finalmente se tuvo un motor que siguiera los lineamientos del motor del juego Quake, y no así la idea del motor del juego Unreal. Aun así, el resultado final es la creación de un motor versátil al que posteriormente será posible agregar características adicionales en el futuro.

Fue necesario, en la programación, relacionar los elementos de la escena con un correspondiente objeto de colisión conocido como “World”, un objeto de la librería OgreOde. Por el cual se hizo un relación de cual de los elementos de la escena se le asignaría el papel de personaje principal para hacer posible un punto de vista en tercera persona, esto es, el seguimiento de cámara. Lo

que siguió fue la programación de los interfaces con el personaje principal e afirmar que todo los objetos de la escena cumplían con efectos de física y colisión.

Lo que se tiene al final es una aplicación, para ejecutarse en el sistema operativo Mac OS X (preferiblemente 10.5), que ejemplifica, de manera rudimentaria, la implementación parcial de un videojuego según los rubros estipulados al principio de la tesis.

4.7. Presentación de Resultados

De todo lo anterior se resume lo que se ha producido a consecuencia del labor de implementar el videojuego.

- Un trabajo escrito que documenta de la tesis, hecho en latex y presentado en formato PDF.
- Una serie de ejecutables que visualizan los 4 escenarios.
- Una biblioteca de Física para OGRE, en formato Mac.
- Una biblioteca de Manejo de Escenas en XML para OGRE, también en formato MAC y en formato Linux.
- Una aplicación de demostración que integra a una de las escenas con un prototipo del motor de juegos.

Con excepción de la tesis, el cual se publicara en alguna pagina web, todos los demás productos (y derivados de ellos) habrán de adquirirse en la siguiente pagina de sourceforge:
http://sourceforge.net/projects/tesis-h4l9k-jp/

Capítulo 5

Conclusiones

5.1. Conclusiones y Trabajo futuro

Cuando se inicia una tesis se concluyen los estudios y a su vez se inicia un proceso de maduración. Es una experiencia, contada generación tras generación, que reafirma el compromiso de cada profesionista con el mundo actual. Por lo menos en esta tesis, se puede afirmar que ha sido un proceso de re-descubrimiento. En el transcurso de la realización de la tesis se ha fortalecido el conocimiento de la programación orientado a objetos, el dominio de al menos 3 plataformas diferentes de escritorio y se ha probado el desarrollo en algunas plataformas móviles (J2ME, iPhone y PSP). Se llego a investigar el uso de varios APIs de desarrollo multimedia y, por ende, útiles para hacer videojuegos. No obstante lo anterior no es suficiente para estarse preparado para programar vidoeojuegos.

Se partió de la idea de que la tesis se desarrollara en un lenguaje Orientado a Objetos. Se vio practico iniciar con Java para dispositivos móviles (J2ME). Poco antes de registrarse la tesis, se dio cuenta de que no era lo suficientemente maduro esta tecnología (a inicios de 2006) para poder desarrollar el juego que se tenía en mente, por lo que se cambio de plataforma y lenguaje. Junto con estos dos cambios se hizo mas ambicioso el alcance del videojuego. Ahora el videojuego contemplaría el diseño y programación de mundos en 3D. Con este cambio se tuvo que adoptar otra historia, otro modo de juego y otro interface. Incluso se podía considerar elementos de inteligencia artificial.

De antemano se supo, por los cambios mencionados anteriormente, que algunos de los objetivos originales ya no eran realizables en el tiempo dispuesto para una tesis. Esto dejo muy en claro que el juego en si seria un alcance para la posteridad. Se desarrollaría el juego en estudios posteriores e incluso se prestaría para su posible explotación comercial. Mientras que lo anterior seria el objetivo final del juego, no obstante, el objetivo final de la tesis seria el inicio de su desarrollo. La tesis fue una exploración de nuevas tecnologías y nuevos conceptos que fueron totalmente desconocidos.

Se alcanzo cubrir muchos aspectos acerca de esta tecnología de 3D en tiempo real y se investigo mucho de lo involucrado en el desarrollo de videojuegos, cumpliendo con el objetivo final. Se ha cumplido con la documentación del proyecto y se hace su entrega, a la facultad de ingeniería, por medio de este trabajo escrito.

De los Resultados obtenidos, destacan:

- Se logro dominar el uso y desarrollo de herramientas 3D a

partir del API de OGRE partiendo de la plataforma de Windows y, como consecuencia del mismo API, el dominio del lenguaje C++. El avance sobre esta linea de pensamiento, de expandir el conocimiento de la tecnología, llevo a portar el desarrollo a las plataformas Linux y luego Mac OS X.

- Se investigo a fondo las diferentes estrategias para implementar Inteligencia Artificial en un Engine basado en OGRE.
- Se desarrollo un sistema de carga de escenas funcional, basado en XML, para facilitar el desarrollo gráfico del mismo juego.
- Se porto una biblioteca de física en OGRE a la plataforma Mac OS X.
- Se creo una variedad de escenas y personajes con los cuales, junto con las tecnologías antes mencionadas y el prototipo del motor, se realizara una aplicación de demostración que se distribuirá gratuitamente a todos los interesados.
- Se hizo un intento de historia apta para un videojuego e igualmente apta para su divulgación por este medio (Lia Fail).
- Se delineo los rubros y conceptos con el cual se piensa seguir desarrollando al juego hasta su posible venta en el futuro.

Consecuentemente hubo mas desarrollos, paralelos a los anteriores, que no directamente caen dentro de los objetivos de la tesis pero que son igual de importantes en su aportación al aprendizaje. Los siguientes resultados adyacentes se anexan:

- Se investigo como se desarrollaría en consolas dedicados de videojuegos por medio del API de XNA, el SDK oficial de Playstation (1 y 2), El SDK oficial del iPhone y la no oficial del Playstation Portable (PSP).
- Se logro explorar, gracias a la cercanía que se tuvo con la DGSCA, diferentes tecnologías (OpenSceneGraph, VrJuggler, Equalizer, Virtools, PVLE, etc.) que se podrían implementar en videojuegos. Se usan actualmente en el desarrollo de aplicaciones de Realidad Virtual.
- Se hizo un intento de portar OGRE a un sistema de multidespliegue por medio de un cluster gráfico que recién se

> instalo en la Sala de Realidad Virtual Ixtli de la DGSCA, lo cual resulto muy exitoso dado que no existían precedentes para correr una aplicación OGRE en un ambiente de supercomputo (Esto se explica mas a detalle en el anexo "OGRE en un cluster", pag. 103).

A la pregunta: "¿Es pesado desarrollar un videojuego?" Se responde, sí. La industria de los videojuegos no se caracteriza por ser tolerante con participantes nuevos. No obstante este fue un paso importante para darse cuenta de los retos que se esperan. Esta *practica* en el desarrollo de videojuegos se presto para darse cuenta del nivel de programación que se requiere para poder integrarse a la industria de los videojuegos. Se concluye que es de suma importancia conocer el grado de rapidez con el cual se puede adaptar tanto a técnicas clásicas como al estado del arte en esta disciplina de la computación.

Se agradece a la facultad la oportunidad de acumular estas experiencias. Se agradece al apoyo moral invaluable que se obtuvo de los asesores. Se agradece a los amigos y familiares, así como los miembros participes de la Dirección General de Servicios de Computo Académico (DGSCA), que presenciaron el desarrollo de esta tesis. Que nunca se extingue en México la pasión por hacer realidad los sueños.

Apéndice A

Anexos

A.1. Instalación del Motor de Render Gráfico OGRE

A.1.1. Compilación del Código Fuente

Las herramientas de desarrollo para la plataforma Mac OS X, conocidas como XCode permiten utilizar un entorno integrado de desarrollo que permite obtener todas las ventajas del sistema operativo, ya que cuenta con interfaces gráficas de usuario.

El paquete del código fuente de OGRE está soportado para la plataforma Mac OS X, por lo que solo debe compilarse en el entorno de desarrollo.

A.1.2. Instalación de ODE para Mac

La manera de funcionar de la librería de render gráfico OGRE requiere que se utilicen librerías separadas para realizar tareas específicas, por ejemplo, si se necesita incluir funcionalidad de física en el sistema desarrollado, se debe utilizar un motor de física adecuado, para este proyecto de tesis se eligió el motor de física ODE por su versatilidad y el soporte con el que ya cuenta para la librería OGRE.

La librería ODE puede ser instalada en la mayoría de los sistemas operativos basados en UNIX, por lo que puede ser instalado de la forma usual para el sistema Mac OS X, sin embargo, para ser utilizada en programas desarrollados con el entorno de desarrollo de Mac OS X, y poder ser ejecutado tanto en procesadores Intel o Power PC, las librerías deben estar disponibles en un formato especial, conocido como *framework*, en el cual se empaquetan las funciones de las librerías.

Para ese proyecto de tesis se utilizó la librería ODE y un módulo adicional que da soporte de funciones de física al motor de render gráfico OGRE. Este módulo está disponible originalmente para el sistema operativo Windows, por lo que fue necesario adaptar este módulo para poder ser usado en la plataforma Mac OS X.

A continuación se describen las acciones que se llevaron a cabo para realizar esta adaptación del módulo.

A.1.3. Preparación del Código Fuente de ODE para la Plataforma Mac

El primer paso que se siguió fue descargar el código fuente de la versión estable de la librería del sitio oficial del motor de física ODE [http://www.ode.org/], que en este caso fue la versión 0.10.1.

Desde la ubicación donde se descargó el código fuente, se descomprime y se agregan los directorios del paquete del código fuente al programa de desarrollo de Apple Xcode, creando un proyecto a partir de estos archivos, cuyo objetivo es una carpeta especial en formato *framework* de desarrollo

Al compilar este código se obtiene el *framework* necesario para poder continuar con la adaptación del módulo de soporte de la librería ODE para ser usada por OGRE.

A.2. Borrador de Lia Fail

A.2.1. Contexto Histórico

Figura A.1: Irlanda en la época mitológica

Lo que sigue es un resumen de la historia del juego, basado en lo que se conoce de la mitología irlandesa (en particular el *Lebor Gabála Érenn* o libro de la toma de Irlanda) antes de la llegada de los romanos. En Roma Cayo Julio Cesar Augusto Germanico (Caligula) acaba de ascender al trono imperial. Caligula sospechaba de algunos miembros de la guardia pretoriana por lo que manda a matar a esos miembros. Un soldado en particular, Octavius Summavigilo, es un pretoriano quien al haber concluido su estancia en Palestina, emprende una campaña de exploración a la desconocida isla de Hibernia (hoy Irlanda), por ordenes del emperador.

Octavius había formado parte del *cohors praetoria*, por lo que había jurado fidelidad al emperador anterior, Tiberio. Tenia un estricto código de honor y sufría de terribles pesadillas debido a que fue testigo de las crueldades que estaban sucediendo en Jerusalén. También no se acuerda de su juventud en Asturias, sufriendo una especie de amnesia. Lo único que sabe es de un nombre, Remiel, que posiblemente fue el nombre de algún familiar.

Hibernia es la ultima frontera de los celtas, quienes habitaban Europa occidental años antes de que llegaran los romanos a poblarla (figura A.1). Se sabe muy poco de la isla antes de su conversión en los últimos años del imperio...y mucho menos se sabe del ultimo paradero de Octavius.

A.2.2. Lia Fail

El juego inicia con la llegada de un buque romano a la costa de Hibernia. Lo observa, desde la costa, un nativo anciano cuya identidad desconocemos. Vemos a Octavius sonriendo y viendo hacia los montes en el horizonte. Se llena su corazón de paz al suspirar y contemplar la visión de las montañas. No muy lejos, en la base de esas montañas, se ve un pueblo – *Laigain*. En el pueblo de Laigain, dentro de una choza, están dos niños y un hombre adulto de unos 30 años de edad. Uno de los niños, Pip, esta jugando con el otro niño, Lorke, y le menciona al hombre mayor que parte ahora a jugar

con Leslie, su amiga desde la infancia. El hombre, con cierto aire noble y de vestimenta algo regio, se dirige a Pip avisándole de que no se vaya muy lejos de la aldea.

Pip le responde: "No te preocupes *Elim*, sé como cuidarme."

Encontramos a Pip jugando en el jardín de Leslie, cuando llegan los legionarios romanos a la aldea. Sale de su choza el misterioso Elim en representación del pueblo para averiguar de que se trata la inesperada visita. Conversa con Octavius, Pretoriano y jefe de la expedición, de antemano dándole la bienvenida a Laigain y a los dominios de su alteza Fiacha Finnfolaidh. Elim explica a Octavius que él es un especie de gobernador de las áreas alrededores. Octavius le presenta a Elim su amigo de la infancia, Pomponius Mela, quien tiene el encargado de tomar observaciones geográficos de la isla. Se separa Elim dialogando con Mela, mientras que los legionarios y Octavius descansan en Laigain.

Pip y Leslie están jugando en el bosque detras de la aldea. De repente encuentran a una niña envuelto en tiras. A la primera la niña tiene miedo a los dos, pero la calma Leslie. Revela que se llama Raquel y que ha estado huyendo de los soldados del rey. Justo en ese momento llegan unos soldados de armadura negros. Intentan huir los tres, escondiéndose entre los arbustos, pero alcanzan raptar a Leslie. Pip corre de regreso al pueblo con Raquel. Leslie, ya rehén, dirige a estos soldados al pueblo de Laigain. Andan los soldados en busca del *Lia Fáil*. Llega Pip a la aldea y conoce de paso a Octavius, pero llegan los soldados negros y se identifican al pueblo como centinelas de las tribus del norte y andan en busca de la piedra de Fal. Los aldeanos se los confrontan con la ayuda de los legionarios recién llegados, pero son muchos los soldados y acaban masacrando a muchos del pueblo y a casi todos los legionarios. Se muere Lorke en esta confrontación. Los soldados traen una espada mágica, Fragarach, que puede partir a cualquier objeto en dos.

Sobrevive Octavius y reúne a los aldeanos, incluyendo a Pip y Raquel, para formar un pequeño batallón para rescatar a Leslie, recuperar la piedra de Fal y vengar a todos los fallidos en la emboscada. Octavius y compañía emprenden su aventura.

A lo largo del juego, este pequeño grupo va de pueblo en pueblo, acabando poco a poco con los soldados. En el camino Octavius conoce a una princesa que trata de escapar de Hibernia, llamada Eithne. Hay un pequeño romance entre ellos antes de que separen caminos[1]. En algún momento Pip

[1]Eithne Imgel es la madre del futuro rey Tuathal Techtmar. Eithne esta embarazada antes de refugiarse en con su padre el rey de Alba (Bretaña)

se siente desahuciado, al no sentir que encontraran pronto a Leslie. Octavius lo reconforta contándole acerca el amor y la persistencia. Pip se vuelve a animar y Octavius le regala una piedra verde que siempre lo había acompañado: Zollverein. Llegan a una fortaleza donde encuentran a uno de los jefes, el que maneja la espada de Fragarach. Octavius se muere en el enfrentamiento con este jefe y le sucede Pip. Octavius, entre sus últimas palabras, reclama a Pip con el nombre romano de Mauricius Summavigilo, para que continuara con la misión. Equipado con Fragarach y la piedra verde, va en busca de Leslie y la piedra de Fal.

Simultáneamente con las aventuras de nuestros heroes, Elim y Mela están reuniendo a la gente de los pueblos para afrontarse al rey de Hibernia, porque sospechan que esta en liga con un banshee. No saben que también es esta unión de poderes quien dirige a los soldados negros. En dos ocasiones se reúnen con Octavius y después una ultima vez con Pip.

Pip descubre, al vengarse y rescatar a Leslie, que la piedra de Fal, Lia Fáil, es la llave al"Otro Mundo" y que falta reunir otros dos tesoros para poder llegar al "Otro Mundo", cumpliendo con una antigua profecía. Debe reunir los 4 tesoros de Tuatha Dé Danann[2], de los cuales, 2 de ellos son la espada de Fragarach y Lia Fáil. Raquel, al enterrarse de esto, le ruega a Pip y Lenna que no sigan en busca del "Otro Mundo", ya que pone en riesgo sus vidas y ella no las quiere perder. Pip entonces le pregunta a Raquel que se debe hacer y los dirige a un pozo encantado donde descansa un espiritu del "Otro Mundo".

En el pozo encantado se encuentran con el espíritu de Elia (Eriu), cerca de unas cascadas. Elia calma a las inquietudes de Raquel y le dice a Pip que es justo que él y Leslie vayan a Tir Na n'Og. Elia se sorprende al encontrar que Pip sostiene la piedra verde y le dice a Pip que Octavius le ha heredado un precioso regalo y que por lo mismo no debe temer nada, dado que por medio de la piedra verde siempre lo estará protegiendo Octavius. Se revela que la piedra Zollverein no solo protege a quien la posee, sino también retiene los recuerdos y emociones de quien la guarda, y por lo tanto puede convocar al espíritu de esa persona a luchar por ellos; es de la voluntad de Dagda[3], el dios bueno, que Pip también se beneficie de la magia de la piedra. Cuando parten Pip, Leslie y Raquel, Elia reza a Dagda, suplicando que los proteja muy bien, y que también proteja a su "hijo".

Acaba Pip acumulando los dos últimos tesoros y se enfrenta al Banshee

[2]El tribu de seres divinos que ocuparon Irlanda antes de la llegada de los Milesianos/Celtas

[3]Rey mítico de los Tuatha Dé Danann que también era el Dios padre de los celtas irlandeses

llamado "Raemhiel", espíritu protector del linaje de los Milesianos. Pip convoca al espíritu de Octavius por medio de Zollverein y inicia la batalla final contro Raemhiel. Raemhiel es derrotado y huye. Antes de que desaparece, ve la piedra verde que sostiene Pip y murmura "...muy listo de tu parte, Rhaziel...".

Finalmente Elim derroca al rey Fiacha Finnfolaidh con ayuda de Pip y Leslie. Se proclama rey, pero no sin antes despedirse de Pip y Leslie, revelando que ellos son los elegidos de los Tuatha Dé Danann para proteger la memoria de la isla y que él los ha estado cuidando desde que nacieron para que pudieran cumplir con sus destinos. También se despide Elim de Mela, pidiéndole que al regresar a Roma puede mencionar a la isla de Hibernia, pero que por el bien del reino no mencione al pueblo ni su historia.

Raquel ahora revela a Pip y Leslie que ella es "Lia Fail", porque la encontraron como una piedra color rosa y se transformo ante el rey Fiacha Finnfolaidh. Raemhiel había aconsejado al rey tomarla prisionera y usarla como arma. No se acordaba como es que llego a Hibernia pero revela que tampoco fue coincidencia la presencia de Octavius, Rhaemhiel y Zollverein en la isla - todos ellos y ella estaban conectados de alguna manera. Ella toma en mano a la piedra Zollverein y le dice: "Es hora de regresar a casa mi pequeño"

Al final del juego vemos a Pip, Leslie y Raquel atravesando el lago Derravaragh sobre un caballo blanco (Epona), con Zollverein en mano. Brillan Raquel y Zolverein y juntos abren un portal a la tierra de Tir Na n'Og. A lo lejos se ven dos siluetas observando que Pip y los demás entran al portal. Entre los dos siluetas se alcanza identificar a Raemhiel. Le dice Raemhiel a la otra figura:"Este ciclo, al menos para nosotros, ha llegado a su fin. Ya nos tenemos de partir, Rama". Desvanecen ambos.

Ya entre los Sidhe (Tuatha Dé Danann) en Tir Na n'Og, Pip, Leslie y Raquel jamás envejecen, guardandose la memoria de los Milesianos, Octavius y los legionarios, bajo la protección del espíritu de Elia. El pasar del tiempo en el "Otro Mundo" es distinto al del mundo humano por lo que Pip y Leslie jamás podrán regresar hasta que se acabe el mundo humano.

Finalmente, en el epilogo del juego, vemos a un árbol solitario, situado arriba de un acantilado, viendo hacia unos montes que circundan unas praderas verdes con un rió en medio. Al ver la cara del árbol que da hacia los montes, vemos una mano que pasa encima de las siglas Iota y Pi que acaba de escarbarse en el. Vemos al espíritu de Octavius diciendo a si mismo:"Acaba un ciclo e inicia uno nuevo...ya pronto Lenna, nos volveremos a encontrar". Se oye a lo lejos una voz que llama a Octavius:"Ya Ramma, deja de jugar ahí, hay que irnos". Voltea Octavius una vez mas hacia los montañas y di-

ce:"Me toca recordar a mi, Rachel...cuidaste muy bien de Mauricius y Leslie y ellos a ti...siempre te amare"

Explicación del Epilogo

Lo ultimo en esta historia es para dar un sentido de continuidad, ya que Lia Fail pretende ser un capitulo intermedio en una serie de videojuegos. No se revelara como es que se conectan estos elementos, o al menos no se pretende, en este trabajo de tesis.

A.3. OGRE en un Cluster/Cumulo Gráfico y con Despliegue Estéreo-Gráfico

Entre los experimentos que se hizo para desarrollar aplicaciones para cumulo gráfico, en la Dirección General de Servicios de Computo Académico, se tenia la re-implementación del ciclo de dibujo de OGRE para despliegue estéreo-gráfico. Ya existía un antecedente en el departamento de Realidad Virtual pero fue implementado en una versión anterior de OGRE (para el cual ya no estaba apto la tesis). Cayo la responsabilidad de actualizar tal implementación a uno de los becarios del departamento, quien casualmente es miembro de esta tesis.

Se intento atacar el problema a través de las soluciones existentes para cúmulos gráficos: Chromium y Xdmx para X/OpenGL distribuido; y Equalizer. Ambas presentaban problemas para atacar el problema de despliegue estéreo-gráfico.:

Chromium es una re-implementación de OpenGL. De manera simple se ve como una capa intermedia entre la aplicación y la librería libGL. Para poder capturar comandos de OpenGL y distribuirlos a los diferentes nodos en un cluster, engaña a la aplicación - por eso la necesidad de re-implementar OpenGL. Al final distribuye la tarea de dibujo a los diferentes nodos según la configuración que se hizo.

En el proceso de adaptar OGRE a usarse en Chromium, se dio cuenta de la inestabilidad de OGRE en OpenGL - Se veía bien mientras se usaba un solo monitor de cualquier tamaño, pero no tan bien cuando se esparcían los datos entre diferentes framebuffers. Agregase la complejidad de forzar un modo estéreo-gráfico por medio del mismo Chromium y tenias resultados muy desagradables y a veces aleatorio.

Un tutorial de como correr OGRE en un cluster o cúmulo gráfico y sus resultados se encuentra en el siguiente wiki de OGRE (escrito por el mismo autor de este pasaje). Solamente se encuentra en ingles por el momento.

http://www.ogre3d.org/wiki/index.php/Ogre_on_a_Cluster

Equalizer se presenta como un API poco invasivo y hecho para aplicaciones en cúmulos gráficos. Aquí la única problema consistía en el tener que adaptar al programa a usarlo. Estaba pensado para hacer aplicaciones partiendo de OpenGL o GLUT, no tanto un API encima de estos. Aun así, un estudiante de la universidad de Beijing logro hacer un ejemplo combinando OGRE y Equalizer. Resulto que su implementación no siempre funcionaba y no siempre se podía compilar. Era una solución que ofrecía muchas ventajas, incluso la adaptación para despliegue estéreo-gráfico, pero a su vez

representaba una considerable inversión de tiempo.

Tiempo después se encontró una solución (a medias) que permitía el despliegue impecable de gráficos 3D a cualquier despliegue X o VNC: VirtualGL. En teoría soportaba despliegue estéreo-gráfico, pero solo si la aplicación lo tiene. Había un solo problema: Todo el calculo se generaba en el nodo maestro. VirtualGL fue diseñado pensando en el 'Rendering remoto' (esto es, se dibujaba en el servidor y se mandaba una imagen jpeg por la red 60 o 120 veces por segundo) y no precisamente para salas de Realidad Virtual. No se tenia tanta potencia en un solo nodo, para cuestiones de estéreo y a la resolución de la sala Ixtli (lo cual llegaba a 3520x1024), pero si para una imagen sin estéreo (aun así, contando con una ancho de banda de 10Gb/s, los resultados eran mediocres). A 11 (a veces 25) cuadros por segundo, sin estéreo-grafía y sin rapidez de interacción (se trababa el teclado a veces), VirtualGL resulto ser una opción poco viable.

Para lo anterior se tuvo que actualizar, oficialmente, el código para despliegue en estéreo de OGRE, el cual tuvo éxito pero no funcionaba con VirtualGL (curiosamente si con Chromium). A pesar del fracaso de usar VirtualGL, se agregaron nuevas soluciones a diferentes problemas (como mostrar GoogleEarth en el cluster) y se amplio el engine de OGRE para usarse en ambientes Virtuales (Una exitosa colaboración entre la comunidad de OGRE y el Departamento de Realidad Virtual). En particular el proyecto sigue en camino, ahora con énfasis en su uso con Equalizer. Con respecto al juego de lia fael - hace posible la creación de una versión del juego para ambientes virtuales.

El parche, a la hora de escribir, para OGRE 1.49 se encuentra en la siguiente pagina:

http://imagen.ixtli.unam.mx/~jp/dist/Ogre_1.49_StereoGL_Patch.tar.bz2

A.4. Historia de la programación y su importancia en los videojuegos

Comúnmente se pierde de vista, al desarrollar un videojuego o una aplicación compleja, es la cuestión de como se podría hacer mas rápido la ejecución de éste o de que manera se puede hacer mas sin desperdiciar recursos. Existe la opinión valida de empezar con un lenguaje de alto nivel para conocer los rubros básicos de la programación de videojuegos, pero conforme se empieza a dominar lo básico, los desarrolladores tenderán a implementar algoritmos mas complejos, por ende, incrementando la cantidad de operaciones por cada ciclo en el juego. Una solución es optimizar los algoritmos implementados y otra, a menudo obligatorio, es bajarse de nivel en el lenguaje de programación usado para poder aprovechar al máximo los recursos de la maquina.

En un principio las primeras computadoras digitales (ENIAC o el Manchester Mark I) se programaban interconectando bulbos o transistores para hacer compuertas lógicas. Las compuertas lógicas son la base para armar, de manera física, algoritmos basados en matemática booleana (posteriormente se eliminó el uso de compuertas lógicas en algunos aparatos comunes como memorias flip-flops, optando mejor por pura lógica transistor-transistor). La programación a este nivel resulta ser una inversión considerable de tiempo, razón por la cual la matemática Grace Hopper, al estar trabajando con el UNIVAC I en la década de los 50, decidió inventar los primeros mnemónicos para simplificar la programación de algoritmos frecuentemente usados. El resultado fue el lenguaje A-0, un antecesor del lenguaje ensamblador. En sí las instrucciones en ensamblador son una secuencia de bits que, traducidos por una unidad lógica, llevan a cabo los diferentes pasos que conforman cada uno de esas instrucciones. A estos pasos se les conocen como instrucciones de Micro-Código (el nivel mas bajo posible de programación en un microprocesador). Este paso, de automatizar las funciones mas intimas de una computadora, seria el principio de una tendencia que determinaría por completo la historia evolutiva de la computación - la creación de maquinas y lenguajes que acomodan mejor al modelo de pensamiento del ser humano.

A pesar de esta tendencia todavía prevalece la regla: para aprovechar mejor una maquina, el programador debe entender mejor a la maquina, no la maquina al programador.

Tomando al micro-código como ejemplo, bajo ciertas circunstancias, se ha tenido que re-definir los comandos de ensamblador para optimizar el rendimiento de los procesos llevados a cabo en un videojuego. Tal es el caso

del procesador de gráficos del Nintendo 64, el RCP de SGI. El RCP tenia la posibilidad de ser modificado en tiempo de arranque y muchas compañías (RareWare, LucasArts, etc.) aprovecharon esta flexibilidad para introducir mejoras y optimizaciones al pipeline gráfico del Nintendo 64.

Siguiendo con la tendencia establecida al ensamblador también se le encimo una capa intermedia para facilitar la comunicación con el programador, esto es, se inventaron los lenguajes de alto nivel. Con estos lenguajes sucedió una explosión de la complejidad en el compilador (el cual había sido un simple traductor a código binario). La compilación se volvió un proceso de múltiples etapas, esto debido a que los lenguajes de alto nivel, al intentar aproximar al lenguaje natural del ser humano, resultaron ser difíciles de interpretar para la computadora (pero no imposible, e.g. LISP). Los lenguajes de alto nivel llevaban un sintaxis y una conjunto enorme de palabras reservadas en su vocabulario. Posiblemente de estos lenguajes, como uno de los pasos del compilador, se podían generar secuencias intermedias de programas en lenguaje ensamblador pero algunos también se podían traducir directamente a secuencias binaria (Lo que hoy seria equivalente a archivos ejecutables). Así surgió BASIC, FORTRAN, COBOL y muchos otros lenguajes. Estos lenguajes tuvieron la ventaja de ser muy fáciles de entender y muy útiles para programar algoritmos mas complejos, a cambio de una probable perdida de velocidad y eficiencia, por lo que esto llevó a la creación de maquinas mas grandes con múltiples unidades lógicas (o incluso múltiples procesadores). Al tener maquinas con mas registros, unidades aritméticos y mas procesadores se presentó una nueva problemática para los que fabricaban computadoras - como hacer mas eficiente el manejo de datos para realmente aprovechar la vasta cantidad de recursos. Fue entonces que se creo la memoria virtual, el procesamiento vectorial y la distribución de tareas. Los que implementaban lenguajes intentaron sacar provecho de estos nuevos conceptos, pero los mismos programadores en lenguajes de alto nivel no tenían manera de optimizar sus códigos, usando estos conceptos, a menos que programaban en ensamblador. Tanto del lado de los fabricantes como del lado de los desarrolladores se habían comprometido hasta cierto grado, pero lo solución a esta brecha en la eficiencia llegaría con los lenguajes de bajo nivel.

LISP nació como un intento de reducir la programación a lógica pura, facilitando la investigación de la inteligencia artificial. De LISP otros científicos se inspiraron a crear el lenguaje Prolog, el cual era mas apta para avanzar el campo de la inteligencia artificial. Finalmente, de los gustos de un ingeniero por simplificarlo todo, salió APL. Estos lenguajes se volvían mas y mas crípticos en su sintaxis, a tal grado que se podía reducir una

pagina entera de operaciones en otro lenguaje (ej. COBOL) a una sola linea (APL). Lo anterior era mucho mas fácil de optimizar para la computadora y reducía el tiempo de compilación debido a que se seguía una lógica mas consistente para la maquina.

Ahora la tendencia evolutiva se había vuelto un compromiso, entre una lógica mas estricta del pensamiento humano y un interpretador mas poderosa para la maquina. Los lenguajes de bajo nivel tuvieron las mismas facilidades que los de alto nivel para implementar algoritmos complejos, con velocidades de procesamiento muy cercanas a programas hechos en ensamblador, a cambio de un sintaxis mucho mas rígido y formas de pensar mas ordenadas.

También de este nuevo compromiso surgieron sucesivas adaptaciones del lenguaje ALGOL. Eventualmente estos descendientes de ALGOL (ALGOL 60, CPL, BCPL, B) convergerían en C. C presentaba casi todas las características minimalistas de APL, con la distinción particular de ser el lenguaje sobre la cual se desarrollaría uno de los sistemas operativos mas poderosos y populares de la historia de la computación - UNIX. También por la época, como resultado de la frustrada lucha por mantener a ALGOL como el lenguaje de propósito general de facto, nació otro lenguaje, tan elegante como C y mucho mas fácil de entender...Pascal.

Fue a estas altura de la historia (en la década de los 60's) cuando nacieron los primeros videojuegos. Un precavido y silencioso desarrollo que tenia solo dos metas: ser rápido y ser divertido. UNIX nació de un videojuego y como consecuencia, tenia a su ventaja la habilidad de responder casi siempre en tiempo real. Los videojuegos, por su lado, empezaron muy lentos en la adaptación de nuevos lenguajes. Al ser necesariamente programas que interactuarban en tiempo real, los programadores sacrificarían todo familiaridad y toda facilidad con la maquina. Desplazados casi 30 años con respecto a otras aplicaciones con fines científicas y de negocios, los videojuegos empezarían su humilde trayectoria con la programación en el hardware - y no necesariamente fue hardware digital: *Tennis for two* fue hecho en una computadora analógica; *Space War!* fue programado en el lenguaje ensamblador de una minicomputadora DEC; Pong fue hecho a base de lógica de compuertas y LEDs.

No fue hasta el auge de la consola de ATARI, que los videojuegos se programarían totalmente en ensamblador. Esta tendencia se seguiría hasta los últimos días de la consola de Nintendo (NES). La programación en C empezaría con las consolas de 16 bits y siendo aun maquinas con recursos limitados, se consideraría un lujo debido al cuidado que se tenia que tomar al no desperdiciar memoria. La programación de videojuegos raramente tu-

vo lugar en lenguajes de alto nivel y mucho menos en lenguajes orientado a objetos, practica que solo se dio en las computadoras personales a mediado de la década de los ochentas. La programación Orientado a Objetos en las consolas de videojuego se daría por primera vez con las consolas de la generación del Playstation 2. Hoy en día, la programación Orientado a Objetos en las consolas, aun se considera un lujo.

Pero porque era un lujo?

Las consolas de videojuegos se caracterizan por ser justos en la cantidad de memoria física que tienen disponible para los juegos. Se debe a que no se tiene contemplado la ejecución de mas de una aplicación (es decir mas de un juego) a la vez. Sí tienen la capacidad para ejecutar varias tareas en paralelo y es por eso que los desarrolladores son los encargados de cuidar cuantos procesos se lanzan y como se maneja la memoria. Programar videojuegos es una tarea que tiene muchos paralelos con programar un sistema operativo debido a estos compromisos con el hardware.

Lógicamente los sistemas operativos, a nivel de kernel, también se ven forzados a programarse en lenguajes de bajo nivel. Raramente se usa, por ejemplo, C++ en el desarrollo del kernel UNIX o Linux (o en el caso de XNU, C++ y C Objetivo). Aun mas rara es la existencia de sistemas operativos que fueron hechos totalmente en en un lenguaje Orientado a Objetos (e.g. JNode esta programado totalmente en Java). Los lenguajes de alto nivel casi nunca están optimizados para el hardware y los de medio y bajo nivel no siempre garantizan los algoritmos mas óptimos en su compilación. Los lenguajes orientados a objetos tienen la desventaja de no siempre ser lo mas óptimo en el uso de memoria. Los lenguajes Orientado a Objetos que usan maquinas virtuales (Java, C# y SmallTalk) tienen una desventaja mas que es la falta de comunicación directa al hardware.

Porque entonces se crearon los lenguajes Orientados a Objetos? Los lenguajes orientado a objetos nacieron con la creación del lenguaje Simula. Basada en Algol 60, presentó muchas innovaciones a la programación con objetos, clases, subclases, métodos virtuales y el famoso garbage collector (entre otras cosas). Simula fue pensado para hacer simulaciones numéricas complejas, por lo tanto, se necesitaba de herramientas mas poderosas - los objetos. Los objetos representaban una manera alterna de resolver problemas partiendo de la sintaxis de los lenguajes estructurados de bajo nivel. No obstante no representaban una evolución de los lenguajes de bajo nivel, ni tampoco de los lenguajes de alto nivel, sino una evolución del paradigma que se había estado manejando desde la época de Grace Hopper - la programación estructurada.

Implementar objetos en C no era imposible, pero si presentaba un reto

de logística a quienes los querían implementar, razón que llevo a Bjarne Stroustrup (programador de AT&T) a desarrollar "C con Objetos" o lo que hoy se conoce como C++.

En la Programación Orientado a Objetos existen dos escuelas de implementación: Simula/C++ y SmallTalk/*C Objetivo*. Ambas implementan tanto clases y objetos, pero la idea de como se invocan son muy diferentes. En C++ (como en Simula) el programador *llama* a un método que se conoce de algún objeto. En C Objetivo (y SmallTalk), el programador manda un *mensaje* al objeto y el objeto responde a ese mensaje como pueda, de esta manera puedes pedir del objeto algún método aunque éste no lo contenga, porque realmente no se esta llamando, se esta enviando un mensaje. C Objetivo tiene en común con C++ en que también desciende de C, pero nada mas. C Objetivo es hoy en día el corazón del API de Cocoa (de Apple) mientras que C++ es mas popular entre las plataformas UNIX y Windows (debido a su antigüedad). Ambos son lenguajes soportados por GCC.

Como se mencionaba, todos estos lenguajes han llegado a usarse en videojuegos para computadoras personales. El problema de su uso en las consolas reside en las limitaciones del hardware de las herramientas de desarrollo. En generaciones anteriores de consolas de videojuegos, lo anterior hacia la tarea de portar juegos desde la computadora personal a la consola mas difícil. En las generaciones actuales de consolas, ya no importa tanto si se llega a usar programación orientado a objetos - siempre y cuando se evita el desperdicio de memoria. No obstante existen plataformas en donde nada mas se pueden desarrollar en lenguajes Orientado a Objetos. Tal es el caso del iPhone de Apple en donde nada mas se puede programar en C/C Objetivo. Las PCs de mano basados en Palm OS y el API de Brew (para celulares cdma2000) se programan en C/C++. J2ME, Java FX y la plataforma Android (Google) se programan totalmente en Java.

Se puede inferir entonces que en un futuro se va poder programar en las consolas desde lenguajes que ahorita tienen su auge en la programación de paginas web - lenguajes que ya no se necesitan compilar. Tal vez por los avances en la tecnología de interfaces, se llega a eliminar la necesidad de programar usando códigos fuentes y compiladores, pero se puede tener por seguro que esto se tardara en llegar a las consolas de videojuegos.

Bibliografía

[1] Multics.
http://stuff.mit.edu/afs/athena/reference/multics-history/
http://www.multicians.org/myths.html#source.

[2] Script creation utility for maniac mansion(scumm).
http://alban.dotsec.net/7.html
http://en.wikipedia.org/wiki/Scumm.

[3] Sierra creative interpreter (sci).
http://agisci.classicgaming.gamespy.com/
http://en.wikipedia.org/wiki/Sierra's_Creative_Interpreter.

[4] Space travel.
http://cm.bell-labs.com/cm/cs/who/dmr/spacetravel.html
http://www.fas.harvard.edu/~lib215/reference/history/spacetravel.html
http://www.livinginternet.com/i/iw_unix_dev.htm
http://www.bellevuelinux.org/thompson.html
http://www.cs.bell-labs.com/who/dmr/spacetravel.html.

[5] Spacewar!
http://www.thedoteaters.com/p2_stage1.php
http://www.digisys.net/users/cogs/spacewar.htm
http://pdp-1.computerhistory.org/pdp-1/index.php?f=theme&s=4&ss=3.

[6] Unix, ken thompson, dennis ritchie.
http://en.wikipedia.org/wiki/Dennis_Ritchie
http://en.wikipedia.org/wiki/Ken_Thompson
http://en.wikipedia.org/wiki/UNIX.

[7] *Computer Images*. Understanding Computers. Time-Life Books Inc., Alexandria, Virginia, second edition, 1988.

[8] *Computer Languages*. Understanding Computers. Time-Life Books Inc., Alexandria, Virginia, second edition, 1988.

[9] Alex J. Champandard. Aigamedev. http://aigamedev.com.

[10] Darryl Charles, Colin Fyfe, Daniel Livingstone, and Stephen McGlinchey. *Biologically Inspired Artificial Intelligence for Computer Games*. IGI Publishing, 2008.

[11] Bill Gates. "what the hell does bill gates know about videogames anyways". *Next Generation*, pages 60,61,62, June 1996.

[12] David Harel. Statecharts: A visual formalism for complex systems*. *Science of Computer Programming 8*, pages 231–274, 1987.

[13] Richard S. Wright Jr. and Benjamin Lipchak. *OpenGL SuperBible*. Sams Publishing, third edition, 2005.

[14] Gregory Junker. *Pro OGRE 3D Programming*. Apress, first edition, September 2006.

[15] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, 1988.

[16] Wikipedia. Adventure game interpreter (agi).
http://www.justadventure.com/articles/Engines/AGI/AGI.shtm
http://en.wikipedia.org/wiki/Adventure_Game_Interpreter.

[17] Wikipedia. Conocimiento. http://es.wikipedia.org/wiki/Conocimiento.

[18] Wikipedia. Simula. http://en.wikipedia.org/wiki/Simula.

[19] Wikipedia. Super mario 64. http://en.wikipedia.org/wiki/Super_Mario_64.

[20] Wikipedia. Tennis for two. http://en.wikipedia.org/wiki/Tennis_For_Two.

[21] Wikipedia. Unrealscript.
http://udn.epicgames.com/Three/UnrealScriptReference.html
http://unreal.epicgames.com/UnrealScript.htm
http://wiki.beyondunreal.com/Legacy:UnrealScript_Language_Reference/Introduction

http://wiki.beyondunreal.com/Legacy:UnrealScript_Lessons
http://wiki.beyondunreal.com/Legacy:UnrealScript
http://en.wikipedia.org/wiki/UnrealScript.

Índice de figuras