CAPITULO I.

INTRODUCCIÓN

1.1 Introducción

La industria de los videojuegos existe desde hace poco tiempo, y con el pasar de los años se ha consolidado como una de las industrias más activas y que más ganancias recibe, esto ha determinado que se entregue productos de muy buena calidad.

En Ecuador el mercado de videojuegos está copado por software importado de este tipo, el desarrollo de videojuegos es casi nulo, convirtiéndonos así en un actor pasivo de esta actividad económica. Esta industria seguirá creciendo, y con el antecedente anterior, sería una buena opción participar activamente con productos nacionales de calidad.

El presente proyecto pretende documentar el desarrollo de un videojuego desde la concepción de la idea, hasta la culminación del mismo, cumpliendo todos los objetivos propuestos. El desarrollo de software de este tipo requiere conocimientos especializados de varias disciplinas como por ejemplo conocimientos avanzados de programación, arquitectura y diseño de software, redes de computadores, inteligencia artificial, diseño gráfico, creación de audio y video, entre otras, que a lo largo del desarrollo de este documento se lo ira detallando. Debido a la complejidad que se afrontó en este proyecto el resultado final no es el de un videojuego profesional, realizado por compañías dedicadas a esta actividad.

1.2 Tema del proyecto

Desarrollo de un videojuego con texturas y modelos 3D, multiusuario (en red), en un entorno cliente/servidor (internet) utilizando el framework de Microsoft XNA.

1.3 Problema

1.3.1 Antecedentes

A nivel mundial, grandes compañías se han dedicado al desarrollo de videojuegos para computador y otras consolas como XBOX, PS2 entre otras, estableciendo un mercado de consumidores considerablemente alto, generando así ganancias significativas a los fabricantes de estos.

Los videojuegos no son un mal hábito para las personas, sino, son una fuente de entretenimiento, además de conseguir el máximo rendimiento de los componentes del computador donde se implantará el software, y según la calidad del mismo, el usuario estará interesado y con interés en probar la aplicación.

Algunas simulaciones computacionales realizadas sobre sucesos o eventos reales que ocurren, han terminado por ser videojuegos como los simuladores de vuelo, pistas de autos, etc. Representando así una realidad virtual en la cual se simulan eventos que pueden ocurrir en la realidad.

El proceso de la producción de videojuegos casi no difiere de la producción de software en general, pero un videojuego de computador tiene más componentes que un software

que cumpla alguna actividad, como audio, el sonido propio para cada etapa o escena del videojuego, si el videojuego tiene historia, este será en base a esta, el diseño de personajes, todo videojuego tiene personajes propios nacidos de la invención de los desarrolladores del mismo en el caso de que no tenga una historia real, niveles del videojuego donde se establece la dificultad o diferentes escenarios donde se desenvuelve la aplicación, y demás componentes, que hacen del desarrollo de videojuegos de computador un conjunto de habilidades que debe poseer el desarrollador de software de entretenimiento.

El desarrollo también depende de la plataforma para la cual se quiera implementar el videojuego, PC, móviles, consolas, ahora con la tecnología a utilizar se puede conseguir la funcionalidad tanto para plataformas PC (S.O. Windows de Microsoft), XBOX 360(Consola desarrollada por Microsoft) y Zune (Dispositivo de multimedia desarrollado por Microsoft).

1.3.2 Situación actual

Nuestro país es netamente consumidor de videojuegos de computador y otras plataformas, la producción de este tipo de software es escasa en el mercado nacional, sin embargo, a nivel nacional está en auge el desarrollo de software corporativo y a la medida para empresas que necesiten automatizar algún proceso, pero con este crecimiento de ofertas de software, también aumenta la competitividad entre profesionales en desarrollo de sistemas.

A nivel nacional, en casi todas las Universidades o Institutos donde se puede obtener un título como profesional en el desarrollo de sistemas o afines, se capacita a las personas en generar software para satisfacer las necesidades de las empresas u otras problemas que pueden ser resueltos con la asistencia del computador, sin embargo, con el antecedente citado anteriormente: el mercado de consumidores de videojuegos para computador, es uno de las más grandes a escala mundial; esto establece otra opción para profesionales en desarrollo de software con una buena perspectiva de desarrollo económico alternativo.

1.3.3 Planteamiento del problema

En nuestro país y provincia, el desarrollo de videojuegos es casi nulo, esto puede notarse fácilmente ya que ningún producto de este tipo es comercialmente reconocido cuya procedencia sea local, el consumismo de productos extranjeros copan el mercado, lo cual nos convierte en actores pasivos de esta industria, este problema que puede deberse a varios factores como:

- La escasa capacitación en el uso de tecnologías para el desarrollo de videojuegos.
- Complejidad en el desarrollo de videojuegos puesto que este tipo de software incluye muchos componentes que deben ser propios de cada videojuego como sonido, historia, texturas, etc.

- Falta de interés por parte de los desarrolladores de software para abrir nuevas opciones que se puede explotar y generar una fuente de ingresos.
- Creencia de que los videojuegos son malos para la sociedad y que no representan utilidad para la misma.

1.4 Objetivos

1.4.1 Objetivo General

Desarrollar videojuegos con modelos 3D para un jugador, multiusuario (en red), en un entorno cliente/servidor (internet).

1.4.2 Objetivos Específicos

- Presentar una investigación sobre la tecnología de Microsoft para desarrollar videojuegos XNA.
- Desarrollar un videojuego en que se utilice esta tecnología.
- Describir estándares de calidad para sistemas de entretenimiento.
- Aprovechar al máximo las funcionalidades que ofrece el API de XNA en la creación de videojuegos.
- Establecer una nueva visión en el campo laboral para los desarrolladores de software.
- Definir las ventajas y desventajas de utilizar esta tecnología.
- Con la investigación a realizarse presentar una guía sobre la creación de videojuegos utilizando esta tecnología.

1.5 Alcance

- El software a desarrollarse podrá ser implementado sobre consolas PC, Xbox 360, Zune, propiedad de Microsoft.
- El videojuego será probado solo sobre PC debido al costo de adquirir las consolas que lo soportan.
- El sistema podrá ser implementado sobre una red de computadores, internet y como última opción para un solo jugador.
- El modelado de los objetos será en 3D.
- El ambiente del videojuego tendrá animaciones y sonido.

- Los objetos que no podrán ser controlados por el usuario tendrán inteligencia artificial.
- El videojuego utilizará lugares de la ciudad de Ibarra – Ecuador como escenarios.
- Los objetos del escenario tendrán efectos como luces, movimiento, etc.
- El control de mando del usuario será a través del teclado y ratón.
- El diseño de los objetos del videojuego serán realizados en una PC normal para mayor compatibilidad donde se instale la aplicación.
- Interfaz agradable e intuitiva para el usuario.

1.5.1 Descripción del videojuego.

La PC donde se realizará el diseño tendrá las siguientes características:

- Procesador Intel Core(TM)2 Duo CPU 2.0 GHz
- 4 GB RAM
- Windows 7 Ultimate 64 bits.

Propuesta para un solo jugador:

El video juego para este modo será al estilo de la saga **Grand Theft Auto**

Descripción:

- Género: Acción.
- Modos de juego: Un jugador.
- Numero de misiones: 10.
- Niveles de dificultad: Cada escenario o misión del videojuego tendrá su propio nivel de dificultad, especificado por cada misión.
- Historia: La trama empieza con un grupo de tres ladrones atacando una sucursal de un Banco (ficticio). Un hombre y una mujer van en cabeza mientras que otro les cubre las espaldas. Sin embargo, al doblar una esquina, el tercer ladrón es disparado por la mujer, quien huye con el otro hombre y el dinero. El traicionado llamado Miguel (ficticio) ha de hacérselo pagar a los traidores: María (ficticio) ex novia de Miguel, y a Luis, que son cabecillas de un cartel colombiano afincado en Ibarra. Las misiones a realizar serán para atrapar a los traidores. (Trama de GTAIII adaptada).

Propuesta para multijugador

El video juego para este modo será al estilo de la saga **Quake.**

Descripción:

- Género: Acción en primera persona.
- Modos de juego: Multijugador.
- Número de conexiones (jugadores) máximo: 4.
- Objetivo: El juego permitirá a los jugadores, cuyas computadoras están conectadas a una red o a Internet, disputar partidas entre sí, en tiempo real. Usa una estructura de arquitectura cliente-servidor que requiere que los clientes de todas las computadoras de los jugadores se conecten a un solo servidor. El objetivo en *Quake* es moverse a través de todo el campo de batalla eliminando (*fragueando*, del inglés *frag*) a los jugadores enemigos y anotándose puntos basándose en los objetivos del tipo de juego. Cuando los puntos de vida de un jugador llegan a cero, el avatar del jugador es eliminado (*fragueado*); luego el jugador reaparece en otro punto del mapa y sigue jugando con sus puntos de vida restaurados, pero sin las armas ni items que recolectó anteriormente. El juego termina cuando un jugador o equipo alcanza un puntaje específico, o cuando se termina el tiempo. (Objetivo de Quake III Arena).

Mapa del Videojuego.

El límite del videojuego será 5 bloques desde el parque Pedro Moncayo hasta la avenida Sánchez y Cifuentes (Norte). Avenida Obispo Mosquera (Este), Avenida José Mejía (Oeste), Avenida Juan de Salinas (Sur).

1.6 Justificación del Proyecto

La presente tesis será una investigación sobre la tecnología para el desarrollo de videojuegos utilizando la plataforma de XNA desarrollada por Microsoft que dará como resultado, una metodología para el desarrollo de videojuegos, y el proceso a seguir para conseguir el producto terminado.

Anteriormente, los videojuegos eran desarrollador en C y C++ cuyos lenguajes ofrecían la manipulación directa de los componentes del computador a bajo nivel. Con la actual tecnología proporcionada por Microsoft manipular los controladores de los componentes del computador queda en manos del API de XNA, mejorando así la utilización del teclado, ratón, gráficos (Direcxt y Open GL) del monitor, sonido, etc. Además de optimizar el código para la ejecución del videojuego.

Desarrollar un videojuego es casi similar a desarrollar cualquier otro software en general, además lo que implica desarrollar un videojuego donde interviene la creatividad para implementar otros componentes, como video, sonido, texturas entre otros.

Debido a la creciente complejidad en el proceso de diseño de los videojuegos, mucha gente joven proceden del campo de la informática o del desarrollo de programas, ya que son los más aptos para afrontar el difícil reto de desarrollar videojuegos.

CAPITULO II.

MARCO TEÓRICO

2.1 Introducción al desarrollo de videojuegos.

Desarrollar un videojuego comprende varias actividades y procesos, los más generales y que abarcan todo el proceso son diseño y producción, para terminar entregando el producto final, en este caso un videojuego. A partir de lo que implica diseñar y desarrollar este tipo de software se puede separar en más procesos y actividades que den un flujo lógico para conseguir este fin.

Desde la concepción del mismo se convierte en todo un reto para quien conciba esta idea, en el cual intervienen varios procesos a seguir para conseguir el producto final, en resumen los procesos se pueden establecer como sigue:

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

Un videojuego va más allá de la creación de software en general, debido a la complejidad del mismo y los componentes que utiliza, por lo antes citado, actualmente los videojuegos son desarrollados por un equipo completo entre programadores, diseñadores, y personas encargadas del sonido, historia, efectos especiales, etc. para entregar al final un producto de calidad, que con las características de un producto de este tipo como jugabilidad, interfaz gráfica, diseño, etc. hacen un medio de entretenimiento de quien lo utilice, además de aprovechar todas las funcionalidades que puede ofrecer un computador o consola de videojuegos.

La evolución de los videojuegos tuvo más auge con la aparición de mundos virtuales en 3D, que de la mano del avance de la tecnología como computadores con varios procesadores, tarjetas de video que soportan grandes cálculos matemáticos que presentados a través de dispositivos de salida como el monitor en caso de PC's y televisores en el caso de consolas de videojuegos como XBOX, Nintendo, etc. atraen al jugador o usuario de la aplicación y estimulando en él la curiosidad por jugar y según como sea del agrado del usuario este tendrá varias horas de entretenimiento.

Los videojuegos además de ser un software de entretenimiento, aportan también con el avance en otras disciplinas, como la inteligencia artificial, simulación computacional, y el desarrollo de hardware más avanzado que soporte la complejidad de los gráficos generados por este tipo de software.

2.2 Teoría de videojuegos.

Desde los inicios de las aplicaciones que podrían ser consideras como videojuegos, están las que tienen gráficos e interactúan con un jugador o varios, los cuales a través de dispositivos de entrada controlan un personaje o de acuerdo al tipo, manifiestan las acciones del jugador en acciones sobre un objeto del escenario, todo esto se puede observar en una pantalla, a partir de esto definir que es un videojuego, va más allá que

interpretarlos como un medio más de entretenimiento como lo son el cine, la televisión, los computadores e incluso los juegos. Es precisamente la exclusividad de lo que puede ser un videojuego lo que lo ha hecho tan difícil de definir formalmente y lo que ha provocado un intenso debate sobre no sólo lo que debería ser, sino también sobre lo que es exactamente. Aunque ya existen múltiples definiciones se podría empezar intentando localizar algunos de los elementos que constituyen un videojuego.

Antes de iniciar con la definición, es necesario indicar como surgió el este término y que se refiere, El PONG (1972) tal vez el nombre del primer videojuego. Dentro de lo que son los videojuegos, es difícil imaginarse un juego comercialmente viable más simple que el PONG. Este consistía en primer lugar, unos jugadores que competían tenían que arrojar una pelota que botaba como en el tenis de mesa; en segundo lugar, los jugadores estaban limitados al movimiento vertical; en tercer lugar, el juego tenía lugar en un monitor de vídeo, y, por último, se anotaba la puntuación según la cual uno ganaba y otro perdía, con estas características se podría considerar que el PONG cumple los requisitos mínimos para ser un videojuego. Aunque existen descripciones detalladas sobre cómo puede definirse el término videojuego, pero a partir de estas características básicas, se puede empezar a delimitar qué se quiere decir cuando se especifica que algo es un videojuego.

En cuanto a la primera parte del término, parece que “video” requiere que la acción del juego aparezca de forma visual en una pantalla, originariamente vídeo se refería a los tubos de rayos catódicos (CRT, de Cathode Ray Tubes) que se utilizaban en los juegos domésticos y arcades, pero ahora los juegos portátiles con visualización basada en píxeles también suelen llamarse videojuegos. La segunda parte del término “juego” es más difícil de definir, como manifestar que un juego es un acto o acciones de una o varias personas que se entretienen, pero en esta definición también están incluidos otros tipos de entretenimiento como el cine o televisión, aquí es donde entra el jugador quien con sus acciones consigue el entretenimiento que el juego le puede ofrecer, entonces la definición seria la siguiente: juego de video, donde las acciones de los jugadores con el objetivo de conseguir entretenimiento aparece en una pantalla.

En los varios enfoques que han utilizado las definiciones de los videojuegos, aparecen de forma persistente unos cuantos elementos, con nombres y descripciones diferentes. Estos elementos se encuentran en el centro de lo que hace que el videojuego sea un medio único, y deben tenerse en cuenta en cualquier debate sobre el tema. Los fundamentales son: algoritmo, actividad del jugador, interfaz y gráficos.

De los cuatro, el más fácil de definir son los gráficos, que se refieren a algún tipo de visualización cambiante y cambiable en una pantalla que produce algún tipo de imagen basada en píxeles. También es necesario especificar que los gráficos del videojuego difieran de las imágenes impresas o filmadas, por el hecho de encontrarse en una pantalla electrónica de algún tipo (una pantalla CRT, LED o LCD, por ejemplo) y tener algún componente movible controlado por el jugador.

Los gráficos no deberían confundirse con el elemento siguiente: la interfaz, puesto que una interfaz puede o no contener gráficos, y no todos los gráficos representan una interfaz. La interfaz se encuentra en la frontera entre el jugador y el propio videojuego, y puede incluir elementos como la pantalla, los altavoces (y micrófonos) y los dispositivos

de entrada (como el teclado, el ratón, la palanca de juego, los remos, los volantes, las pistolas de luz, etc.), además de elementos gráficos en la pantalla (como botones, barras de desplazamiento, cursores, etc.) que invitan a la actividad y permiten que ésta tenga lugar. Así en realidad la interfaz es el punto de unión entre la entrada y la salida, entre el hardware y el software, entre el jugador y el propio juego material, y es el portal a través del cual se desarrolla la actividad del jugador.

Con las acciones del jugador se puede establecer que exactamente son estas acciones las cuales se convierten en el centro de la experiencia del videojuego, y quizás sea lo más importante desde el punto de vista del diseño. Un videojuego se compone de reglas, las cuales el jugador las tiene que realizar para que se cumplan otras reglas, con esto se definen las condiciones para que permanecer jugando caso contrario no se podría jugar, todo esto se desarrolla en el escenario del videojuego, entonces en este ámbito de este elemento con certeza se puede establecer que este es el elemento del videojuego sobre el cual se escribe más, y hasta ahora todas las teorías del videojuego de las que se pudieron consultar parecen estar de acuerdo con la idea de que sin actividad del jugador no habría juego. La naturaleza de la actividad del jugador es necesariamente, una acción que tiene algún aspecto físico y no es estrictamente una actividad que se desarrolle en el plano puramente mental. La actividad del jugador consiste en la entrada de datos por medio de la interfaz del usuario que la limita y normalmente también la cuantifica a modo de puntaje. Además, se podría dividir la actividad del jugador en dos ámbitos: la actividad que hace el avatar del jugador como resultado de su actividad y la actividad que hace físicamente el jugador para conseguir un resultado determinado. Ambas no deberían confundirse, puesto que la traducción de una a otra puede variar mucho.

Finalmente, en el corazón de cualquier programa de videojuego hay un algoritmo, el programa que contiene el conjunto de procedimientos que controlan los gráficos y el sonido del juego, el input y el output donde se implican los jugadores, y el comportamiento en el juego de los jugadores controlados por ordenador. Se podría decir que el algoritmo es el responsable de la representación, las respuestas, las reglas y la aleatoriedad que componen un juego. La representación es la interpretación de los gráficos, los sonidos y los movimientos del juego, y la unificación de estos elementos puede resultar en una experiencia de juego continua y coherente. Las respuestas incluyen las acciones y reacciones que produce el algoritmo como respuesta a las situaciones y datos cambiantes dentro del juego. Eso incluye el control de los actos del juego y de los personajes que no juegan, así como la acción que se desarrolla en la pantalla del avatar del jugador, acción determinada por el input del jugador. Las reglas son las limitaciones que se imponen a las actividades y representaciones del juego, que las determinan y que regulan las respuestas y los movimientos del juego. Incluso los videojuegos más abstractos o abiertos tienen algún tipo de reglas, aunque simplemente consistan en limitaciones a lo que el jugador puede hacer en el contexto del juego. Finalmente, la mayoría de juegos tienen algún elemento de aleatoriedad o, quizás, de impredictibilidad, ya que la verdadera aleatoriedad de lo que se pudo investigar es computacionalmente imposible. La aleatoriedad impide que el juego sea exactamente igual cada vez, y con esto se consigue mantener la intriga en los jugadores y convertir el juego interesante, a través de la variación de los hechos, los momentos y el orden en que se producen. En sentido estricto, la aleatoriedad no es un elemento necesario, puesto que los juegos de problemas y los juegos basados en la narrativa y que generalmente

sólo se juegan una vez puede contener muy poca o nada de aleatoriedad. Sin embargo, la mayoría de juegos tienen algún grado de aleatoriedad para evitar una predictibilidad para no volverse aburridos (por ejemplo, la mayoría de juegos de ajedrez de ordenador, no siempre empiezan con la misma jugada).

Los juegos de ordenador son vividos por los jugadores como narraciones. En los juegos, el jugador recibe una tarea bien definida, como ganar la partida, acabar el primero en una carrera, llegar al último nivel o conseguir la puntuación más alta. Esta tarea es la que hace que el jugador viva el juego como una narración. Todo lo que le sucede en el transcurso del juego, todos los personajes y objetos que se encuentra lo acercan o alejan del objetivo. Como complemento un videojuego debe tener una historia o guion para ser del agrado del jugador, la descripción de cada personaje que interviene en la ejecución del mismo es necesario para una correcta compresión de lo que el jugador puede hacer y que reglas debe cumplir para completar el nivel, que según el tipo de videojuego pueda avanzar al siguiente nivel. Las actividades del jugador si bien definen la experiencia del videojuego esta no podría ser ejecutada sin la información sobre que tiene que hacer y qué objetivos debe cumplir.

Depende de cómo este contemplado el videojuego para que el jugador alcance la victoria, un videojuego que nunca acabe será rápidamente abandonado, como por ejemplo los juegos de tetriz o puzles que mantienen al jugador atrapado a este, no por presentar nuevos escenarios sino más bien al incrementar la dificultad, cuando se logra pasar de nivel y a medida que logra conseguir pasar uno la dificultad aumenta hasta que el jugador pierda consecutivamente, en contraste con videojuegos que presentan misiones, en las cuales los escenarios son diferentes en estos casos existen niveles finales donde al completarlos los jugadores esperan la próxima versión del videojuego.

Los juegos representan un arte nuevo y vital, tan adecuado a la era digital como lo fueron los anteriores medios como el lienzo y el pincel. Facilitan el acceso a nuevas experiencias estéticas y convierten la pantalla del ordenador en un reino de experimentación e innovación ampliamente accesible. Y los juegos han sido acogidos por un público que, en cambio, no está atraído por lo que se considera arte digital.

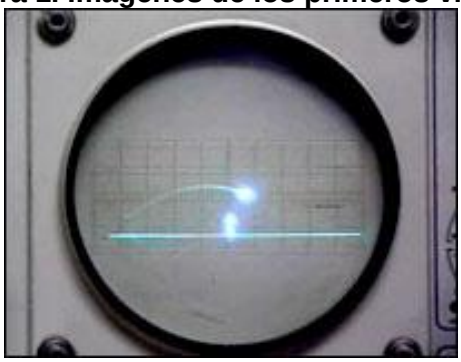
Como el mapa de un juego de aventuras que se va desvelando incesantemente, queda mucho territorio por explorar. La producción de videojuegos hace necesario aclarar sus funciones económicas e ideologías que dan forma a los juegos. La primera, la recepción de los videojuegos por parte de un amplio grupo de consumidores han hecho de esta actividad una de las más rentables a nivel mundial y la segunda, geográficamente se puede considerar que los videojuegos no son del gusto y agrado por todos, pero si lo suficiente como para ser uno de los productos de más aceptación a nivel mundial con una popularidad impresionante, en este contexto con la integración de los videojuegos a los sistemas operativos, a los teléfonos móviles, a los PDA y a prácticamente todos los tipos de tecnología de pantalla accesibles, los videojuegos tienen una generalidad y una accesibilidad que no ha tenido ningún otro medio de la historia.

También se están explorando los múltiples usos de los videojuegos. Investigaciones sobre las aplicaciones de los videojuegos en la educación y la formación. Está claro que el videojuego es una parte importante de la cultura popular y que es probable que lo siga siendo durante mucho tiempo, independientemente de las formas que adopte en el futuro.

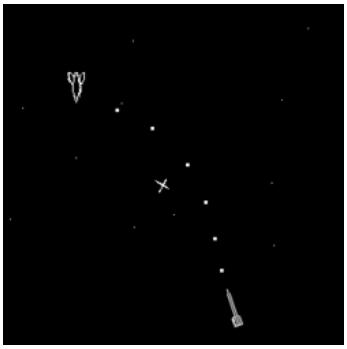
2.3 Historia de los videojuegos

El primer videojuego podría ser Tennis for Two título que tiene en debate con Spacewar, principalmente por la forma de cómo es presentado ante el usuario, mientras el primero utilizaba un osciloscopio a modo de monitor conectado a una computadora analógica, el juego constaba de una línea horizontal que era el campo de juego y otra pequeña vertical en el centro del campo representando la red. Los jugadores debían elegir el ángulo en el que salía la bola y golpearla, el segundo fue un juego interactivo de ordenador implementado sobre un PDP-1.

Figura 1. Imágenes de los primeros videojuegos



Tennis for Two



Spacewar

Fuente: Wikipedia

Con la aparición de gráficos en 3D, comenzó la revolución de los videojuegos, al principio de los años 70, por parte de la empresa Silicon Graphics apareció OPEN GL existente hasta nuestros días, un API que proporciona funciones para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. El principal competidor, una solución por parte de Microsoft con el nombre de DirectX.

Actualmente los videojuegos que se conocen en la actualidad son verdaderas obras de arte, desarrollados por empresas dedicadas enteramente a esta actividad y que por su calidad, son concebidos con agrado por los usuarios que gustan de este tipo de software.

2.4 Introducción a XNA

Microsoft XNA (XNA no es un acrónimo), simplemente es un conjunto de herramientas que facilita el desarrollo de videojuegos utilizando el lenguaje C# y Visual Studio.

Dentro de las herramientas que provee XNA están métodos de ayuda para cálculos matemáticos necesarios por la aplicación, renderización de los objetos del videojuego, manejo de teclado, reproducción de sonido y video, aplicación de efectos especiales sobre los modelos 3D, soporte para el desarrollo de videojuegos multijugador (LAN, internet y varios jugadores en la misma máquina.) y una gran ventaja de esta plataforma es poder migrar el código para el soporte en consolas de videojuegos obviamente

creadas por Microsoft como XBOX 360 y Zune con mínimos cambios en el código, pero en el presente proyecto solo se lo implementó sobre PC.

Como desventaja, XNA no tiene herramientas nativas que ayuden al desarrollo de videojuegos, como soporte para la reproducción de animaciones de modelos 3D, física, entre otros, como los tienen otras herramientas como OGRE, y en comparación de ambos, si bien XNA es un framework para el desarrollo de videojuegos y OGRE es un motor de videojuegos, se comprendería en algo la limitación de XNA, pero afortunadamente gracias a una comunidad activa de desarrolladores de videojuegos que emplean esta herramienta, como codeplex se pueden encontrar varias librerías gratuitas y comerciales que satisfacen algunas de la necesidades a la hora de desarrollar un videojuego, como por ejemplo en el proyecto realizado se utilizaron las siguientes librerías:

- XNAnimation (librería de animación 3D).
- JigLibX (motor de física).
- DPSF (sistema de partículas).

Los videojuegos utilizan un sin número de componentes que se encargan de operaciones especificas dentro de la ejecución del sistema como son las reproducción de sonido, video, animaciones, física, colisiones, etc. las cuales se ejecutan de manera independiente en algunos casos, como por ejemplo las animación de los personajes del videojuego no dependen del sistemas de colisión a menos que estos interactúen con un objeto físico del escenario.

XNA como se mencionó anteriormente no es un motor de videojuegos, se entiende como motor de videojuegos, como el medio por el cual todos los componentes se constituyen como un único componente, es en este punto donde se realizó el mayor esfuerzo en la construcción del sistema propuesto, ya que en las fuentes consultadas no había información concisa y clara de cómo realizar esta tarea la cual era imprescindible para este proyecto, no obstante y a medida que el proyecto lo requería, se logró construir un motor de videojuegos, con lo que se comprobó la versatilidad y flexibilidad de XNA, en un principio se hizo difícil la incorporación de los elementos necesarios para conseguir este fin, pero luego de la experiencia adquirida esto no fue mayor problema, la estructura de cómo implementar un sistema de este tipo es importante ya que sin un flujo lógico propuesto desde el inicio del desarrollo, puede terminar un sistema desordenado que puede dar como resultado de esto un proyecto destinado al fracaso.

XNA utiliza un lenguaje de programación orientado a objetos cuyas características están pensadas en la reutilización de código, manejo de objetos o entidades del mundo virtual que utilizan algoritmos especiales y acciones propias de cada objeto.

La matemática es un ámbito muy importante al desarrollar un videojuego y gracias a XNA que dentro de las herramientas que ofrece están métodos matemáticos que se aplican a cálculos con matrices, vectores, y demás estructuras que pueden existir dentro de un mundo virtual sea este 2D o 3D. El proyecto desarrollado implemento un escenario en 3D por lo cual en su mayoría se utilizó vectores de 3 dimensiones y matrices, acepto en los casos en lo que se requería dibujar texturas 2D y texto, según el temario indicado se explicará más detalladamente.

Los bondades de XNA más todo el potencial que puede ofrecer .NET hace de esta herramienta un medio por el cual, principiantes puedan desarrollar productos de calidad e incluso los profesionales en desarrollo de aplicaciones de entreteniendo ya que a través de XNA no tendrán que preocuparse por controlar los tiempos de refresco entre cada escena, escribir el mismo código por cada objeto, cálculos matemáticos complejos, migración hacia otros dispositivos, etc.

2.5 XNA

El desarrollo de videojuegos es realizado generalmente en el lenguaje de programación C++, ya que este ofrece la posibilidad del manejo de hardware directamente, al igual que la utilización de las funciones que ofrecen DirectX y Open GL, algunos de los motores de videojuegos han utilizado este lenguaje, como el motor de UNREAL 3 el cual es utilizado sobre algunos productos exitosos con efectos visuales agradables, sonido en 3D, formas y figuras más detalladas, luces e iluminación, sombras, etc.

XNA facilita radicalmente el acceso al mundo de DirectX y Direct3D, ofreciendo a los desarrolladores nuevas posibilidades para el desarrollo de aplicaciones para que aprovechen las capacidades de generación de escenas gráficas complejas en tiempo real, combinado con la potencia de C# y .Net.

Entre las herramientas que provee este framework esta XACT que remplazó a DirectSound dentro de la API de DirectX 10 y que a través de las clases de XNA se consigue reproducir sonido con efectos especiales como: reverberación, sonido 3D, subir o bajar el volumen de los sonidos y demás funciones, todo esto está a cargo del framework, además facilita la lectura de eventos desencadenados por el teclado y ratón en caso de que el videojuego este implementado sobre PC, en el caso de las consolas también provee facilidad de lectura de datos de los controles de mando a través de XInput que remplazo a DirectInput.

Figura 2. Dispositivos de entrada

Teclado

Mando XBOX

Fuente: El autor

2.6 Entorno de XNA

Para desarrollar y ejecutar una aplicación XNA es necesario tener en cuenta lo siguiente:

Un entorno de desarrollo en este caso Visual Studio 2008 con la versión de XNA 3.1 previamente instalado dentro de este IDE, además de la última versión de DirectX que es lo más recomendable, hasta la redacción de este documento la versión actual es la 11.0

Figura 3. Entorno de ejecución de XNA

<table>
<tr><td colspan="3">Microsoft Visual Studio 2008</td></tr>
<tr><td colspan="3">XNA Framework</td></tr>
<tr><td colspan="2">.net Framework</td><td>.net Compact Framework</td></tr>
<tr><td>Windows</td><td>XBOX 360</td><td>Zune</td></tr>
</table>

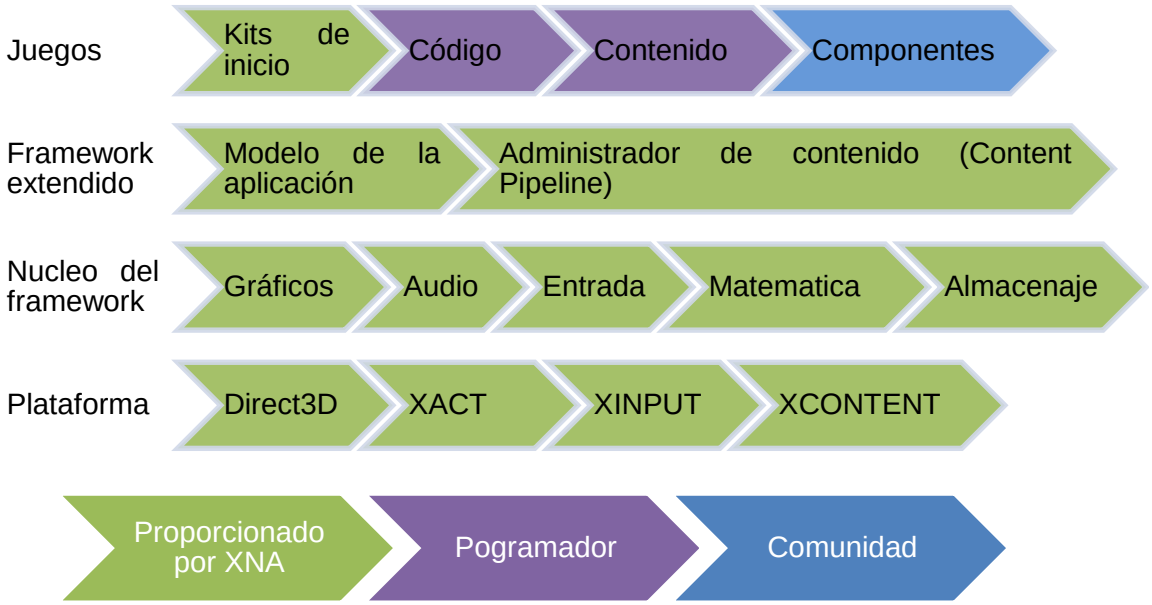
Fuente: Wikipedia

Como se observa en la imagen, el framework XNA se ejecuta bajo .NET. Se puede apreciar, que según en la plataforma donde esté en ejecución, el framework .Net no será el mismo, pues Xbox 360 y Zune a diferencia de Windows, utilizan el compact Framework, que además, no es el mismo compact framework de Windows Mobile. De esta manera se sabe que XNA funcionará tanto en PC con Windows (XP o Vista), Xbox 360 o Zune, pero no por el momento en Windows Mobile, actualmente XNA funciona correctamente en Windows Phone.

Uno de los objetivos es facilitar el desarrollo de videojuegos para las tres plataformas (Windows, Xbox 360 y Zune). Es decir, crear un juego para Windows y luego crear su versión para Xbox rápida y fácilmente sin mayores cambios en el código. Esto es posible gracias a las API que proporciona el framework XNA, que en su mayoría sirven para todas las plataformas. Evidentemente se encontrarán funcionalidades que únicamente se podrá utilizar para una determinada plataforma, pero en general, muchos videojuegos hechos con XNA son 100% compatibles entre las diversas plataformas.

Otro de los objetivos del framework XNA está enfocado en la misma dirección que el anterior, facilitar el desarrollo de videojuegos. Hacer un juego no es fácil, no lo es para los profesionales que se dedican a ello, menos lo va a ser para la gente que empieza. Hay un mucho código que escribir, pruebas a realizar y errores que tratar antes de incluso dibujar un punto en la pantalla pero gracias al framework XNA, algunos problemas están resueltos lo demás es por cuenta del desarrollador o en el mejor de los casos del equipo de desarrollo.

Figura 4. Modelo de capas del framework XNA.



Fuente: http://creators.xna.com/en-US/

Plataforma:

Es la capa más baja, contiene las API nativas encapsuladas por XNA que son utilizadas en las capas superiores, en otro nivel, por las clases administradas. Algunas de las API que incluye esta capa son Direct3D 9, XACT, XINPUT, XCONTENT.

Núcleo del framework:

El núcleo, en realidad, es la primera capa del framework XNA. Proporciona las funcionalidades básicas sobre las que las otras capas trabajan. Si se quisiera proporcionar funcionalidad adicional gestionando con DirectX, se construiría en esta capa. Las áreas agrupadas en el núcleo son las siguientes:

- **Gráficos:**

Como se ha visto anteriormente, la API gráfica (Graphics) que utiliza XNA es Direct3D. En XNA se puede programar Shaders y efectos, y para programar los juegos sin tener que desarrollar los shaders, existen clases que encapsulan shaders, como BasicEffect para definir el detalle final del renderizado de la escena y modelos 3D, o SpriteBatch para el muestreo de sprites en 2D.

- **Audio:**

XNA admite los siguientes formatos de archivos de sonido: .xap, .wav, .wma y .mp3.

El sonido en XNA, antes de la versión 3.0 sólo se reproducía a través de XACT (Cross-platform Audio Creation Tool), que es una librería de alto nivel para audio, fue realizada por Microsoft como parte de las SDK de directX y originalmente fue diseñada para Xbox

como API para el procesamiento óptimo de la señal digital. Posteriormente fue modificada para su funcionamiento en Windows.

Dentro de las herramientas que proporciona el XNA Game Studio, se encontrará una aplicación para crear los proyectos de audio necesarios para la API XACT. La aplicación se llama exactamente igual: Microsoft Cross-platform Audio Creation Tool (XACT), es un editor donde se puede empaquetar/agrupar nuestra música (en archivos .wav), configurando opciones como el volumen, las repeticiones mediante bucles, la mezcla de canales, etc. Esta herramienta guarda los proyectos en archivos .XAP para luego importarlos a nuestro proyecto XNA a manera de un banco de archivos de audio.

En el "pack" de herramientas suministradas por XNA también encontraremos XACT Auditioning Utilility, que es el servidor que reproducirá el sonido.

A partir de la versión 3.0, además de importar archivos .xap, también se puede añadir directamente ficheros .mp3, .wav y wma, los cuales se los podrá reproducir con clases específicas para manejar archivos de audio que no se pueden agregar dentro de un proyecto de XACT, herramienta que va ser detallada en los siguientes capítulos.

- **Entradas:**

La API de entradas (Input) del usuario, es XINPUT de DirectX, es de acceso inmediato, no requiere ningún tipo de inicialización previa ni nada por el estilo, es más, no hay que preocuparse de asignar o liberar el dispositivo de entrada. Lo único que hay que hacer es llamar al método GetState sobre el controlador adecuado y listo. Hay clases para los controladores necesarios: el teclado (Keyboard), el ratón (Mouse) y por supuesto el mando de la Xbox (GamePad). Hasta se pueden controlar los motores de vibración del mando de una forma sencilla.

- **Matemáticas:**

La API de matemáticas (Maths), proporcionan una gran colección de clases y métodos para el cálculo matemático. Por ejemplo, incluye los tipos de datos que comúnmente se usan en la programación de videojuegos, tales como Vector2, Vector3, Vector4, Matrix, etc. Además también incluye tipos de volúmenes como BoundingBox, BoundingSphere y BoundingFrustum, que poseen métodos para la detección de colisiones.

Las matemáticas en XNA a diferencia de DirectX (mano izquierda), usa por defecto el sistema de coordenadas basadas en la regla de la mano derecha (right-hand coordenate system). No obstante se podría utilizar la regla de la mano izquierda (left-hand coordenate System), en dicho caso, muchos cálculos matemáticos se deberían realizar por cuenta propia.

- **Almacenamiento:**

La API de almacenamiento (Storage), proporciona formas para leer y escribir los datos de los juegos, como por ejemplo, las partidas guardadas, puntuaciones etc.

Framework extendido, extensible o de extensión

El objetivo principal de esta capa es facilitar el desarrollo al programador. Existen dos elementos principales en esta capa:

- **Modelo de aplicación:**

El propósito del modelo de aplicación (Application model) es el de apartar al programador de los problemas con la plataforma en la que se ejecuta el videojuego. No se tendrá que preocupar, de crear una ventana para el juego o controlar los eventos del SO si estamos en Windows. Tampoco de la creación de contadores de tiempo, bucle del juego, etc.

También proporciona la clase GraphicsDevice, que se encarga de la creación y gestión del dispositivo gráfico, y un GraphicsComponent que se lo utilizará para el renderizado de las escenas, y que a su vez gestiona dicho GraphicsDevice.

Además, en esta parte también se proporciona un modelo de componentes, que permite crear GameComponent para incluirlos a los proyectos que se esté desarrollando, ya estén hechos por uno mismo o por otras personas. Esto permite, si se desease, crear una propia librería de componentes y reutilizarlos en diferentes proyectos, así como compartirlos con la comunidad.

- **Administrador de contenido (Content Pipeline):**

La Content Pipeline permite a los desarrolladores incorporar contenidos multimedia a los proyectos de XNA, tales como imágenes, sonido, modelos 3D, efectos, etc. Facilita el acceso a estos archivos y nos proporciona una interfaz unificada sin una excesiva complejidad.

Permite una amplia gama de formatos de archivos diferentes, que básicamente son los más utilizados en la creación de videojuegos. No obstante, es un número limitado, no soporta todos los formatos actuales, además de que muy comúnmente, en la industria de los videojuegos, los desarrolladores utilizan archivos personalizados para sus proyectos, por lo que el Content Pipeline es extensible. Incorpora un marco de trabajo que permite fácilmente incluir soporte a diferentes tipos de archivo.

Juegos

La capa más alta del XNA. Aquí se encuentra el código del juego, incluyendo los componentes, los kits de inicio etc.

- **Kits de inicio:**

Hay multitud de kits de inicio para crear videojuegos, no se tiene que programar el juego desde el principio. Se pueden utilizar estos kits como punto de partida o simplemente como referencia, ver cómo está escrito y aprender de ellos.

- **Código:**

Sencilla y llanamente el código del juego.

- **Contenido:**

Archivos de imagen, sonido, etc.

- **Componentes:**

Los componentes creados por uno mismo o por otras personas.

2.6.1 La Content Pipeline

Dentro de un proyecto XNA, existen mucho contenido multimedia propio de un videojuego, cada uno con diferentes formatos, y cabe resaltar que dentro de este framework, a menos que se extienda la content pipeline solo admite ciertos formatos que pueden ser usados a través del procesador de contenidos.

Figura 5. Formatos soportados por la content pipeline

3D	2D	MATERIALES	SONIDO
.X (MICROSOFT DIRECTX)	.DDS (MICROSOFT DIRECTX)	.FX (MICROSOFT DIRECTX)	.XAP (MICROSOFT XACT)
FBX (AUTODESK)	.BMP		
	.JPG		
	.PNG (W3C)		

Fuente: El autor

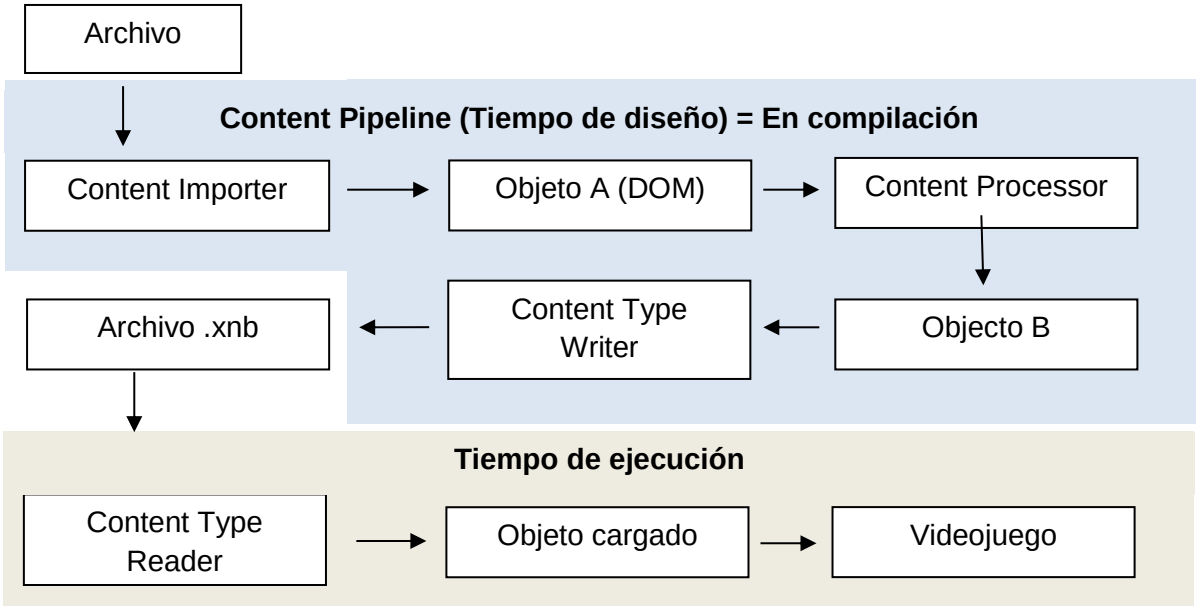
En el transcurso de construcción de un archivo de contenido digital, la content pipeline invoca cuatro componentes principales para transformar el archivo original de contenido en un archivo binario para el videojuego .xnb en caso de no hacer más extensible el procesador de contenido.

Los videojuegos profesionales utilizan su propio generador de contenido, de lo investigado se puede estar seguro que el motivo de esto es por seguridad, ya que una extensión de un archivo de contenido y con el correcto algoritmo para entender este archivo puede ser leído solo por la aplicación que lo genero o por el programa que tenga lo necesario para entenderlo, además creando un generador de contenido propio se puede aplicar algoritmos para comprimirlos, con esto se puede lograr que la aplicación sea más ligera sin sacrificar la calidad del videojuego.

XNA ofrece la opción de generar procesadores de contenido propios de manera fácil y sencilla ya que dentro de su API provee métodos para conseguir este objetivo.

Esto se explicará mejor con un gráfico:

Figura 6. Proceso de la content pipeline sobre archivos del proyecto.



Fuente: http://geeks.ms/blogs/jcanton/archive/tags/XNA/default.aspx

El procesador de contenido se encarga de leer el archivo multimedia y pasarlo dentro del videojuego, pero con un formato propio de XNA, con esto se logra mayor seguridad de los archivos propios del videojuego y del autor, ya que al extender el procesador de contenido para que solo pueda procesar un formato especifico de archivo en este caso el definido por el usuario, se estaría asegurando que esos archivos solo puedan ser interpretados por la aplicación que se esté desarrollando.

Content Importer

Cuando se añade un contenido, este lo recoge un importador. El importador es el encargado de obtener los datos y normalizarlos. Después de que el importador ha hecho su trabajo, los datos existen en un DOM de formato conocido.

DOM (Document Object Model), es el término que se utiliza para indicar una colección de clases o un esquema (por ejemplo un xml).

Este DOM devuelto por el importador será el que utilizará el content processor (Procesador de contenido). En ocasiones, no tiene por qué devolver un DOM, sino que podría también retornar un objeto personalizado creado por el desarrollador.

XNA proporciona un número limitado de importadores de contenido estándar para los formatos de archivo comúnmente más utilizados en la creación de videojuegos. Para los formatos de archivo que no estén soportados por estos importadores estándar, podemos crear nuestro propio importador personalizado.

Content Processor

El procesador de contenido recibe los datos que el importador ha generado y crea un objeto para su uso en tiempo de ejecución. Este objeto puede ser tan simple como un modelo o mucho más complejo.

Los procesadores están diseñados para que sean fáciles de escribir, compartir y reutilizar. El procesador funciona con independencia del formato del archivo original, debido a que todos sus datos están almacenados en un formato conocido. Además existen métodos de ayuda que permiten manipular y generar nuevos datos de contenido DOM.

La content pipeline de XNA incluye algunos procesadores, por ejemplo:

- Model (para objetos simples con texturas).
- Texture2D (para sprites o combinaciones con modelos).
- Effect (para manejar los materiales de los modelos), etc.

Estos procesadores nativos de xna se los puede hacer más flexibles para que al ser procesados, ofrezcan más información como por ejemplo: modificar el procesador de modelos para que aparte de procesar el modelo construir una esfera a partir de este, útil para la detección de colisiones, encapsular esta información y guardándolo en el tag del modelo para su uso en la programación del videojuego.

Content Type Writer

El content Type Writer es el responsable de recoger el objeto generado por el procesador y grabarlo físicamente en disco. Este archivo binario se guardará en un formato específico de XNA con la extensión .xnb

Content Type Reader

El content Type Reader es el encargado de almacenar el objeto en memoria cuando queramos acceder al contenido, dicho de otra manera, deserializa el archivo creado por el writer y lo almacena en un objeto en memoria para su uso. El reader es llamado por XNA mediante el componente Content Manager.

2.6.2 Ciclo de vida de un videojuego en XNA

Se entiende como ciclo de vida de un videojuego independiente de la herramienta utilizada como la ejecución de la aplicación desde el momento en que la aplicación inicializa todas las variables, carga el contenido, realiza el bucle de actualización y dibujado en pantalla de los recursos y lógica del videojuego hasta que estos son liberados, que es cuando la aplicación es cerrada.

XNA provee a los desarrolladores clases principales para gestionar todos los recursos de la aplicación, estas clases heredables, con esto se logra que cualquier objeto se comporte como un componente de nuestro videojuego.

Game: usado por la clase principal, es la responsable de manejar el videojuego y donde se añaden los componentes del mismo.

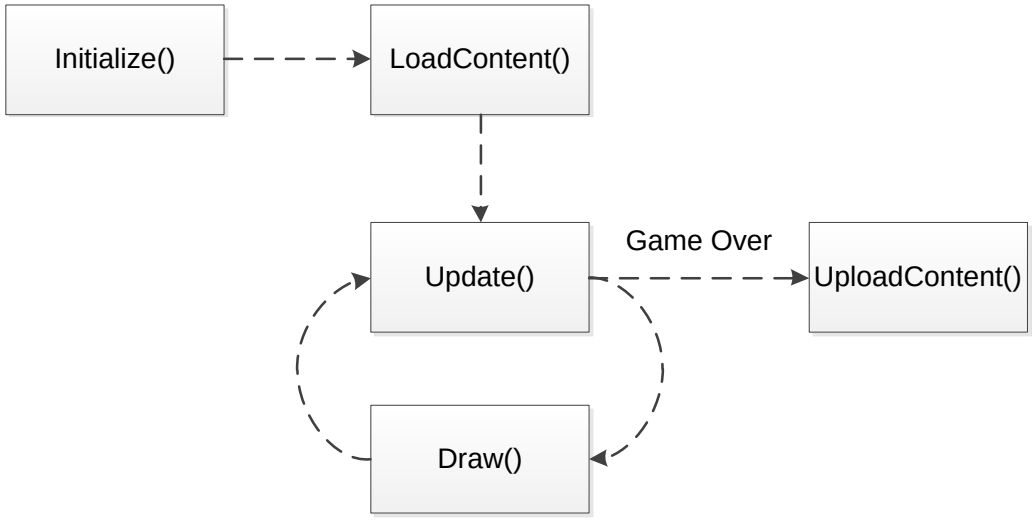
GameComponent: componente del videojuego que no se dibuja, útil para la cámara y otros objetos.

DrawableGameComponent: gamecomponent que se dibuja ya que implementa el método Draw, generalmente utilizado por modelos 3D, sprites, etc.

La clase Game es la más importante ya que es aquí donde se inicializa y configura el dispositivo gráfico, esto debe realizarse una sola vez, motivo por el cual solo debe haber una instancia de esta clase a lo largo del desarrollo del proyecto.

Al iniciar un proyecto en XNA este nos provee de cinco métodos importantes, los cuales se encargan de funciones específicas y que deben ser utilizadas de manera correcta.

Figura 7. Ciclo de vida de un videojuego XNA



Fuente: Libro Learning XNA 3.0

El método Initialize() inicia todas la variables de la lógica del juego que pertenezcan al componente, este método es llamado después de instanciar el objeto es decir pasa por el constructor del objeto o componente, aunque tiene el mismo comportamiento que el constructor, pero por motivos de llevar una estructura lógica del proyecto es preferible inicializar los objetos utilizados dentro de este método.

LoadContent() siguiendo con la estructura de un videojuego XNA en este método se deberá cargar el contenido del videojuego o del componente, como fuentes, sprites, modelos, audio, video.

Update(), controla toda la lógica del componente.

Draw(), realiza todo el proceso de pintado: sprites, objetos 3D, post-procesado, etc.

Como se notará en el gráfico hay un bucle que involucra dos métodos Update y Draw cada vez que se actualiza el comportamiento de un objeto del videojuego este debe ser pintado en pantalla y viceversa.

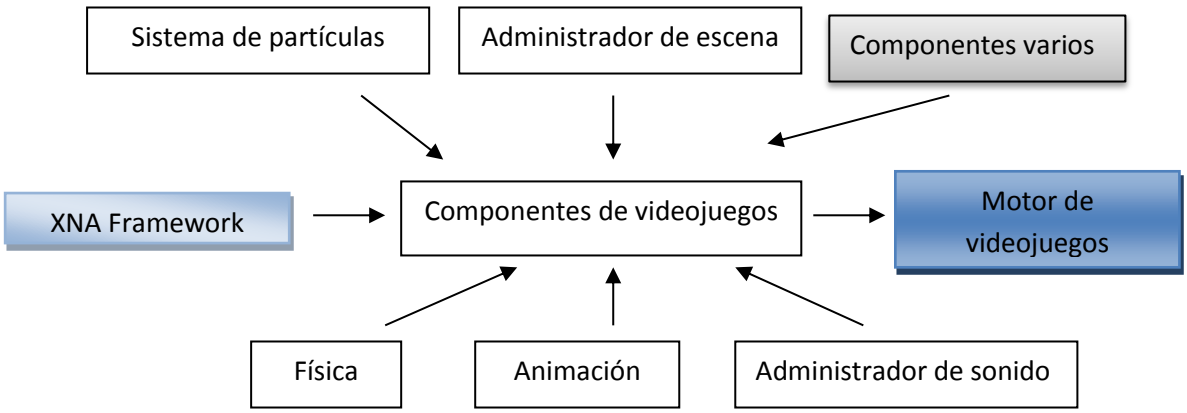
En este bucle es donde mayor cuidado se debe tener ya que una mala implementación de algún algoritmo sobre un objeto haría que la aplicación no se comporte como debería, un resultado de esto, y de fácil percepción es la baja cantidad de frames por segundo que son actualizados y pintados.

Una vez realizado toda la lógica del videojuego habrá un momento que el jugador cierre la aplicación, pierda el nivel, o cualquier otro motivo que provoque que todos los recursos sean liberados de memoria, esto se lo realiza a través del método UnloadContent().

Los frames por segundo o framerate es la cantidad de veces que el videojuego es actualizado y posteriormente dibujado en pantalla, en XNA por defecto son 60 fps, que a lo largo del desarrollo puede ir reduciendo este número o según el rendimiento de la maquina este será más alto o más bajo. También sirve para propósitos de depuración, si después de modificar el comportamiento de algún objeto en la lógica del videojuego, al compilar la aplicación se observa lentitud y ver un número de frames bastante bajo, esto podría deberse a una mala programación.

Como se definió anteriormente sobre que es XNA y las facilidades que ofrece en cuanto al desarrollo de videojuegos se refiere, aun así, con todo lo antes expuesto, y tomando en cuenta que XNA es un framework de desarrollo, a partir de este antecedente se tuvo que hacer el paso a motor de videojuegos ya que gran parte del código para desarrollar un videojuego se debe obtener de terceros o implementarlo uno mismo.

Figura 8. Paso de XNA framework a motor de videojuegos básico.



Fuente: El autor

En el gráfico anterior se muestra la forma lógica en la que se concibió construir el motor de videojuegos, aunque faltan más componentes que a lo largo de la descripción de cómo se desarrolló esta aplicación se lo irá mostrando.

Las librerías que pudieron ser obtenidas a través de internet y que cumplían con los requerimientos para construir el motor de videojuegos fueron:

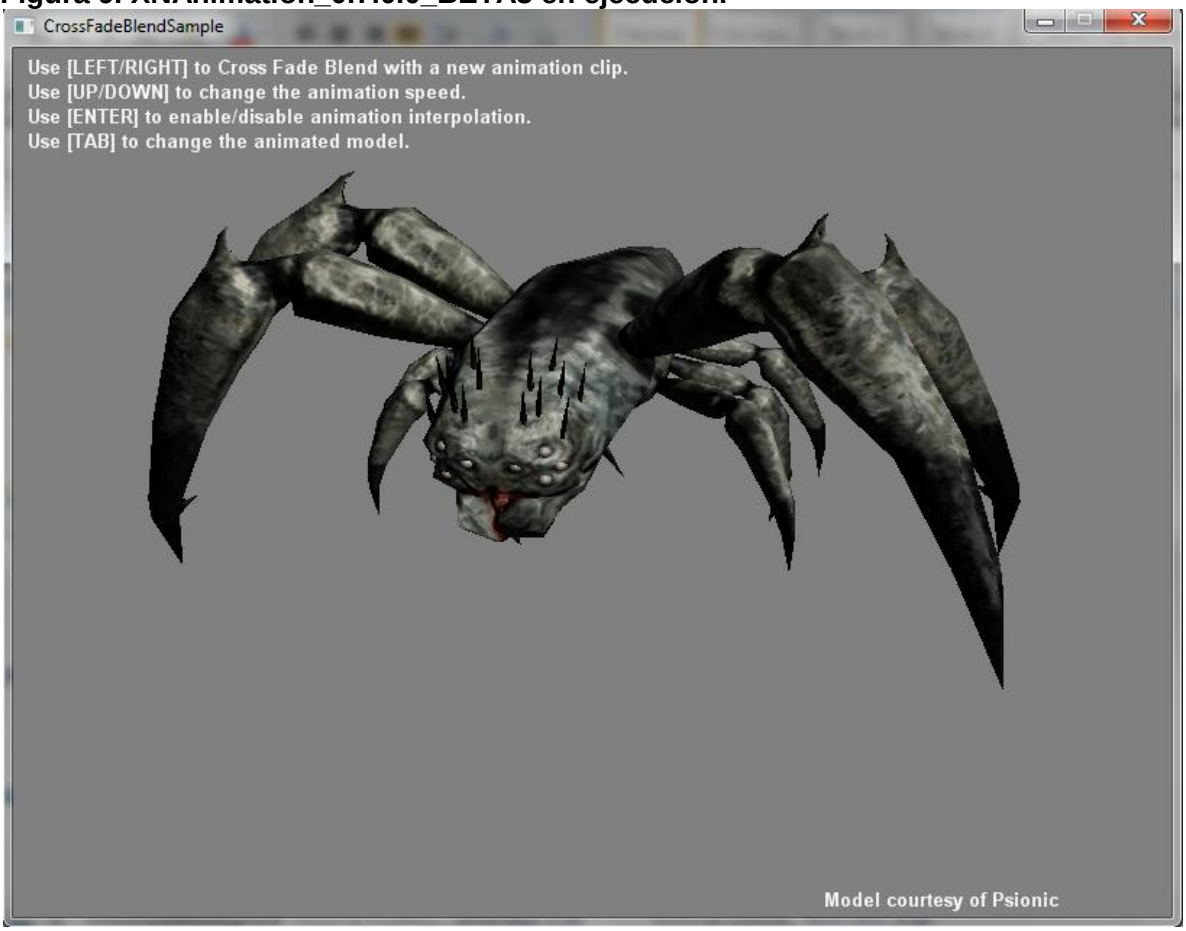
- XNAnimation_0.7.0.0_BETA3
- JigLibX 0.3.1.
- DPSF Versión 1.5.0.0

Todos estos componentes se lo pueden descargar de forma gratuita a través de la página web de codeplex a excepción de DPSF que tiene su propia web oficial.

XNAnimation_0.7.0.0_BETA3

Excelente librería de animación a través de keyframes de los modelos animados utilizados por el videojuego, soporta el formato .X y .FBX que son nativos de XNA. Además de ofrecer funcionalidades sobre el modelo y los huesos que conforman el esqueleto del modelo. La renderización del modelo animado también presenta algunas opciones de efectos que puede aplicarse en tiempo de ejecución como por ejemplo: puntos de luz, color difuso, color ambiente, color especular, etc.

Figura 9. XNAnimation_0.7.0.0_BETA3 en ejecución.



Fuente: El autor

JigLibX 0.3.1.

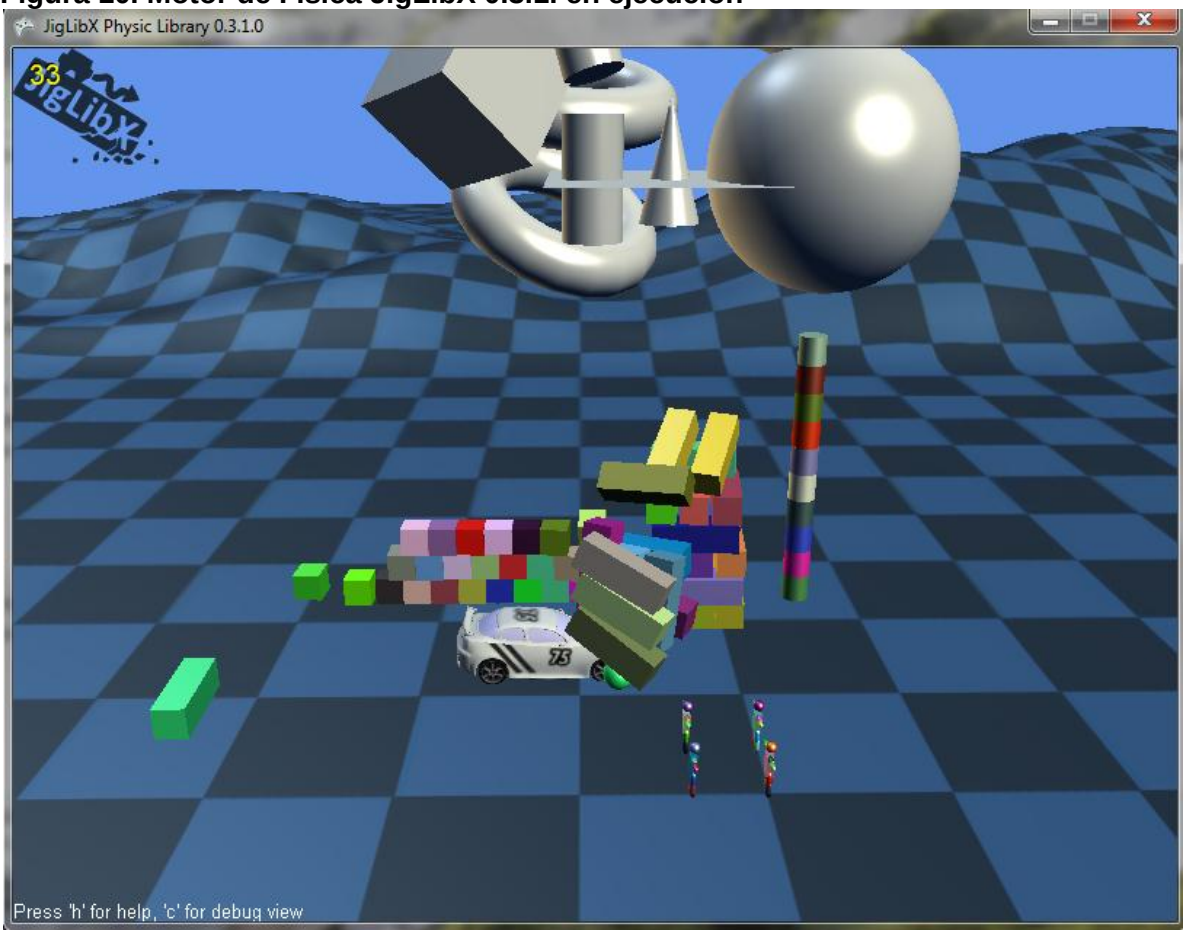
Motor de física y uno de los más utilizados a nivel mundial pero no en videojuegos profesionales. Rápido sobre PC pero un poco lento sobre XBOX 360, la descarga de esta librería trae consigo el código fuente y un ejemplo donde se aplica entidades físicas donde se puede observar el comportamiento de estos en el mundo 3D.

Existen otros motores de física, pero se adaptó esta debido a que muchas de las características que requería el proyecto, solo se las podía implementar utilizando esta librería. Su principal ventaja ante las demás y que ayudo en gran parte al desarrollo del

proyecto es la simulación de la física sobre vehículos, característica que ninguna otra librería ofrecía.

El ejemplo que acompaña a esta librería es de fácil entendimiento para usuarios que ya hayan utilizado XNA, no obstante el código tuvo que ser modificado para que se comporte como se requería para luego ser implementado sobre el proyecto.

Figura 10. Motor de Física JigLibX 0.3.1. en ejecución

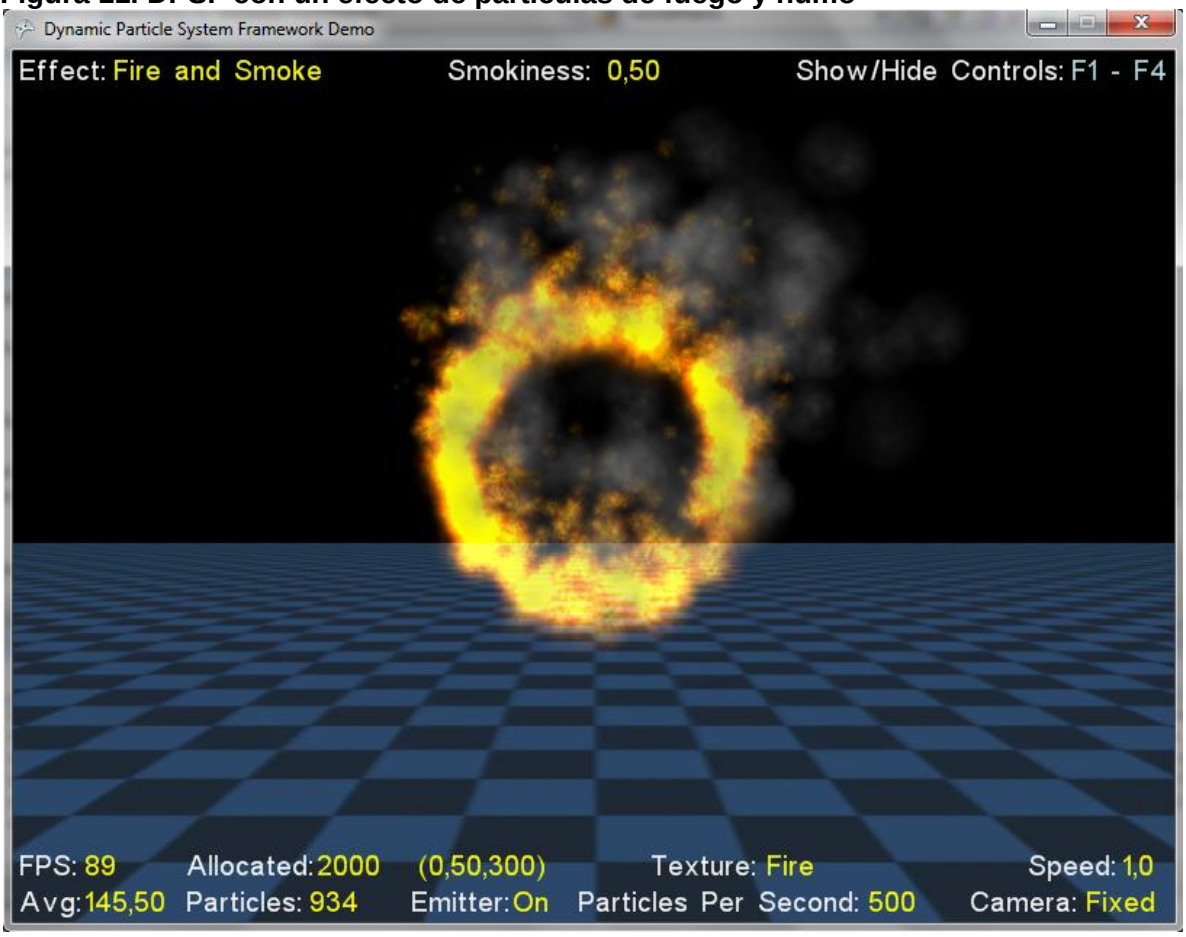


Fuente: El autor

DPSF Versión 1.5.0.0

Dynamic Particle System Framework, es un framework de sistema de partículas dinámico (traducción al español), con una amplia documentación. La descarga de este componente no viene con el código fuente, pero está acompañado con plantillas de sistema de partículas fácilmente personalizables donde muestra su uso y lo que se puede lograr con esta excelente librería, su uso es relativamente sencillo, consiguiendo efectos de partículas realmente buenos.

Figura 11. DPSF con un efecto de partículas de fuego y humo



Fuente: El autor

Dentro de los proyectos alojados por la comunidad de codeplex, existen motores para videojuegos que pueden ser utilizados, y en un principio se trató de utilizar uno de estos, pero por la falta de experiencia en la utilización del framework, y más aún, en lo que representaba utilizar un motor de videojuegos, se optó por construir uno propio, con las ventajas y desventajas que esto presentaba.

Como principal ventaja, se concluyó que el código fuente del motor construido, puede ser fácilmente modificable ya que se conocía su estructura y de cómo funcionaba para cumplir su propósito, sin menospreciar los motores existentes que obviamente son mejores que el que se construyó, esto debido a que por lo general un motor de estos lo construyen varias personas con experiencia en desarrollo de videojuegos.

La desventaja presentada y la que influyó en el tiempo de desarrollo de este proyecto es exactamente la antes mencionada, el tiempo, la construcción del motor fue la parte más difícil que se tuvo que afrontar, ya que cada componente del motor debía comunicarse con uno más componentes del mismo.

Además de estas librerías utilizadas, se requirió de más componentes como son: el administrador de sonido, el administrador de escenas o de pantallas, manejo de entrada de datos por teclado, ratón y a excepción del videojuego desarrollado para más de un jugador, se incluyó el soporte para la entrada de datos a través del control de mando de una XBOX, aunque no se probó la aplicación sobre esta consola, en la programación se

incluyó esta característica. Todo esto y que a través de ejemplos conseguidos en internet ayudaron a la construcción de un motor de videojuegos que se adapte a las necesidades de los videojuegos que fueron desarrollados, dicho motor que fácilmente en un futuro se lo puede mejorar y tal vez comercializar videojuegos nacionales.

2.7 Componentes.

Lo más importante dentro del desarrollo del presente proyecto, fue la construcción del motor de videojuegos, independientemente del género de juego desarrollado, a continuación se explica que es un motor de videojuegos.

Un motor de videojuego es un término que hace referencia a una serie de rutinas de programación que permiten el diseño, la creación y la representación de un videojuego.

La analogía con el motor de un automóvil es ilustrativa: el motor debajo del capó no es visible pero le da la funcionalidad al automóvil que es la de transportar. La misma analogía permite explicar algunos de los aspectos que generalmente maneja un motor de juego: las texturas y los modelos 3D serían la carrocería, pintura y exteriores.

Del mismo modo que carrocería, pintura y exteriores de un automóvil no son funcionales sin un motor al que son añadidos, los gráficos y los guiones del juego no funcionan sin un motor de juego. Un ejemplo de motor de juego seria el motor gráfico del juego Doom.

Entre los términos utilizados dentro del presente documento y que se los nombra para hacer referencia al motor de videojuegos desarrollado son los siguientes con su respectiva explicación:

Assets (Activos)

Son los modelos, animaciones, sonidos, IA, físicas. Son los elementos que forman el juego en sí, el código hace funcionar los assets. Dependiendo del formato del asset el procesador de contenido será el encargado de insertarlo dentro de la aplicación.

Application Programming Interface (Interfaz de Programación de Aplicaciones)

Es un sistema de rutinas, de protocolos y de herramientas para desarrollar programas de aplicación. Un buen API hace más fácil desarrollar un programa proporcionando todos los bloques del desarrollo del programa. El programador pone los bloques juntos.

Entre estos los más importantes son el DirectX (de Microsoft) y el OpenGL (que trabaja con la mayoría de los sistemas operativos).

La única API soportada por XNA es obviamente DirectX.

Render (Renderización)

Es la parte del código que pone en pantalla los ambientes y objetos.

Objetos 3D

Los objetos se almacenan por puntos en un mundo 3D, llamados vértices. Los vértices van formando polígonos; cuanto más polígonos posea un objeto, más complicado se hace, lleva más tiempo de procesamiento pero es más detallado. El juego no necesita saber cuántos objetos hay en memoria o como el Render va a mostrarlos, solo le interesa que el render los despliegue de la forma correcta, y que el modelo este en el cuadro correcto de la animación.

Higher-order surfaces (superficies de alto orden)

Renderizado matemáticamente, usado en las tarjetas gráficas más recientes y poderosas. También llamado: Patches (parches).

Patches (Parches)

Son perfectos para describir geometrías, sobre todo cuando se trata de curvas, pues la expresan mediante fórmulas matemáticas logrando colocar puntos en el mundo del juego.

Three-point polygon (polígonos de 3 puntos-Triángulos)

El más usado por las tarjetas aceleradoras 3D.

Culling

Codificado que logra que los objetos que no se ven en determinado cuadro de la animación por causa de objetos que los obstaculizan (como una pared) no tomen tiempo de renderizado. Así se reduce la cantidad de trabajo del motor. El Culling es más fácil de implementar en juegos en donde la visión es controlada como los RTS en comparación con lo FPS. Un método de Culling puede ser por “Árboles BSP”.

Retesselation

Técnica usada por la característica de TruForm de ATI que consiste en tomar un modelo basado en triángulos y transformarlo en uno de High-Order Surfaces para alisarlo y de nuevo pasarlo a un modelo de Triángulos.

Iluminación (lighting)

Distintos APIs proveen diferentes tipos de iluminación como luces direccionales, luz ambiente, puntos de luz, etc. Se debe tomar en cuenta que la aplicación de iluminación en algunas escenas necesitan de sombras para dar mayor realismo.

Vertex Lighting

Se determinan cuantos polígonos cruzan el vértice, se toma el total de todas las orientaciones de los polígonos (Normal) y se asigna la normal al vértice. Para cada vértice, un polígono dado reflejará la iluminación en una forma levemente distinta. La ventaja es que al hardware le toma menos tiempo el procesarlo, pero este tipo de iluminación no produce sombras.

Flat Shading Lighting (Iluminación de Sombreado Plano)

Consiste en que cada polígono represente un valor leve que se pase al polígono completo que genere una imagen plana del mismo, a esta imagen también se le asigna un color determinado.

- Vertex Shading (Sombreado de Vértice, Gouraud shading): solicita al motor de renderizado un color para cada vértice, luego por medio de interpolación se renderiza cada píxel por la distancia en relación con su respectivo vértice.
- Phong Shading: es similar al Gouraud Shading, trabajan con la textura, solo que el Phong Shading usa a los píxeles en lugar de los vértices.

El Phong Shading toma más tiempo de procesamiento que el Vertex Shading pero su resultados son mucho mejores en cuestión de suavizado de texturas.

- **Light Map Generation** (Generación del mapa de luz): se usa una segunda capa de textura (mapa de luz) que dará el efecto de iluminación a los modelos, es un efecto excelente pero debe tomarse antes del renderizado pero si se tienen Luces Dinámicas (o sea luces que se mueven, encienden o apagan sin intervención de programa) se debe estar regenerando los mapas en cada Frame de animación lo que toma mucha cantidad de memoria (pero son de render rápido).
- **Textura**: es esencial para que las escenas 3D se vean reales, en si las texturas son imágenes que se rompen en los distintos polígonos del modelo, muchas imágenes tomarán mucho espacio en la memoria por eso se debe usar técnicas de compresión:
- **Mapeo MIP:** consiste en preprocesar las texturas creando múltiples copias del mismo cada una la mitad del anterior, esto porque si la textura solo es pegada al polígono cada textura es a cada píxel y tomara más tiempo de render; así cada Texel (elemento de Textura) toma menos espacio.
- **Texturas Múltiples:** requiere múltiples renderizados por lo que para obtener buen resultado se necesita una tarjeta con Acelerador de Gráficos, provee mejor calidad que el simple mapeo. Se puede colocar una imagen sobre otra (más transparente) para dar el sentido de movimiento pulso o hasta sombra.
- **Bump Mapping**: técnica vieja de texturas que tratan de mostrar como la luz se refleja en el objeto. Solo hasta hace poco se vuelto a retomar.
- **Antialiasing**: El anti-aliasing revisa los polígonos y difuminará las bordes y vértices, para que los bordes no se vean como dentados. Esta técnica se puede hacer de dos maneras. La primera se realiza de modo individual, entremezclando polígonos para sobreponerlos unos delante de otros.

La segunda manera se hace por medio de tomar todo el marco y quitarle los bordes dentados, pero esto requiere de mucha memoria.

- **Vertex and Pixel Shaders** (Vértices y Sombreo de Pixeles): Con este método se pueden extraer y utilizar directamente las características y facilidades de la tarjeta de video, sin tener que utilizar mucho la API. Pero no es utilizable en todas las tarjetas.
- **Stencil Shadowing** (Plantilla de Sombreado): la idea es renderizar una vista de un modelo desde la perspectiva de la fuente de luz y después utilizar esto para

crear o para generar un polígono con la forma de esta textura sobre las superficies afectadas por el modelo. Así se obtiene una iluminación que parece real. Pero es costosa, porque usted está creando texturas “en vuelo”, y hace múltiple render de la misma escena.

El manejo del cache de textura es imprescindible para que el juego se desarrolle rápido (y para cualquier motor), ya que si se presenta un constante swapping de las texturas en la tarjeta el juego se verá lento y tedioso, algunos APIs descargan cada textura cuando esto pasa, pero eso haría que en cada cuadro se refresquen las texturas dando más lentitud. Todo se trata de cargar la menor cantidad de veces una misma textura, pero eso también depende del API que se utilice. Otra técnica es la compresión de texturas, comprimir texturas es como comprimir MP3, los algoritmos de compresión logran una relación 4:1 que no es mucho pero ayuda.

- **LOD** (level of detail, nivel de detalle): el sistema de nivel de detalle está relacionada con la complejidad geométrica de los modelos. Algunos sistemas necesitan que se hagan múltiples versiones del modelo, para que dependiendo de cuan cerca se esté del modelo así será su cantidad de polígonos. Otros sistemas ajustan dinámicamente esta característica pero en este caso da más carga al CPU
- **Depth Testing** (prueba de profundidad): Con esto se empieza a eliminar los píxeles ocluidos y se pone en práctica el concepto de sobre dibujado. La prueba de profundidad es una técnica utilizada para determinar que objetos están delante de otros en la misma localización del píxel.
- **Sobre Dibujado**: es la cantidad de veces que se ha dibujado un píxel en un frame. Se basa en la cantidad de elementos existentes en la tercera dimensión (profundidad).
- **Scripting Systems** (Sistemas de scripting):
- **Pre-scripted Cinematics**: usada normalmente en una situación que necesita la explicación en una manera controlada. Para presentar las escenas de la historia, ahora se utilizada el cortar-escenas que presenta la historia en vídeo digital y luego por medio de transiciones se pasa a las gráficas reales del juego.

El scripting le permite al diseñador tomar mando de la escena y manipularla, como colocar objetos o eventos que el jugador no controla. En muy complicado, se necesita de una mente muy metódica y lógica, la mayoría de estos scripts se basan en lenguaje C.

- **Visual Scripting Systems**: como lo dice su nombre, permite manejar el script en un ambiente gráfico en lugar de un código escrito, se maneja un carácter real en un ambiente del juego real.
- **Sonido**: Creative Labs ahora ha proporcionado sus extensiones manejadores de sonido EAX para DirectX, y la nueva iniciativa de OpenAL (biblioteca audio abierta). OpenAL, como suena, es un API para los sistemas de los sonidos de la misma manera que OpenGL es un API

Para el procesado de sonido es muy similar al procesado de los modelos, muchas veces un software los procesa antes de pasar al hardware respectivo, por ejemplo DirectSound hace al sonido para la Tarjeta de sonido lo que Direct3D hace al modelado antes de llegar a la Tarjeta 3D. Esto es llamado “premezcla” en el software

- **Music Tracks in Games** (pistas de audio): Hay dos formas de manejar el sonido. Uno es por medio de archivos .wav (o similares), lo cual emite un muy buen sonido, pero se requiere de mucha memoria. Por otro lado se pude utilizar archivos midi, esto reduce la necesidad de memoria, pero los sonidos no son tan buenos.
- **Inteligencia Artificial** (IA o AI): es la característica más importante que se le atribuye a un motor al lado de la representación de modelos o Render. AI provee de estímulo al juego. Es crítico en la parte de la forma de juego (game play).

La inteligencia artificial de determinado juego puede tornarse muy compleja, primero se debe definir la línea base del comportamiento de los NPC (Non Player Characters - Personajes que no son el Jugador), primero debe definirse que hace el NPC (patrulla, guarda, etc.), luego se delimita su "visión de mundo", que es lo es el NPC puede ver del mundo del juego; se debe tomar en cuenta que el personaje no solo estará en medio del mundo del juego sino que también interactuara con él, después vienen las rutinas de Toma de Decisión: si el NPC está patrullando, y hay un sonido, ¿debe tomarle importancia o no?, ¿investiga su origen o no?, etc.

Es un sistema de reglas para las acciones que responden (o inician) y que el jugador debe responder, esto a un concepto más general de AI.

- **Emergent game play** (Forma de Juego Emergente): consiste en programar al AI con un conjunto de reglas que le permitan al programa adherir situaciones que el programador no previera.

2.8 Herramientas adicionales utilizadas para la creación del videojuego.

Los presentes proyectos desarrollados constan de modelos 3D, texturas, sonido y video para lo cual se utilizaron los siguientes programas informáticos:

- Autodesk 3ds Max 2010 de 32 bits.
- Adobe Photoshop CS4 (64 Bit).
- Adobe Soundbooth CS4.
- Adobe Fireworks CS4.
- Camtasia Studio 7

Además de los plugins que fueron necesarios como Panda DirectX Max Exporter dentro de Autodesk 3ds Max 2010 y NVIDIA Photoshop Plug-ins para Photoshop.

Se notara qué todos estos programas son comerciales, en un principio se trató de utilizar software libre que cumplan con las mismas características de los antes citados, pero por facilidad de uso, la amplia documentación y una comunidad activa en internet se escogieron estos programas, sin menospreciar el software que puede ser encontrado en los repositorios de software libre que es muy bueno, pero se requería mayor conocimiento sobre algunas áreas, como por ejemplo:

El equivalente en software libre de un modelador 3D y que remplace a Autodesk 3ds Max 2010, la mejor opción Blender, pero para el uso de este software se requería conocimientos previos y avanzado cuanto a modelado 3D, lo cual puede ser fácilmente comprobado, no solo al utilizar este software como usuario inexperto sino también en comentarios en foros encontrados sobre este tema, aunque la principal ventaja aparte del costo entre el software comercial y el libre, es el tamaño de la aplicación que es muy baja en comparación con otras herramientas. Por estas razones desde el principio se descartó la utilización de esta aplicación dentro del proyecto.

Como resultado y como se puede constatar no se utilizó ningún paquete de software libre, como se aclaró anteriormente la falta de conocimiento sobre la utilización de software multimedia, y demás dificultades que se afrontó, se decidió en utilizar software que ya tenía un mercado de consumidores activos y con gran cantidad de información que puede ser obtenida a través de internet.

La herramienta más importante fue Autodesk 3ds Max 2010, y según lo investigado algunos de los videojuegos exitosos en el mercado fueron hechos con este programa incluido algunas películas. Aunque los modelos de los personajes utilizados dentro del proyecto desarrollado no son muy detallados, debido principalmente a la complejidad del diseño de estos “no del programa”, en futuros proyectos se espera que a través de un equipo de diseñadores conseguir mejores resultados, como se mencionó antes, lo más general en el desarrollo de videojuegos es tener grupos que se encarguen de cada actividad.

La reproducción de sonido dentro de XNA está a cargo de una herramienta colaborativa de XNA llamada XACT, que a través de bancos de sonido se los puede reproducir en el proyecto, también permite la aplicación de efectos a los sonidos como reverberación, reproducción en bucle, volumen de cada sonido, sonido 3D, etc.

XNA permite la reproducción de sonido, pero solo de archivos de sonido en formato .wav, .wma, .mp3 que a través de la herramienta anteriormente citada se los puede integrar con los proyectos de XNA, por este motivo se utilizó programas adicionales para convertir archivos de sonido que se pudo obtener de fuentes externas en archivos legibles para el proyecto para posteriormente incorporarlos al videojuego.

Adobe Photoshop CS4 (64 Bit) y Adobe Fireworks CS4 reconocidos programas para la creación y edición de imágenes de una manera profesional, Camtasia Studio 7 para la creación y edición de videos.

CAPITULO III.

ANÁLISIS DEL PROYECTO

3.1 Análisis de las herramientas a utilizar

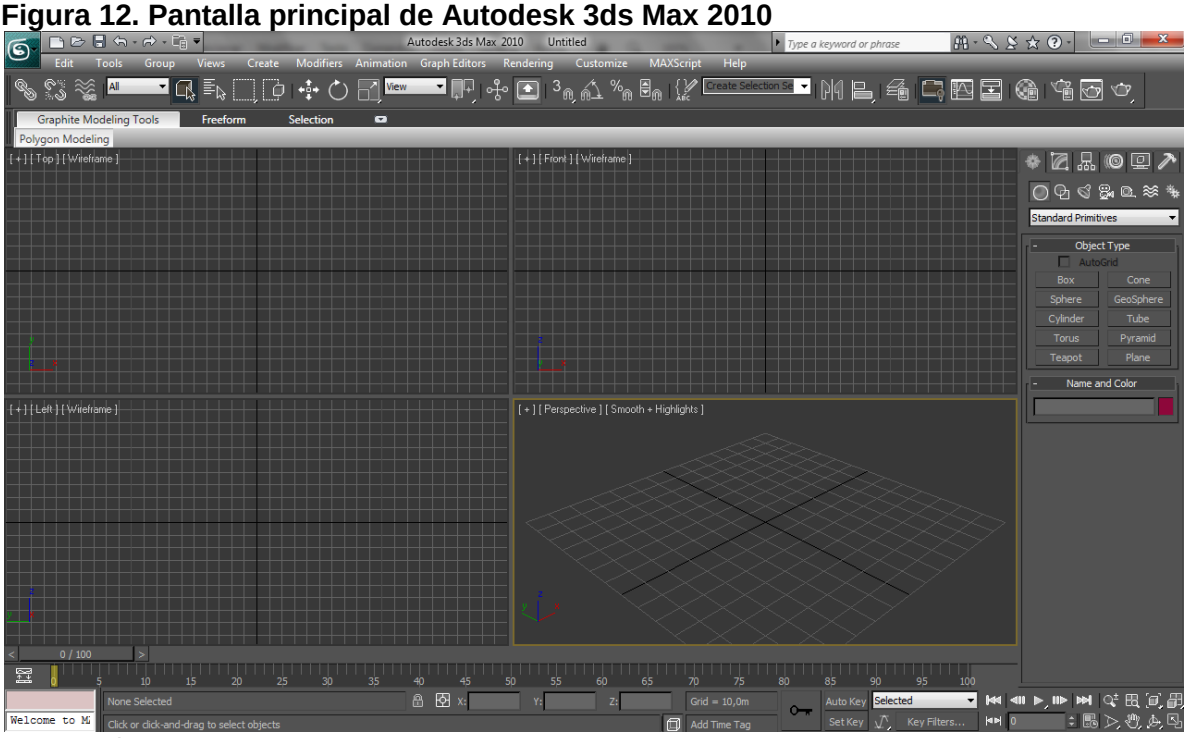
En la actualidad existe dos tipos de herramientas de software, las comerciales y las de uso libre, diferenciado principalmente por el valor que se paga por el uso de dicho software.

En el desarrollo del presente proyecto desde un principio se adoptó tecnologías de Microsoft ya que al ser XNA un producto del mismo lo más recomendable fue utilizar productos y estándares de dicha empresa, partiendo de este antecedente se utilizó DirectX que es el único API gráfico soportado por este framework.

3.1.1 Herramientas comerciales

Para el desarrollo del presente proyecto el modelador 3D que se utilizó fue Autodesk 3ds Max 2010 que dentro de las características más importantes y lo que fue considerado al momento de la elección fue el hecho de que este modelador es utilizado por algunas empresas dedicadas al desarrollo de videojuegos algunos de ellos de gran éxito en el mercado, no solo es utilizado para el modelado 3D, también puede ser utilizado para la creación de películas, texturas con efectos especiales, y demás contenidos multimedia.

Figura 12. Pantalla principal de Autodesk 3ds Max 2010



Fuente: El autor

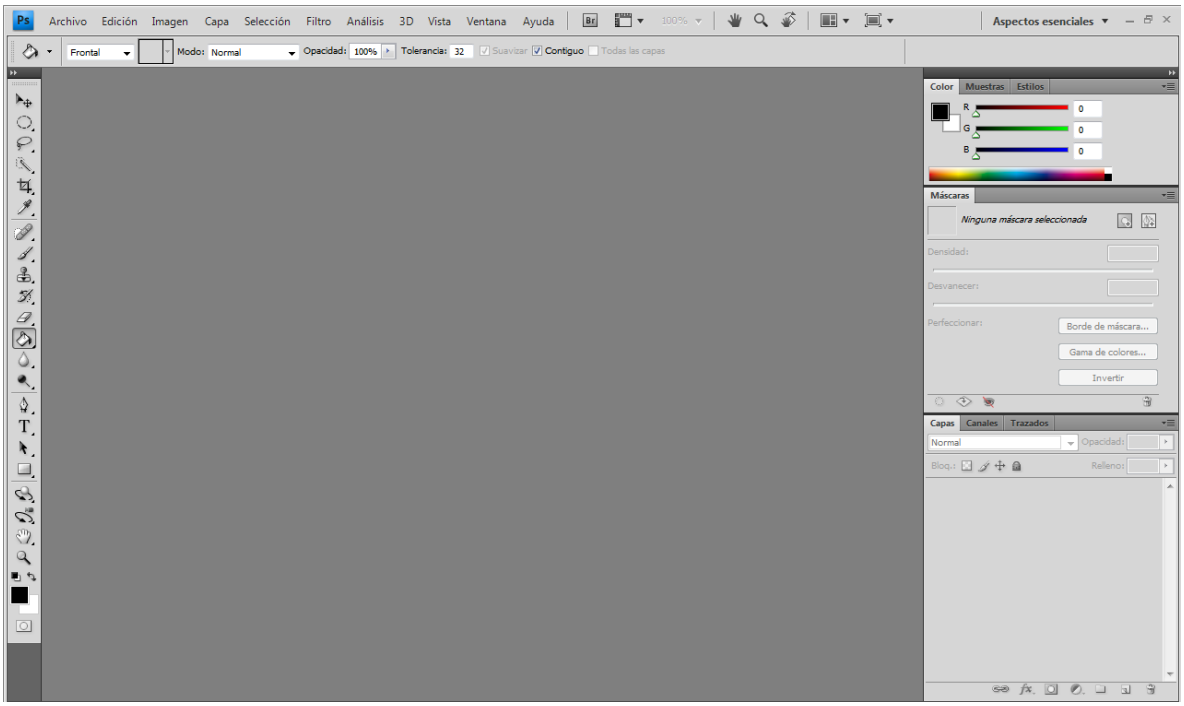
Dentro del desarrollo de videojuegos es imprescindible un modelador 3D que facilite el diseño de personajes, escenarios y ambientes propios de este tipo de software, de entre los diferentes programas informáticos y los más importantes son Autodesk 3ds Max 2010, Maya, Milkshape y Blender.

Desde la primera vez que se utilizó este programa y con los escasos conocimientos de modelado 3D, su aprendizaje se volvió sencillo e intuitivo, gracias a la facilidad de uso del programa.

El aprendizaje de este software de modelado 3D se tornó más fácil debido a la amplia documentación encontrada a través de internet y a una comunidad activa que puede ser de ayuda en algunos de los problemas que puede presentarse a la hora de diseñar, independientemente del propósito de la utilización de este paquete informático.

La edición de imágenes y texturas se lo realizó con dos paquetes informáticos de la suite de Adobe: Adobe Photoshop y Adobe Fireworks

Figura 13. Pantalla principal de Adobe Photoshop CS4 (64 Bit).



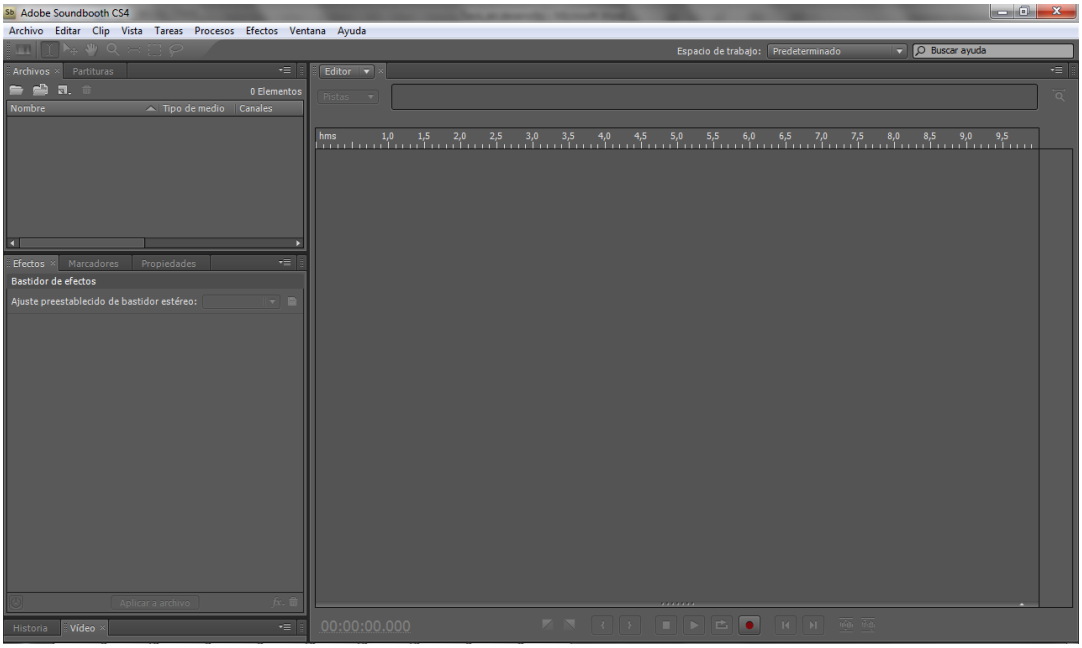
Fuente: El autor

Adobe Photoshop utilizado en su mayoría para el realce de las texturas utilizados por algunos de los modelos 3D y también como visualizador de dichas imágenes, ya que algunos de los formatos de las imágenes solo podían ser vistos con este programa o en su defecto instalando plugins adicionales que hacen más extensible el programa. El único plugin instalado fue el de NVIDIA para importar y exportar imágenes con formato de directX .DDS

Adobe Fireworks como complemento de Photoshop para la edición de imágenes que a lo largo del desarrollo del videojuego se lo utilizaba como por ejemplo: las texturas de las armas, el objetivo hacia donde apunta el arma del jugador, y demás texturas 2D.

Adobe Soundbooth de la misma suite de Adobe pero para propósito de edición y creación de audio.

Figura 14. Pantalla principal de Adobe Soundbooth.



Fuente: El autor

El formato soportado por XACT y que puede ser modificado y reproducido en la aplicación a desarrollarse es .wav, que es donde intervino la utilización de este programa, se lo utilizó para la normalización de todos los sonidos utilizados y el cambio de formato, cabe destacar que los sonido utilizados fueron obtenidos a través de internet e incluso de otros videojuegos, haciendo un poco más fácil el desarrollo del sistema.

La falta de tiempo y el desconocimiento del uso de paquetes informáticos para la producción de sonido fueron los impedimentos para que los sonidos no sean propios de los videojuegos a desarrollarse.

Por último para la creación de video y captura de sonidos esta Camtasia Studio 7.

Figura 15. Pantalla de inicio de Camtasia Studio

Fuente: El autor

3.1.2 Software Libre

Uno de los principales competidores dentro del software de modelado 3D esta Blender, de código abierto y actualmente es compatible con todas las versiones de Windows, Mac OS X, Linux, Solaris, FreeBSD e IRIX.

Figura 16. Logo Blender

Fuente: Wikipedia

La principal ventaja es el tamaño del paquete informático que es muy pequeño en comparación con herramientas comerciales, pero como desventaja es la dificultad por parte de usuarios inexpertos en la utilización de dicho programa, con una interfaz poco intuitiva y demás defectos de presentación. Pero para usuarios avanzados los resultados que se pueden conseguir no tienen nada que envidiar a herramientas comerciales.

El software encontrado a través de los repositorios de software libre es muy bueno dependiendo del objetivo para el que es utilizado y de los gustos de quien esté desarrollando un proyecto.

Las herramientas utilizadas para el desarrollo del presente proyecto fueron variadas, algunas de las cuales por su sencillez no constan dentro de la redacción de este documento.

3.1.3 Conclusión

La idea misma de desarrollar un videojuego es todo un reto, esta actividad comprende varios componentes indispensables para realizar un producto de calidad. Más aun el proyecto desarrollado que obviamente no estará a la altura de productos comerciales desarrollados por empresas internacionales, se utilizaron herramientas comerciales que satisfagan algunas de las necesidades del proyecto, sin menospreciar el software libre que también es muy bueno, pero por facilidad de uso y aunque era poco el conocimiento inicial sobre la creación de videojuegos no se podía invertir más tiempo en la investigación de otros paquetes informáticos que suplanten los que ya se conocía.

Como se mencionó anteriormente la principal razón de optar por software comercial fue el tiempo que se iba a dedicar en el desarrollo, diseño, efectos especiales, sonido, video y demás componentes, que como se puede observar son muchos como para ser afrontados por una sola persona, y por último la razón del desarrollo de este proyecto fue en un principio presentar una investigación sobre cómo desarrollar videojuegos, que luego y durante el desarrollo de este, se decidió incorporar al sistema, componentes propios necesarios para elaborar este tipo de software.

3.2 Descripción de la solución

Los proyectos desarrollados se presentaron como investigación sobre un framework destinado para este fin, dando como resultado videojuegos no de calidad pero si aceptable dentro del término de lo que significa un videojuego.

3.2.1 Género de videojuegos

Los videojuegos se pueden clasificar como un género u otro dependiendo de su representación gráfica, el tipo de interacción entre el jugador y la máquina, la ambientación y su sistema de juego, siendo este último el criterio más habitual a tener en cuenta.

Cada vez es más es habitual que un videojuego contenga elementos de diversos géneros, cosa que hace difícil su clasificación.

3.2.1.1Acción

Beat 'em up

Llamados "videojuegos de lucha a progresión" son videojuegos similares a los de lucha, con la diferencia de que en este caso los jugadores deben combatir con un gran número de individuos mientras avanzan a lo largo de varios niveles. En los beat'em up suele ser posible jugar dos o más personas a la vez de forma cooperativa para facilitar el progreso.

Lucha

Los videojuegos de lucha, como indica su nombre, recrean combates entre personajes controlados tanto por un jugador como por la computadora. El jugador ve a los combatientes desde una perspectiva lateral, como si se tratase de un espectador. Este tipo de videojuegos ponen especial énfasis en las artes marciales, reales o ficticias (generalmente imposibles de imitar), u otros tipos de enfrentamientos sin armas como el boxeo o la lucha libre. Otros videojuegos permiten también usar armas blancas como pueden ser espadas, hachas, martillos, etc. o ataques a distancia, normalmente de carácter mágico o etéreo.

Disparos

Disparos en primera persona

En los videojuegos de disparos en primera persona, conocidos también como FPS, las acciones básicas son mover al personaje y usar un arma, un arma se anuncia en la pantalla en primer plano y el jugador puede interactuar con éste. Esta perspectiva tiene por meta dar la impresión de estar detrás del personaje y así permitir una identificación fuerte (Perspectiva de primera persona). Las gráficas en tres dimensiones aumentan esta impresión.

Disparos en tercera persona

Los videojuegos de disparos en tercera persona, conocidos también como TPS, se basan en el alternamiento entre disparos y pelea o interacción con el entorno, pero a diferencia de los juegos de mira (primera persona), se juega con un personaje visto desde atrás y en ocasiones, desde una perspectiva isométrica.

Estos videojuegos sacrifican precisión por una gran libertad de movimientos.

Sigilo

Estos videojuegos se basan en el sigilo, la furtividad y la estrategia en vez de buscar la confrontación directa con el enemigo.

Plataformas

En los videojuegos de plataformas el jugador controla a un personaje que debe avanzar por el escenario evitando obstáculos físicos, ya sea saltando, escalando o agachándose. Además de las capacidades de desplazamiento como saltar o correr, los personajes de los juegos de plataformas poseen frecuentemente la habilidad de realizar ataques que les permiten vencer a sus enemigos, convirtiéndose así en juegos de acción.

3.2.1.2Simulación

Este género se caracteriza en marcar un aspecto de la vida real, llevada a un juego, donde tienes total control de lo que pasa. En muchos de ellos se enfocan tanto en inmiscuir al jugador hasta hacerlo creer que lo que está pasando es real, sobre todo con los géneros de simulación de combate o de pilotaje

Simulación musical

Su desarrollo gira en torno a la música ya sean de tipo karaoke o en los que se debe bailar.

Simulación de combate

Género poco llevado a la práctica que se caracteriza por el elevado realismo en todos los aspectos relevantes en cuanto al desarrollo de las partidas.

Simulación de construcción

Género muy popular en PC, donde el programa le proporciona al usuario todas las herramientas para construir un proyecto, el cual debe ser lo más apegado a la realidad como sea posible, en el cual se consideran desde gastos de construcción y mantenimiento, hasta una línea de tiempo, física y clima que afecta todas las decisiones que tomes. La particularidad de poder experimentar, tomar decisiones y afectar el desempeño de tu simulación, los hace tremendamente adictivos.

Simulación de vida

Tomando el mismo concepto del género, el simulador de vida se enfoca en controlar un personaje con capacidades y emociones humanas, y controlar todos los aspectos de su vida, desde donde vivirá, que estudiará, con quien se casará.

Arcade

Los videojuegos arcade, se caracterizan por la simplicidad de acción rápida de jugabilidad, esto obtuvo la gloria en la época de 1980. No requiere historia, solo juegos largos o repetitivos donde la prioridad es entretener al jugador.

Deporte

Los videojuegos de deporte son aquellos que simulan juegos de deporte real, entre ellos encontramos, golf, tenis, fútbol, hockey, juegos olímpicos, etc.

Carreras

Principalmente son videojuegos que se dedican a comenzar de un punto y llegar a una meta antes que los contrincantes. Juegos así se han desarrollado de su forma común en vehículos, hasta otras formas como juego de plataformas. La idea principalmente es competir en llegar primero, y algunas veces se suelte ampliar este concepto, originando herramientas y trampas para la carrera.

Agilidad mental

Estos son juegos donde se tiene que pensar y agilizar el pensamiento. El objetivo aquí es resolver ejercicios con dificultad progresiva para desarrollar la habilidad mental.

Educación

Los juegos educativos son aquellos que enseñan mientras promueven diversión o entretenimiento (Didactismo).

3.2.1.3Aventura

Aventura conversacional

El jugador encarna a un protagonista que por lo general debe resolver incógnitas y rompecabezas con objetos diversos. En estos, el jugador utiliza el teclado para introducir órdenes como “coger la cuerda” o “ir hacia el oeste” y el ordenador describe lo que pasa.

Aventura gráfica

A comienzos de los 1990, el uso creciente del ratón dio pie a los juegos de aventura de tipo "Point and clic", también llamados aventura gráfica, en los que ya no se hacía necesaria la introducción de comandos. El jugador puede, por ejemplo, hacer clic con el puntero sobre una cuerda para recogerla.

Rol

Se caracterizan por la interacción con el personaje, una historia profunda y una evolución del personaje a medida que la historia avanza.

En otra subcategoría de los videojuegos de rol se han puesto de moda los RPG en línea o MMORPG (Massive Multiplayer Online RPG) donde cada jugador crea un personaje y mediante una conexión a internet, entra a un mundo donde miles de jugadores se unen a la aventura, exploran, intercambian y evolucionan juntos.

El proyecto desarrollado consta de 3 partes derivados del tema principal *"Desarrollo de un videojuego con texturas y modelos 3D, multiusuario, en un entorno cliente/servidor (internet) utilizando el framework de Microsoft XNA."* que se especifican a continuación.

- Videojuego para un solo jugador.
- Videojuego multijugador con soporte de red.
- Videojuego multijugador a través de internet.

Todas estas partes constas de su respectiva descripción realizada en la presentación del proyecto ante el Honorable Consejo Académico y que vale citarlos a continuación.

Propuesta para un solo jugador:

El video juego para este modo será al estilo de la saga **Grand Theft Auto**

Descripción:

- Género: Acción en tercera persona.
- Modos de juego: Un jugador.
- Numero de misiones: 10.
- Niveles de dificultad: Cada escenario o misión del videojuego tendrá su propio nivel de dificultad, especificado por cada misión.
- Historia: La trama empieza con un grupo de tres ladrones atacando una sucursal de un Banco (ficticio). Un hombre y una mujer van en cabeza mientras que otro les cubre las espaldas. Sin embargo, al doblar una esquina, el tercer ladrón es disparado por la mujer, quien huye con el otro hombre y el dinero. El traicionado llamado Miguel (ficticio) ha de hacérselo pagar a los traidores: María (ficticio) ex novia de Miguel, y a Luis, que son cabecillas de un cartel colombiano afincado en Ibarra. Las misiones a realizar serán para atrapar a los traidores. (Trama de GTAIII adaptada).

Propuesta de videojuego multijugador con funcionalidad a través de una red LAN

El video juego para este modo será al estilo de la saga **Quake.**

Descripción:

- Género: Acción en primera persona.
- Modos de juego: Multijugador.
- Número de conexiones (jugadores) máximo: 4.
- Objetivo: El juego permitirá a los jugadores, cuyas computadoras están conectadas a una red o a Internet, disputar partidas entre sí, en tiempo real. Usa una estructura de arquitectura cliente-servidor que requiere que los clientes de todas las computadoras de los jugadores se conecten a un solo servidor. El objetivo en *Quake* es moverse a través de todo el campo de batalla eliminando (*fragueando*, del inglés *frag*) a los jugadores enemigos y anotándose puntos basándose en los objetivos del tipo de juego. Cuando los puntos de vida de un jugador llegan a cero, el avatar del jugador es eliminado (*fragueado*); luego el jugador reaparece en otro punto del mapa y sigue jugando con sus puntos de vida restaurados, pero sin las armas ni items que recolectó anteriormente. El juego termina cuando un jugador o equipo alcanza un puntaje específico, o cuando se termina el tiempo. (Objetivo de Quake III Arena).

Propuesta de videojuego multijugador con funcionalidad a través de internet

Será la misma presentada con funcionalidad a través de una red LAN, con cambios en la comunicación con el servidor de juegos, obviamente por la topología que utiliza este tipo sistema con salida hacia internet. Pero la jugabilidad será la misma.

Nota: Debido a la complejidad de los proyectos a desarrollarse y cabe resaltar que fueron presentados como propuestas más no se presentaron como directrices a cumplirse en su totalidad, la historia y los objetivos de los videojuegos planteados cambiaron a lo largo del desarrollo del proyecto.

La historia y los objetivos de los videojuegos cambiaron con respecto a lo que inicialmente se propuso, afectando así la jugabilidad, pero esto no quiere decir que los objetivos del proyecto hayan cambiado.

Con la respectiva aclaración, el presente capitulo muestra cómo se desarrolló la aplicación como parte de la solución al tema presentado como proyecto de tesis.

3.2.2 Proceso lógico para el desarrollo del sistema.

Lo más importante dentro del desarrollo de un sistema de este tipo es construir o tener un motor de videojuego que viene a ser el núcleo del software que maneja todos los

componentes necesarios para la ejecución del juego en un ambiente de hardware específico y que provee la comunicación entre estos.

En otras palabras es la parte del videojuego que se encarga de abstraer la complejidad del hardware y permite al desarrollador preocuparse por los detalles que hacen del videojuego único y un reto que debe ser asumido.

Dentro de la comunidad de codeplex existen varios motores (Engines) que pueden ser utilizados para facilitar las tareas al desarrollador, no obstante, estos están desarrollados para personas con conocimientos avanzados en el desarrollo de videojuegos, por lo cual no se utilizó uno de los ya existentes.

Cabe resaltar que XNA es un framework que ayuda al desarrollo de videojuegos, mas no es un motor de videojuegos, por lo cual fue necesario el desarrollo y construcción de un Game Engine propio, que al principio la concepción de realizar esto presenta algunas ventajas y desventajas citadas a continuación.

Ventajas:

- Conocimiento completo de cómo está construido el motor de videojuegos, haciendo fácil de mantener y en futuro extensible.
- Cualquier cambio en el motor seria de fácil implementación debido a lo especificado anteriormente.
- Cualquier error presentado en la ejecución del proyecto será única y exclusiva responsabilidad del desarrollador, mas no de los componentes utilizados.
- El código fuente será de uso libre y sin restricciones para quien lo desee utilizar en proyectos propios a excepción de los componentes utilizados que no fue posible la obtención de los códigos fuentes de donde fueron descargados.

Desventajas:

- Debido al poco conocimiento acerca del desarrollo de videojuegos la construcción del motor se volvió una tarea difícil y ardua dando como resultado la inversión de mayor tiempo en la construcción del motor antes que el desarrollo del videojuego.
- Demasiada complejidad en incorporar módulos propios del videojuego y que se comuniquen con otros.
- El resultado de la construcción de un motor propio no es el de un engine profesional y con características avanzadas de videojuegos comerciales, pero esta desventaja se la puede mitigar en un futuro por las ventajas presentadas anteriormente y con el conocimiento adquirido durante el desarrollo del mismo.

Para iniciar la descripción de cómo fue desarrollado el sistema es necesario presentar los requerimientos básicos de un motor de videojuegos y que se supone que no deben ser diferentes de los videojuegos comerciales, estos son los siguientes:

- Manejador de Pantallas
- Manejador de Entrada y Salida
- Manejador de Sonido
- Manejador de Video
- Manejador de Sistema de Partículas

- Manejador de la Red (Videojuego multijugador.)
- Manejador de Cámaras
- Manejador de niveles
- Manejador de entornos y escenarios propios de cada nivel.
- Manejador de actores que intervienen en el videojuego.

Estos requerimientos son los básicos y que fueron implementados en el sistema desarrollado. En un videojuego comercial realizado por empresas estos requerimientos pueden ser más y de mayor cobertura con mayor complejidad.

3.2.3 Términos usados en el desarrollo de videojuegos

Antes de comenzar con la descripción de cómo se realizó la implementación de los requerimientos citados anteriormente es necesario aclarar algunos de los componentes utilizado en el desarrollo del proyecto.

Básicamente esta sección se centrara en la definición y descripción de cómo funcionan los videojuegos en 3D a través de la visión de una cámara, un poco de matemática aplicada en el desarrollo del proyecto y por último las características de un dispositivo gráfico y como XNA administra dicho dispositivo.

Siguiendo un orden específico y que se cree es el más lógico, a continuación se describirá lo que es una cámara.

Cámara de videojuegos

Algunos de los videojuegos exitosos hasta ahora y los primeros en ser comercializados fueron hechos con gráficos 2D, sin embargo, en la actualidad, las tarjetas gráficas y procesadores, ofrecen más ventajas en el área de gráficos 3D.

La primera diferencia entre los gráficos 3D y los 2D es la adición de una dimensión extra, programar juegos en 2D es muy similar a pintar un dibujo sobre un lienzo: se pinta en dos dimensiones siendo las coordenadas (0, 0) la esquina superior izquierda de la pantalla, y X movimientos positivos hacia la derecha, mientras Y movimientos positivos hacia abajo.

Si dibujar en 2D es como pintar un dibujo, dibujar en 3D es más como grabar un video con una cámara de video. Las coordenadas en 3D están basadas sobre un sistema de coordenadas en tres dimensiones. Este sistema de coordenadas, a veces referido como mundo 3D, el centro alrededor del origen están en la coordenada (0, 0, 0). Sin embargo en los juegos 3D existen 2 componentes básicos de escena: situando objetos en el mundo, y situando una cámara en el mundo y especificando la dirección hacia a donde apunta. Solo los objetos que la cámara está viendo son visibles en la pantalla.

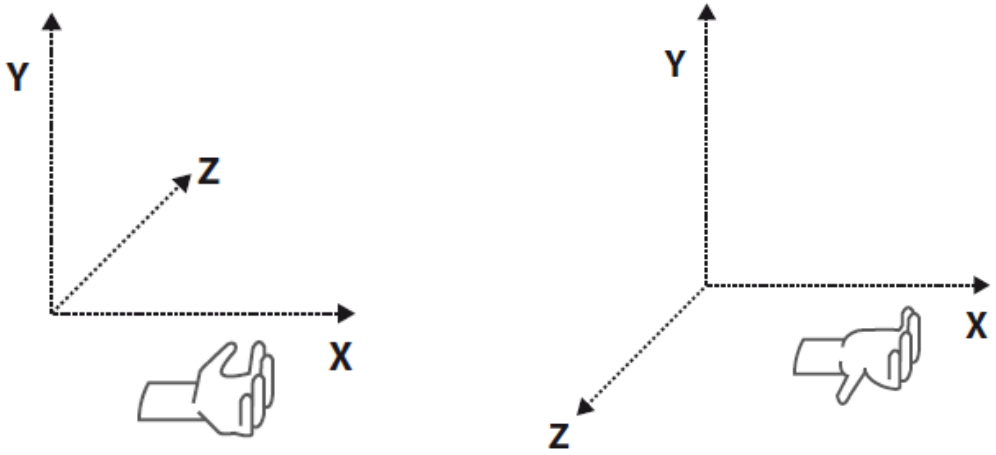
Dependiendo de donde está la cámara y en qué dirección está apuntando, un objeto que se está dibujando en el origen en un juego 3D puede ser visible en la mitad de la pantalla, en la parte inferior de la pantalla o en la pantalla completa.

Existen dos tipos diferentes de sistemas de coordenadas 3D, cada uno con el eje Z moviéndose positivamente en la dirección opuesta. La dirección en que el eje Z se mueve

positivamente determina la orientación. Las dos posibles orientaciones son: sistema de coordenadas de la mano izquierda y mano derecha.

Un camino para distinguir entre un sistema de coordenadas de la mano izquierda y un sistema de coordenadas de la mano derecha es poner las manos, palmas hacia arriba, con los dedos apuntando en la dirección del eje X positivo y curvando hacia arriba positivamente en Y. La dirección en la cual el pulgar apunta indica la dirección positiva en Z.

Figura 17. Sistema de coordenadas



Mano-izquierda
Sistema de coordenadas

Mano-izquierda
Sistema de coordenadas

Fuente: El autor

XNA usa el sistema de coordenadas de la mano derecha, esto quiere decir que cuando se esté viendo el origen desde un ángulo donde X movimientos positivos hacia la derecha y Y movimientos positivos hacia arriba, Z movimientos positivos hacia la persona que esté viendo la pantalla.

Ahora que ya está explicado el sistema de coordenadas utilizado por XNA, se describe lo que es una cámara, como se menciona previamente, dibujar una escena 3D se representa más por la grabación de una película con una cámara de video. Se tiene definido donde está localizada la cámara, qué está apuntando y varias otras propiedades,

Estas propiedades son guardadas en un objeto de tipo matriz. Cabe destacar que las matrices son el corazón de algunas de las cosas que se puede hacer con gráficos 3D. Afortunadamente XNA hace que la definición y utilización de matrices sea más fácil ya que una cámara puede representarse con una matriz o dos.

Los dos objetos de tipo matriz que se utiliza para la construcción de una cámara en XNA son: la vista y proyección, la matriz vista contiene información que determina dónde está la cámara en el mundo, que dirección está apuntando, y en qué dirección esta. La matriz proyección contiene información que determina propiedades de la cámara basados en el ángulo de la vista, que tan lejos la cámara puede ver. Estas matrices representan la transformación desde el mundo 3D al plano 2D de la pantalla.

Para crear la matriz vista se usa métodos estáticos provistos por la clase matriz llamada CreateLookAt, el cual retorna una matriz y acepta los siguientes parámetros.

Parámetro	Tipo	Descripción
cameraPosition	Vector3	Coordenada de la posición de la cámara.
cameraTarget	Vector3	Coordenada del punto hacia donde la cámara está viendo.
cameraUpVector	Vector3	Vector que indica la dirección hacia arriba.

Se notará que los primeros dos parámetros no necesitan mayor descripción, pero el tercer parámetro se indica el lado en que la cámara está viendo en comparación con una cámara de video, puede ser hacia arriba o hacia abajo.

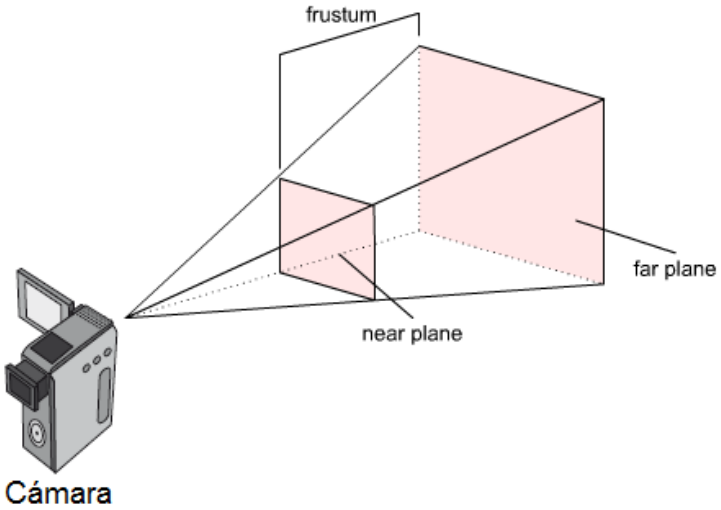
Para crear la matriz proyección se usa métodos estáticos provistos por la clase matriz llamada CreatePerspectiveFieldOfView, el cual también retorna una matriz y acepta los siguientes parámetros.

Parámetro	Tipo	Descripción
fieldOfView	Float	Angulo de vista de la cámara en radianes, típicamente es 45 grados o π/4
aspectRatio	Float	Aspecto de radio de la cámara. Comúnmente el ancho de la pantalla dividido por el alto de la pantalla.
nearPlaneDistance	Float	Que tan cerca un objeto puede ser visto con relación a la localización de la cámara.
farPlaneDistance	Float	Que tan lejos un objeto puede ser visto por la cámara.

La matriz proyección construye lo que se llama viewing frustrum o el campo de visión para la cámara. Esencialmente define que es visible en una área 3D y si puede ser dibujada en pantalla.

En el siguiente gráfico se puede observar como una cámara funciona hipotéticamente.

Figura 18. Campo de visión en tres dimensiones.



Fuente: El autor

Dependiendo del tipo de juego la implementación de una cámara puede tener variantes, las utilizadas dentro del proyecto fueron las siguientes:

- Cámara en tercera persona para la versión para un solo jugador.
- Cámara en primera persona para la versión multijugador.
- Cámara libre.

Dentro del manejador de cámaras esta la implementación de dichas cámaras a código fuente para la utilización dentro del proyecto y cuya explicación de la funcionalidad de dicha cámara sigue a continuación más no del código.

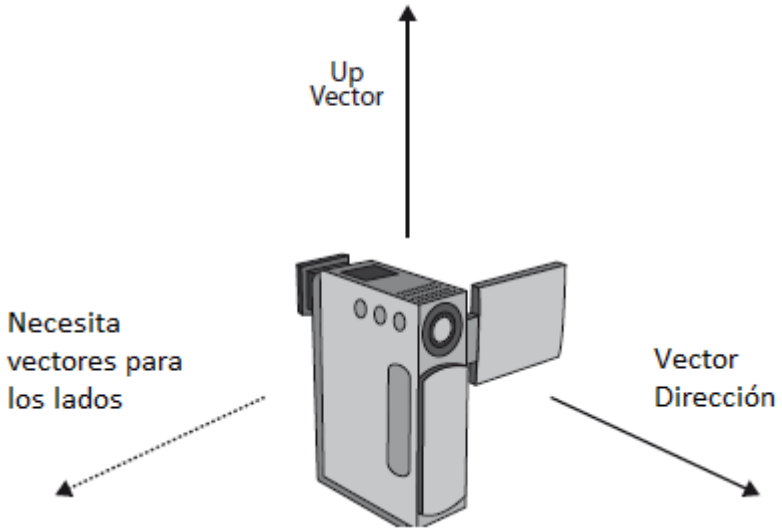
La cámara en tercera persona por lo general está ubicada detrás del personaje que es controlado por el jugador y que persigue a dicho jugador, esta implementación presenta algunas desventajas con relación a otras cámaras, como por ejemplo, el campo de visión no es total como una cámara en primera persona, así que no se podrá observar con más detalle el entono del jugador, además si un objeto del escenario se interpone entre el jugador y la cámara, este ocultara al jugador, pero existen algoritmos para resolver este problema como lo podría ser: haciendo transparentes los objetos que se interponen, algoritmos que esquiven los objetos que se atraviesan entre la cámara y el jugador sin perder de vista al jugador.

Los anteriores problemas citados si se presentaron en el proyecto pero no se pudieron resolver debido a la complejidad de implementarlos, si bien la concepción de cómo resolver el problema es sencillo pero la implementación de dicha solución resulto muy compleja, así que si un objeto se interpone entre el jugador y la cámara el jugador deberá moverse de tal manera que este siempre visible.

Para las versiones multijugador se utilizó una cámara en primera persona, lo que implica que no se verá al jugador pero se verá el arma que tiene dicho jugador, este tipo de cámara más lo utilizan los videojuegos de disparos, género en el cual encaja la versión multijugador, al igual que una cámara en tercera persona esta presenta algunas desventajas como: no se puede observar al jugador y la interacción del mismo con el mundo del videojuego, etc.

Y por último la cámara libre se utilizó para ubicar los objetos de juego en el escenario, cabe destacar que este tipo de cámara no representa ningún tipo de jugabilidad para el jugador, sino más bien se lo utilizó para propósitos de diseño en caso de la construcción de los escenarios de juego y depuración en el caso de corregir errores en la implementación de las otras cámaras.

Figura 19. Representación de los componentes de una cámara



Fuente: El autor

3.2.4 Matemáticas aplicadas en el desarrollo de videojuegos.

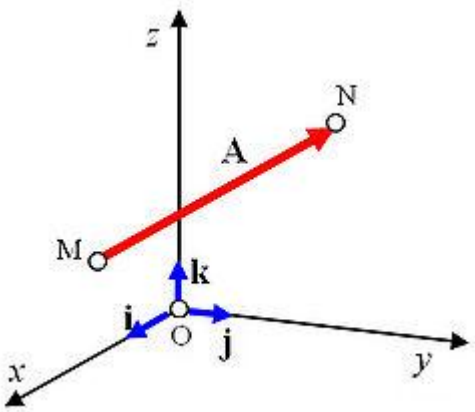
XNA provee de métodos correspondientes a ciertas estructuras utilizadas dentro del desarrollo de videojuegos lo cual facilita enormemente el desarrollo de aplicaciones bidimensionales o tridimensionales, con respecto a esto, las estructuras utilizadas dependen del tipo de aplicación a desarrollarse sea en 2D donde a lo largo de todo el desarrollo se utilizara vectores de dos dimensiones X y Y junto con matrices donde sea necesario su utilización, a diferencia de aplicaciones 3D donde prima la utilización de vectores 3D cuyas dimensiones son X, Y y Z y que en algunos casos también se utiliza vectores 2D.

En esta sección se definirá las estructuras matemáticas utilizadas y los métodos más importantes de cada uno de estos.

- **Vectores**

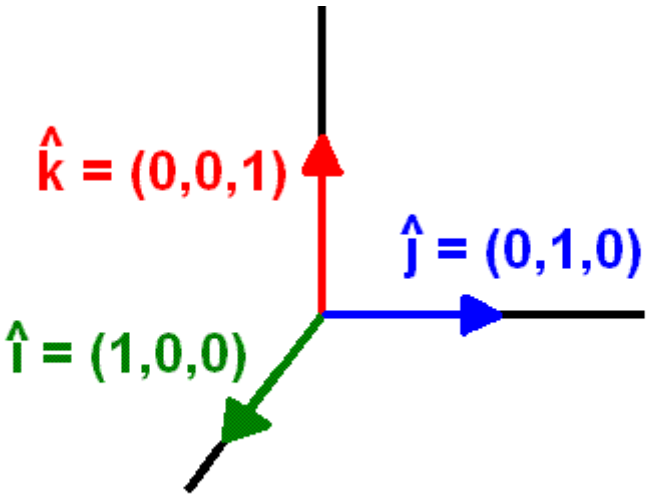
Un vector es todo segmento de recta dirigido en el espacio, el cual presenta:

  - Un origen.
  - Un módulo o longitud.
  - Una dirección u orientación de la recta que lo contiene.
  - Un sentido que indica hacia qué lado se dirige.



Vectores bases unitarios

Son vectores que tienen módulo 1. Para representar el sistema cartesiano, se la hará mediante vectores unitarios, que corresponden a cada uno de los ejes del sistema de referencia, y por lo tanto son unidimensionales.



Componentes de un vector.

Cualquier vector puede ser representado como resultado de la suma de sus 3 vectores unidimensionales, cada uno de ellos en la dirección de sus ejes cartesianos.

$$\vec{r} = \vec{r}_x + \vec{r}_y + \vec{r}_z$$

Si se considera ahora los vectores unitarios y se nota que la longitud del vector es sólo un escalar podemos sustituir cada uno de los sumandos de la expresión anterior por el producto de un escalar por el correspondiente vector unitario.

$$\vec{r}_x = \alpha\vec{u}_x;\ \vec{r}_y = \beta\vec{u}_y;\ \vec{r}_z = \gamma\vec{u}_z$$

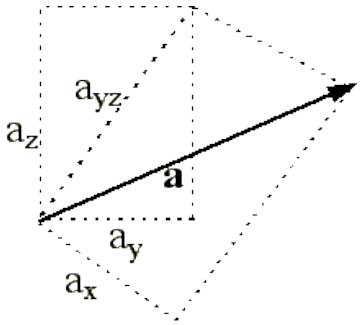
Por lo que:

$$\vec{r} = \alpha\vec{u}_x + \beta\vec{u}_y + \gamma\vec{u}_z$$

Módulo de un vector

Un vector no solo nos da una dirección y un sentido, sino también una magnitud.

Gráficamente: es la distancia que existe entre su origen y su extremo.



Aplicando Pitágoras:

$$|a| = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

Vectores unitarios

Un vector unitario tiene de módulo la unidad.

La normalización de un vector consiste en asociarle otro vector unitario, de la misma dirección y sentido que el vector dado, dividiendo cada componente del vector por su módulo.

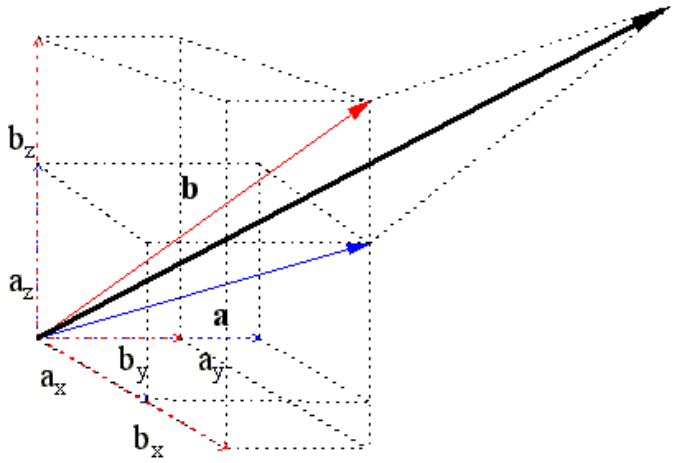
$$\vec{u}_{\vec{v}} = \frac{\vec{v}}{|\vec{v}|}$$

Por lo que el vector puede representarse de la siguiente forma:

$$\vec{r} = (x, y, z)$$

$$\vec{r} = x\vec{\imath} + y\vec{\jmath} + z\vec{k}$$

Suma y resta de vectores



$$a + b = \left(a_x i + a_y j + a_z k\right) + \left(b_x i + b_y j + b_z k\right) = (a_x + b_x)i + \left(a_y + b_y\right)j + (a_z + b_z)k$$

$$\vec{a} = (x_1, y_1, z_1) \qquad \vec{b} = (x_2, y_2, z_2)$$

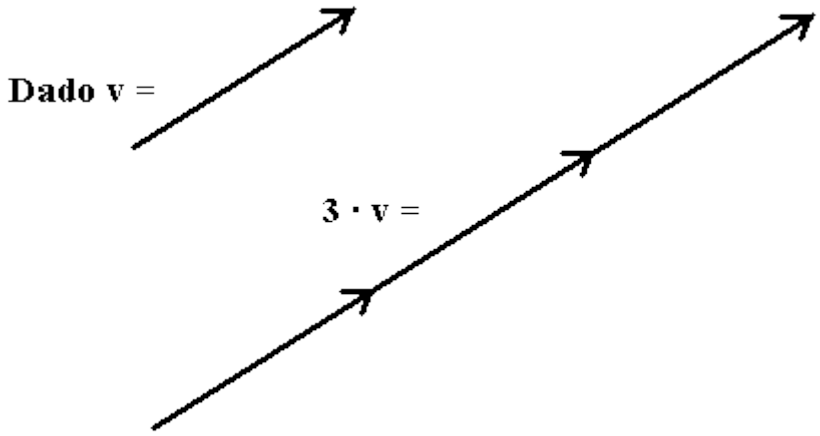
$$\vec{a} + \vec{b} = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$$

$$\vec{a} - \vec{b} = (x_1 - x_2, y_1 - y_2, z_1 - z_2)$$

Producto de un vector por un escalar

El resultado de multiplicar un escalar **k** por un vector **v**, expresado analíticamente por **kv**.

El módulo es **k** veces la longitud que representa el módulo de **v**



Producto punto

El producto escalar o producto punto es una operación definida sobre un espacio vectorial cuyo resultado es una magnitud escalar.

Definido por:

$$\vec{a} \cdot \vec{b} = (x_1 \cdot x_2, y_1 \cdot y_2, z_1 \cdot z_2)$$

$$a \cdot b = |a| \cdot |b| \cdot \cos\alpha$$

Para obtener el ángulo entre vectores:

$$\cos\left(\vec{a}, \vec{b}\right) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \cdot |\vec{b}|} = \frac{a_x b_x + a_y b_y + a_z b_z}{\sqrt{a_x^2 + a_y^2 + a_z^2} + \sqrt{b_x^2 + b_y^2 + b_z^2}} \Rightarrow \alpha = arccos(a, b)$$

$$\alpha = arccos(a, b)$$

Su utilización en los videojuegos es muy variada como por ejemplo para representar de mejor manera la posición de los componentes del videojuego, al no tener variables x, y, z sino una estructura que nos indica la posición, para representar de mejor manera la dirección en la que se están moviendo los objetos del juego.

La mayoría de motores de videojuegos utilizan vectores para representar en una figura:

- La translación.
- La escala.
- La rotación.

Además representan los vértices de un modelo 3D como un vector (x, y, z).

A partir de la definición y operaciones vectoriales, que fueron presentadas, una de las principales aplicaciones de vectores es realizada en los gráficos vectoriales que es lo siguiente a tratar en esta sección.

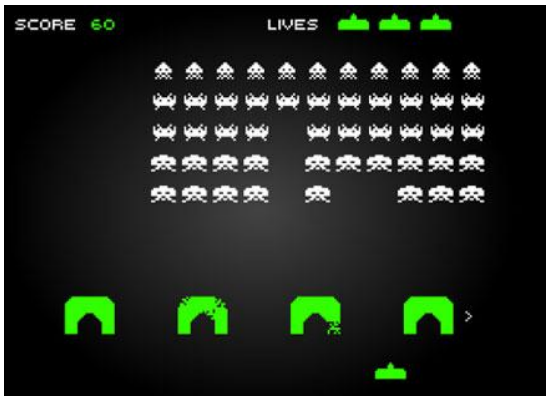
Gráficos vectoriales.

Una imagen vectorial es una imagen digital formada por objetos geométricos independientes (segmentos, polígonos, arcos, etc.), cada uno de ellos definido por distintos atributos matemáticos de forma, de posición, de color, etc. Por ejemplo un círculo de color rojo quedaría definido por la posición de su centro, su radio, el grosor de línea y su color.

Y es por estos atributos matemáticos de forma y posición que se denominan vectoriales, entre las principales aplicaciones: formatos en la web como SVG y SWF, diseño de tipografía.

Los primeros videojuegos se los realizaba utilizando gráficos vectoriales.

Figura 20. Space Invader



Fuente: El autor

Space Invader uno de los primeros videojuegos en salir al mercado, nótese que utiliza solo gráficos vectoriales que representan el jugador, los enemigos, el contador de vidas restantes y el acumulador de puntaje, e incluso las balas disparadas por el jugador.

- **Matrices**

Comenzó a utilizarse en los años 1850 como una forma abreviada de escribir un sistema de m ecuaciones lineales con n incógnitas.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}$$

Hoy en día además son utilizadas para la organización de los datos.

Columnas

Filas $\left[\left(\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}\right)\right]$

Se llama matriz de orden “m x n” a un conjunto rectangular de elementos a_{ij} dispuestos en m filas y en n columnas.

Suma de matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} \end{bmatrix}$$

$$A = \begin{pmatrix} 2 & 2 & 7 \\ 5 & 1 & 3 \end{pmatrix}; B = \begin{pmatrix} 2 & 1 & 1 \\ 3 & 6 & 2 \end{pmatrix}$$

$$A + B = \begin{pmatrix} 4 & 3 & 8 \\ 8 & 7 & 5 \end{pmatrix}$$

Producto de un número por un escalar.

$$\lambda \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \lambda a_{11} & \lambda a_{12} & \lambda a_{13} \\ \lambda a_{21} & \lambda a_{22} & \lambda a_{23} \\ \lambda a_{31} & \lambda a_{32} & \lambda a_{33} \end{bmatrix}$$

$$A = \begin{pmatrix} 2 & 1 & -2 \\ 5 & 3 & 4 \end{pmatrix}$$

$$(-2) \cdot A = \begin{pmatrix} -4 & -2 & 4 \\ -10 & -6 & -8 \end{pmatrix}$$

Producto entre matrices

Dadas dos matrices A = (a_{ij}) m x n y B = (b_{ij}) p x q donde n = p, es decir, el número de columnas de la primera matriz A es igual al número de filas de la matriz B, se define $A \cdot B$ de la siguiente forma:

El elemento que ocupa el lugar (i,j) en la matriz producto se obtiene sumando los productos de cada elemento de la fila i de la matriz A por el correspondiente de la columna j de la matriz B.

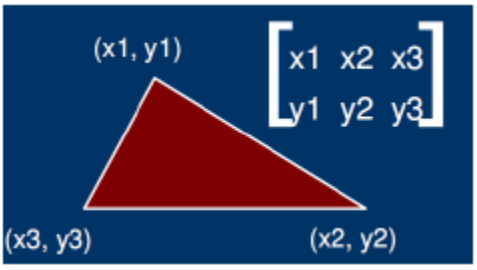
Matrices en gráfica computacional.

La aplicación principal y que corresponde al desarrollo de videojuegos son la utilización de gráficos tanto 2D como 3D.

Las tarjetas de videos realizan todo el procesamiento de gráficos utilizando matrices, se tiene una matriz de posición, de traslación, de escalamiento, de rotación y de la vista, estos matrices también se los puede descomponer en vectores dependiendo de su uso.

Para aprovechar esto, algunos motores de videojuegos se están jugando por trabajar con matrices como XNA.

Un modelo 3D, contiene polígonos, generalmente triángulos, representados de forma matricial, donde además cada uno de esos polígonos contiene vértices que también se representan de forma matricial.

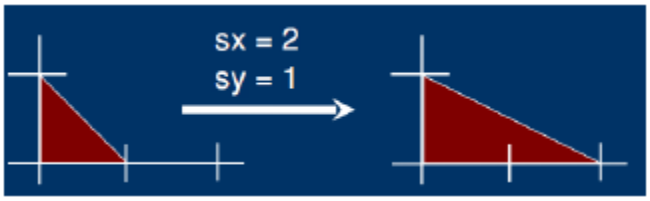


Escalamiento

Factores (sx, sy) permiten incrementar o decrementar el valor de las coordenadas (x, y) del objeto.

$$x' = sx * x$$

$$y' = sy * y$$



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Ejemplo:

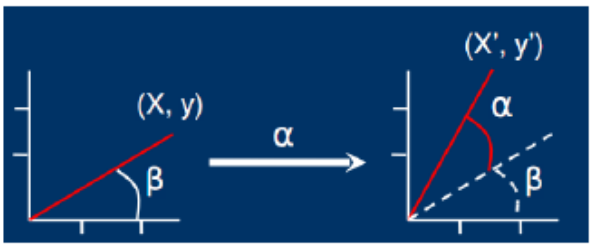
$$P = \begin{bmatrix} x0 & x1 & \cdots xn \\ y0 & y1 & \cdots yn \end{bmatrix} s1 = \begin{bmatrix} 2.2 & 0 \\ 0 & 2.2 \end{bmatrix} s2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 2.2 & 0 \\ 0 & 2.2 \end{bmatrix} \begin{bmatrix} 0.5 & 0 \\ 0 & 1 \end{bmatrix} P = s1 * s2 * P$$

Rotación

$$x' = x * \cos(\alpha) - y * \sin(\alpha)$$

$$y' = x * \sin(\alpha) + y * \cos(\alpha)$$

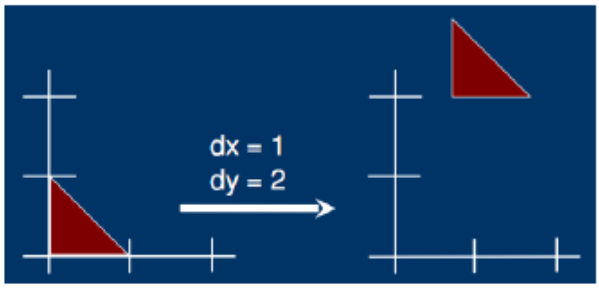


$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

Traslación

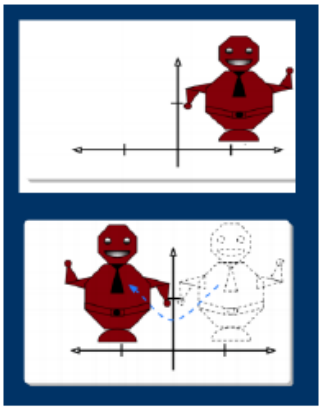
$$x' = dx + x$$

$$y' = dy + y$$



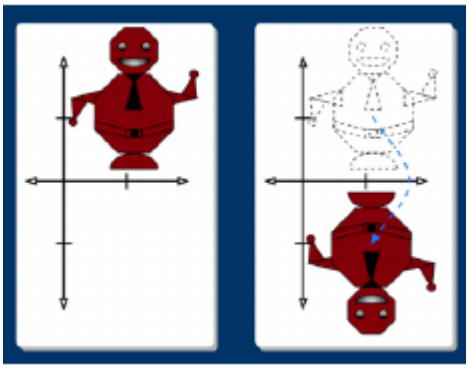
Reflexión en el eje Y

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



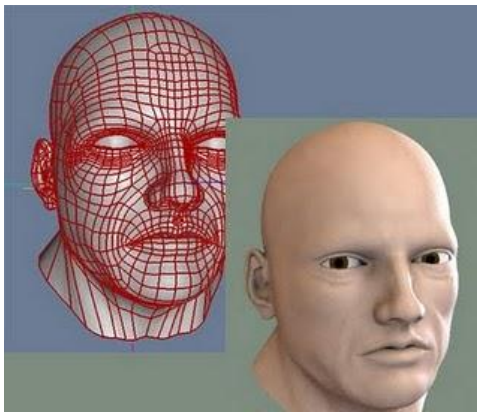
Reflexión en el eje X

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Algunas de las aplicaciones de las matrices como ya se mencionó anteriormente en la organización de los datos, representación matricial de una imagen, diseño de mapas basados en tiles para videojuegos 2D, diseño de tableros para videojuegos de tableros.

Figura 21. Bitmap: Gráficos matriciales.



Fuente: Wikipedia

Todos los métodos tanto para vectores como matrices son provistos por el API de XNA en el desarrollo de videojuegos, y que fueron descritos en términos matemáticos para una mejor comprensión, ya que antes de aplicar matemática en el desarrollo de un sistema independientemente del objetivo del mismo, es necesario conocer cómo funciona dichas operaciones matemáticas.

Dentro de XNA se manejan Matrices de dimensiones de 4 x 4 cada fila representa un componente del objeto quien es propietario de la matriz, esto se ilustrara mejor con un ejemplo:

$$\begin{pmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{pmatrix}$$

Se supone que un objeto es situado dentro del mundo de videojuego sin aplicarle rotación, sin escala, es decir con el tamaño original del objeto, pero en la posición 0, 0, 0 del mundo 3D, para esto se utilizará una matriz identidad, esto significa que en la diagonal principal los valores serán de 1.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Los componentes de esta matriz y como XNA lo interpreta es como sigue:

$(1 \quad 0 \quad 0 \quad 0)$ Representa el lado derecho del objeto que puede ser obtenido invocando al atributo Right de dicha matriz el cual devolverá un vector de tres dimensiones que corresponden a las coordenadas dentro del sistema de coordenadas. El mismo componente pero con valores negativos representa el lado izquierdo del objeto.

$(0 \quad 1 \quad 0 \quad 0)$ Representa la parte de arriba del objeto que puede ser obtenido invocando al atributo Up de dicha matriz el cual devolverá un vector de tres dimensiones que corresponden a las coordenadas dentro del sistema de coordenadas. El mismo componente pero con valores negativos representa la parte de abajo del objeto.

$(0 \quad 0 \quad 1 \quad 0)$ Representa la parte trasera del objeto que puede ser obtenido invocando al atributo Backward de dicha matriz el cual devolverá un vector de tres dimensiones que corresponden a las coordenadas dentro del sistema de coordenadas. El mismo componente pero con valores negativos representa la parte de frente del objeto.

$(0 \quad 0 \quad 0 \quad 1)$ Representa la traslación del objeto que puede ser obtenido invocando al atributo Translation de dicha matriz el cual devolverá un vector de tres dimensiones que corresponden a las coordenadas dentro del sistema de coordenadas.

En el ejemplo anterior nótese que la posición del objeto con respecto al mundo 3D es el vector (0, 0, 0) representado por los componentes de la matriz M_{41} , M_{42} , M_{43} correspondientes a los ejes X, Y y Z del plano cartesiano.

En realidad todos los atributos antes explicados solo utilizan las primera tres celdas de cada vector por ello el resultado de invocar los atributos correspondientes devuelven en vector de tres dimensiones, en conclusión la última columna no se utiliza para devolver estos atributos pero toda la estructura matriz utilizada por XNA representa la translación, rotación y escala de los objetos pertenecientes al mundo 3D.

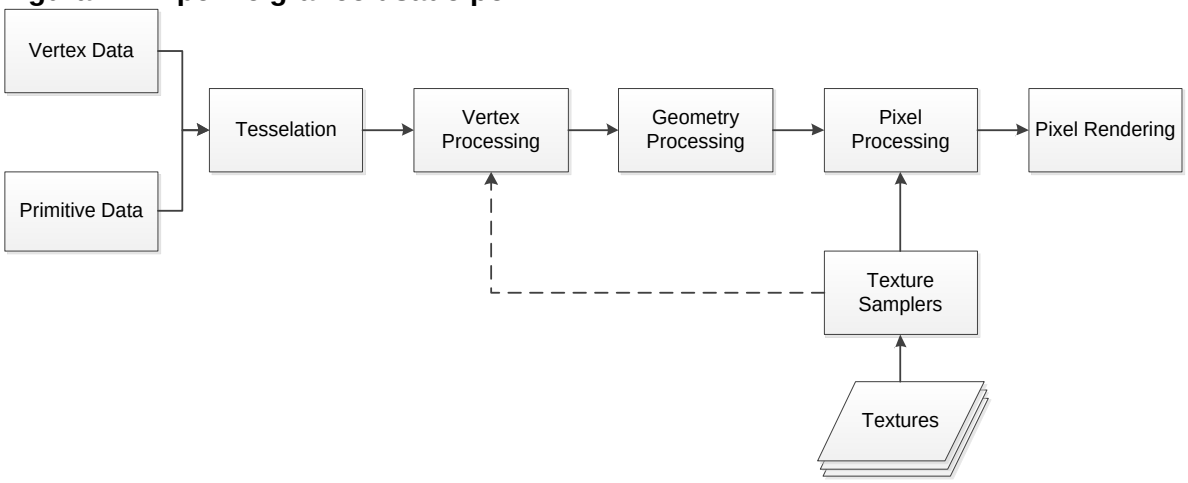
3.2.5 XNA Graphics Pipeline

Es importante conocer el pipeline gráfico usado por el este framework además de cómo se puede sacar el máximo partido a las tarjetas gráficas que un computador incorpora dentro de sus componentes de hardware a través de HSLS.

XNA no presenta mayores restricciones en cuanto al hardware de las maquinas cliente o usuarios de la aplicación, no obstante según la complejidad de gráficos utilizada por el videojuego, sería necesaria la presencia de un dispositivo gráfico, pero cabe resaltar que esta restricción se presenta por parte de la aplicación más no del framework. Comprometer la calidad del videojuego por limitaciones de hardware, será una decisión del desarrollador, para este proyecto se pensó en la compatibilidad con la mayoría de

máquinas de los usuario, por esta razón los videojuegos desarrollados no constaran de gráficos avanzados.

Figura 22. Pipeline gráfico usado por XNA



Fuente: http://geeks.ms/blogs/jcanton/archive/2010/04/01/xna-graphics-pipeline.aspx

Cada uno de estos módulos cumple un papel especial hasta conseguir pintar un objeto en pantalla, ya sea un modelo 3D o una imagen 2D (la cual es pintada sobre un quad "anclado" a la pantalla).

Estos módulos se los explica a continuación:

Vertex Data: Contiene un buffer de vértices sin transformar indexado o no indexado, es posible indicar mediante VertexDeclaration que información (position, color, texture coordinates, normals,…) viene definida por cada vértice en este buffer.

Primitive Data: Contiene las primitivas de geometría como points, lines, triangles y polygons leídos del index buffer que referencian a los vértices contenidos en el Vertex Data. En el index buffer aparecen cada una de estas primitivas y está asociada a una lista ordenada de vértices que la componen (hay que recordar que el orden es importante).

Textures: Se almacenan el conjunto de texturas que son usadas por el modelo que se va a pintar, como máximo se pueden usar 8 (texture0 – texture7).

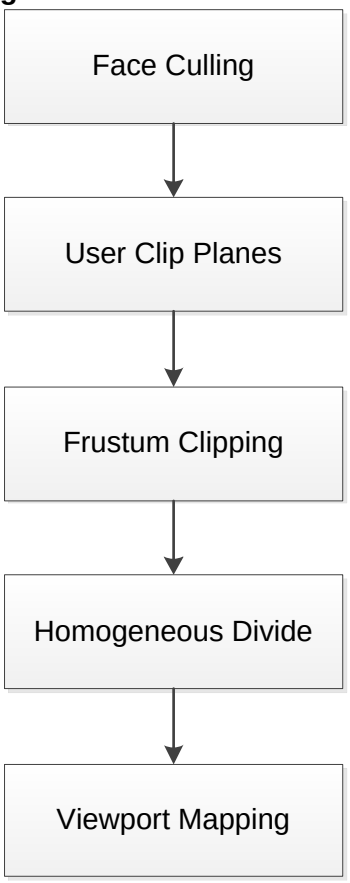
Texture Samplers: Todas las texturas que se vayan a usar en Vertex Processing o Pixel Processing tienen que ser usadas a través de Samplers en los cuales se indica el modo de direccionamiento de la textura (TextureAddressModel) y como será filtrada (TextureFilter).

Tesselation: (No disponible bajo XNA) En esta unidad se trabaja con N-Patches o Displacement maps para generar nuevos vértices y dar mayor nivel de detalle a los modelos, evitando enviar previamente todos esos vértices a la tarjeta gráfica (lo cual produciría incurrir en un cuello de botella).

Vertex Processing: Aquí es donde se ejecuta el código HLSL del Vertex Shader de nuestro Effect, para cada uno de los vértices almacenados en el vertex buffer. La principal tarea que se debe realizar en el Vertex Shader es pasar todos estos vértices de Model Space a Projection Space que viene dada por la configuración de la cámara.

Greometry Processing: En este módulo tienen lugar varias tareas:

Figura 23. Procesamiento de geometría



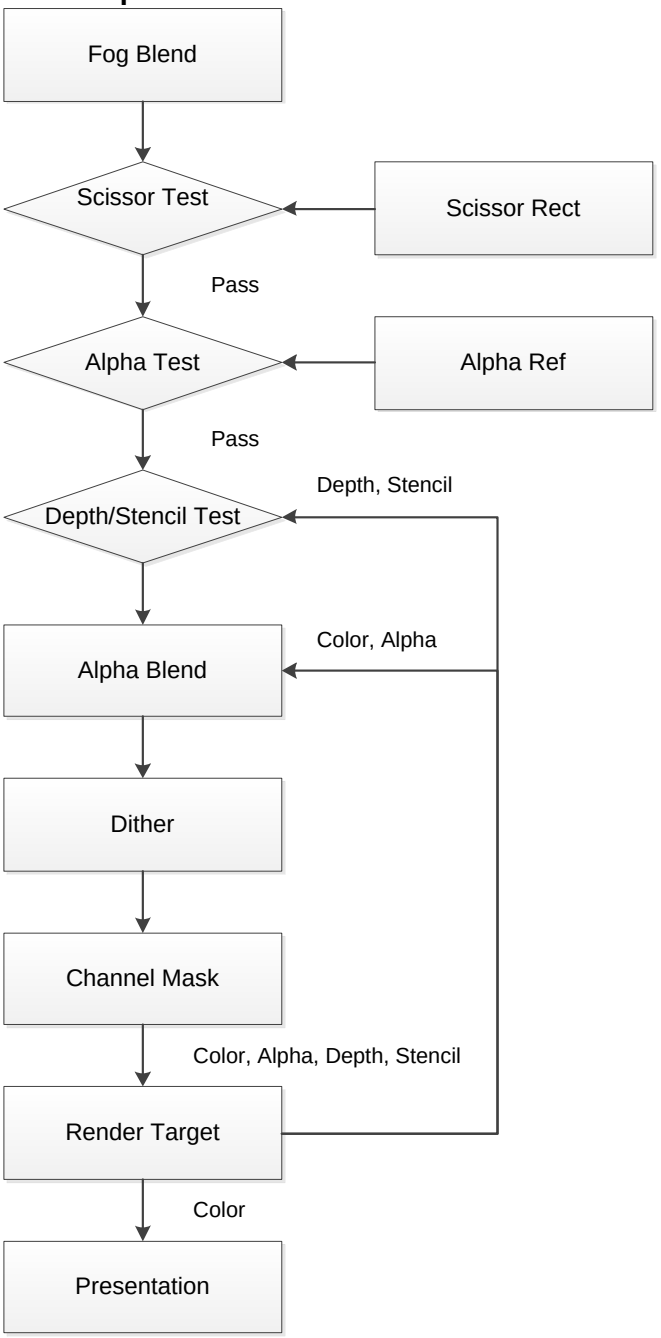
Fuente: http://geeks.ms/blogs/jcanton/archive/2010/04/01/xna-graphics-pipeline.aspx

- **Face Culling:** Los triángulos que no miran hacia la cámara (el ángulo formado por el vector normal del triángulo y el vector lookAt de la cámara es mayor de 90º) son borrados de la escena.
- **User Clip Planes:** Si el usuario a definido plano de corte, aquí es donde se realiza el test y todos los triángulos que están por detrás de estos planos son borrados de la escena.
- **Frustum Clipping:** Clipping es el proceso por el cual los triángulos que no aparecen dentro de la visión de la cámara son borrados de la escena.
- **Homogeneous Divide:** La información x, y, z de cada vértice es transformada a non-homogeneous space antes de ser enviada al rasterizer.
- **Viewport Mapping:** Se lleva cabo el proceso de asignación de pixeles de pantalla a cada triángulo.

Pixel Processing: Es donde se ejecuta el código HLSL del Pixel Shader de nuestro Effect por cada uno de los pixeles que forman el modelo, desde la función de Pixel Shader se suele hacer uso de los Texture Samplers para aplicar las texturas al modelo y alguna información de la geometría (WorldPosition, Normal,…) para realizar ecuaciones de iluminación (Lambert, Phong) a nivel de pixel (PerPixel Lighting), ó técnicas más avanzadas como Normal Mapping, Parallax Mapping, etc.

Pixel Rendering: En este módulo tienen lugar los siguientes test configurables en el Device:

Figura 24. Renderizado de pixeles



Fuente: http://geeks.ms/blogs/jcanton/archive/2010/04/01/xna-graphics-pipeline.aspx

- **Flog Blend:** La API proporciona la funcionalidad de aplicar niebla a la escena y es aquí donde se realiza el test.
- **Scissor Test:** También podemos definir un rectángulo de la pantalla (Scissor Rect) e indicar al Device que sólo se renderize esa porción del buffer. El test en el que se determina si los pixeles están dentro o fuera de dicho rectángulo se realiza aquí.
- **Alpha Test:** Se realiza una comprobación para evaluar mediante los valores del canal alpha si cumplen o no una condición marcada por el Alpha Ref.
- **Depth/Stencil Test:** Se llevan a cabo los test de Depth (profundidad), se actualiza el depth buffer con la profundidad del pixel si este será visible. También se aplica el Stencil Test (plantilla) para ver si el pixel afectará al color final del pixel en pantalla.
- **Alpha Blend:** En él se realizan las mezclas de colores que hacen posible el dibujado de objetos semitransparentes.
- **Dither:** Utiliza un algoritmo de interpolación de colores para combinar los pixeles adyacentes y así conseguir una gama de colores más consistente.
- **Channel mask:** Se puede escribir solo en los canales que deseemos (RenderState.ColorWriteChannels).
- **RenderTarget:** Se colocan los pixeles en el Render Target.
- **Presentation:** El RenderTarget es presentado por pantalla en el monitor.

Es importante conocer la generación de gráficos asistidos por computadora y como el computador junto con sus respectivos componentes ayudan al desarrollo de esta, para luego implementar algunas de las técnicas utilizadas para la optimización del rendimiento de un videojuego como Frustum Culling, o Z-Prepass.

Antes se mencionó la utilización de shaders para incrementar los efectos visuales de los gráficos utilizados en un videojuego, antes de entrar en definiciones, es necesario conocer dos conceptos necesarios para utilizar shaders como lo es pixel shader y vertex shader el cual es interpretado por el pipeline gráfico utilizado por XNA y que ya fue explicado.

Pixel Shader y Vertex Shader

Desde que comenzó la revolución 3D en el ámbito de los juegos de computadora, por mediados de la década de los 90', la tendencia de la tecnología aplicada a este rubro ha sido trasladar el trabajo de procesamiento de gráficos tridimensionales, desde la CPU hacia la tarjeta de video.

En primer lugar fue el filtro de las texturas, para lo cual se crearon chips especialmente dedicados para realizar esta tarea. Así nacieron las famosas placas aceleradoras 3D, que incorporaban dichos chips y una cantidad de memoria propia en la misma tarjeta. Luego, con la salida de GeForce 256 de NVIDIA, el procesador gráfico pasó a encargarse de lo que, hasta ese momento, realizaba la CPU. Se trataba de la función de Transformación e Iluminación (Transform & Lighting), utilizada para llevar a cabo los cálculos de geometría y de iluminación general de una escena en 3D. Hubo una versión mejorada de este motor, a la que se llamó de Segunda Generación. Ésta vino incluida a partir de la GeForce 2 y la gama Radeon de ATI, avanzando un poco más en cuanto a materia gráfica.

El gran cambio se dio a partir de la incorporación de los Píxel shaders y Vertex shaders. Esto permitió a los programadores una mayor libertad a la hora de diseñar gráficos en tres dimensiones, ya que puede tratarse a cada píxel y cada vértice por separado. De esta manera, los efectos especiales y de iluminación pueden crearse mucho más detalladamente, sucediendo lo mismo con la geometría de los objetos.

Figura 25. Shaders

Una tarjeta ASUS basada en el GeForce 256, primero en incorporar Transform & Lighting por hardware

Una parte del famoso árbol que mostraba las bondades de la tecnología de Transformación e Iluminación

La prueba de 3DMark 2003 que evalúa los shaders

Fuente: El autor

Se podría decir que son pequeños programas que se encargan del procesamiento de vértices (Vertex shaders) y de pixeles (Píxel shaders). La principal ventaja es que, como su naturaleza lo indica, pueden ser programados por el desarrollador, otorgando una flexibilidad que hasta antes de la aparición de los shaders era poco más que impensada. Recursos como las operaciones condicionales o los saltos se utilizan de forma similar que en los lenguajes más conocidos. Sin los shaders, muchos de los efectos eran realizados en conjunto con la unidad de procesamiento central, disminuyendo en gran medida el rendimiento y limitando el avance a nivel gráfico de los mismos.

Un vertex shader es una función que recibe como parámetro un vértice. Sólo trabaja con un vértice a la vez, y no puede eliminarlo, sólo transformarlo. Para ello, modifica propiedades del mismo para que repercutan en la geometría del objeto al que pertenece. Con esto se puede lograr ciertos efectos específicos, como los que tienen que ver con la

deformación en tiempo real de un elemento; por ejemplo, el movimiento de una ola. Donde toma una gran importancia es en el tratamiento de las superficies curvas, y su avance se vio reflejado en los videojuegos más avanzados de la actualidad. Particularmente, en el diseño de los personajes y sus expresiones corporales.

En cambio, un píxel shader no interviene en el proceso de la definición del “esqueleto” de la escena (Wireframe), sino que forma parte de la segunda etapa: la rasterización (Rendering). Allí es donde se aplican las texturas y se tratan los pixeles que forman parte de ellas. Básicamente, un píxel shader especifica el color de un píxel. Este tratamiento individual de los pixeles permite que se realicen cálculos principalmente relacionados con la iluminación del elemento del cual forman parte en la escena, y en tiempo real. Teniendo la posibilidad de iluminar cada píxel por separado es como se lograron crear los fabulosos efectos de este estilo que se pueden apreciar en Doom 3, Far Cry y Half Life 2, por mencionar sólo los más conocidos. La particularidad de los píxel shaders es que, a diferencia de los vertex shaders, requieren de un soporte de hardware compatible. En otras palabras, un juego programado para hacer uso de píxel shaders requiere si o si de una tarjeta de video con capacidad para manipularlos.

Figura 26. Diferencia entre aplicar y no aplicar pixel shaders.

Fuente: El autor

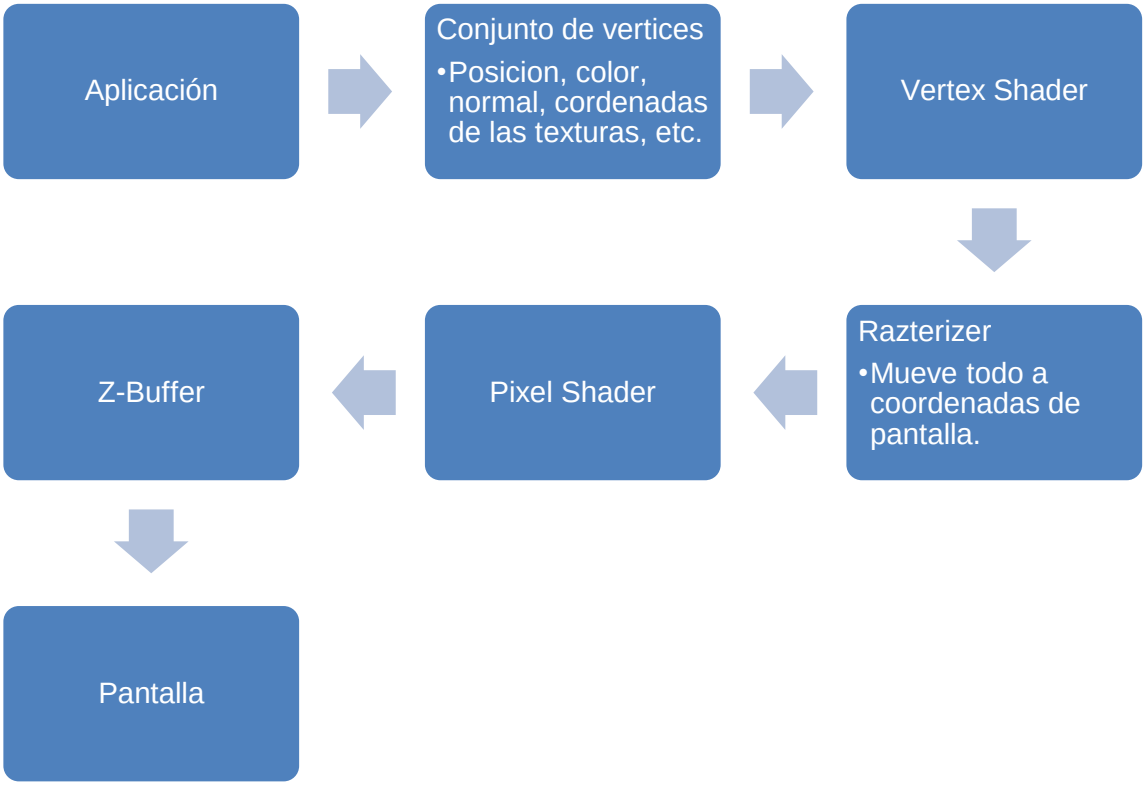
En la actualidad la mayoría de computadores por defecto ya tienen incorporado una tarjeta de video, por lo cual esto no será un impedimento.

3.2.6 Shaders

Hay una parte muy interesante de cuando programar gráficos se trata, esa parte es el utilizar el GPU (Graphics Processing Unit), es decir, programar la tarjeta de video para hacer cálculos y administrar la memoria.

Actualmente existen lenguajes de alto nivel (Cg de nVidia, HLSL de Microsoft y GLSL para OpenGL) que básicamente funcionan de la siguiente manera:

Figura 27. En XNA para programar shaders se debe utilizar HSLS.



Fuente: El autor

El High Level Shader Language or HLSL (Lenguaje de Sombreador de Alto Nivel) es un lenguaje de sombreado desarrollado por Microsoft para su uso con la API de Microsoft, Direct3D. Es análogo al lenguaje de sombreado GLSL usado con el estándar OpenGL. También es muy similar al lenguaje de sombreado CG de Nvidia, ya que fueron desarrollados juntos.

Los shaders son mini programas que se escriben para ser ejecutados en la GPU (Graphics Processing Unit, el procesador de las tarjetas gráficas de nuestros ordenadores).

A continuación se muestra la sintaxis y estructura de un shader implementado en XNA a través del IDE Visual Studio:

```
//Estas variables representan las matrices de posición de nuestro objeto
//la matriz de nuestra cámara, y nuestra proyección
float4x4 World;
float4x4 View;
float4x4 Projection;


//Esta estructura indica que es lo que va a tomar el shader
//En este caso solo tomara la posición del vértice, podemos agregar
//las coordenadas de textura, normales, colores, etc.
struct VertexShaderInput
{
    float4 Position : POSITION0;
};

//Esto es muy parecido a la estructura de entrada, pero aquí
//especificamos que queremos que devuelva nuestro método
struct VertexShaderOutput
{
    float4 Position : POSITION0;
};

//Esta función es nuestro vertex shader
VertexShaderOutput VertexShaderFunction(VertexShaderInput input)
{
    VertexShaderOutput output;

    //Sólo aplicaremos una sencilla transormación
    float4 worldPosition = mul(input.Position, World);
    float4 viewPosition = mul(worldPosition, View);
    output.Position = mul(viewPosition, Projection);

    return output;
}

//Esta función es nuestro pixel shader
float4 PixelShaderFunction(VertexShaderOutput input) : COLOR0
{
    //Aquí especificamos que queremos que para todo píxel
    //que se dibuje lo pinte de color rojo :)
    //podemos por ejemplo regresar el valor de una textura respecto
    //a la coordenada actual
    return float4( 1, 0, 0, 1);
}

technique Technique1
{
     //Para cada pasada especificamos que funciones vamos a llamar
     //y además debemos de decirle bajo que perfil se compilarán
    pass Pass1
    {
        VertexShader = compile vs_1_1 VertexShaderFunction();
        PixelShader = compile ps_1_1 PixelShaderFunction();
    }
}
```

Como se podrá observar la sintaxis se asemeja más a la escritura de programas en el lenguaje ensamblador, pero esta presenta mucha más legibilidad y fácil comprensión, pero también tiene algunas desventajas como una de las principales es la falta de optimización del código.

Las tarjetas de video están especializadas en cálculos en coma flotante y matricial a diferencia de las CPU que son procesadores de propósito general, por lo que el rendimiento de los juegos y aplicaciones con XNA y Shaders puede verse incrementado enormemente. Es más en XNA todo se pinta mediante Shaders, por eso para poder ejecutar XNA en cualquier ordenador se requiere una tarjeta que soporte Pixel and Vertex Shader 1.1 al menos, que en la actualidad no es una restricción de consideración importante, ya que cualquier chip gráfico de los que llevan varios años integrando en las placas base como las típicas de Intel 950 ya soportan esto.

Este es el principal motivo por el cual el rendimiento 3D de aplicaciones XNA está tan cercano al de aplicaciones DirectX escritas en C++. Como actualmente el cálculo intensivo se desplaza a la GPU, para la CPU quedan otro tipo de cálculos como puede ser la lógica del juego o la IA, y es esto lo que se escribe en C#.

Actualmente existen aplicaciones que facilitan la escritura shaders como FX Composer de Nvidia, Render Monkey creada por ATI.

Ya aclarado algunos de los conceptos utilizados por esta tecnología se prosigue con la descripción de la construcción del motor de videojuegos utilizado por el proyecto a desarrollarse.

3.2.7 Construcción del motor de videojuegos (Game Engine)

Dentro de las bondades de externas de XNA, es decir, que no es parte del API de este framework, es la amplia comunidad activa que utiliza y desarrolla aplicaciones con XNA, aunque cabe mencionar que la poca información que se pudo obtener a través internet estaba en inglés y la única fuente de información en español fue xnacomunity, alojada dentro del portal de codeplex, pero no obstante el mayor problema que se encontró hasta este momento, era los pocos proyectos que sirvieron de referencia para el desarrollo del proyecto, que ninguno de los cuales representaban un videojuego completo, si más bien era con fines de ser tutoriales básicos de cómo hacer aplicaciones simples con este framework.

Con el anterior antecedente el desarrollo de motor de videojuegos representa el proceso más lógico a modo de propuesta presentado por el autor del presente proyecto, ya que no fue posible contar con fuentes que describan como conseguir este fin, aunque a través de internet si existen páginas web que orientan al desarrollador en los pasos iniciales, pero eran muy básicos como aplicarlos en el desarrollo de proyecto con las características presentadas y que debían ser cumplidas.

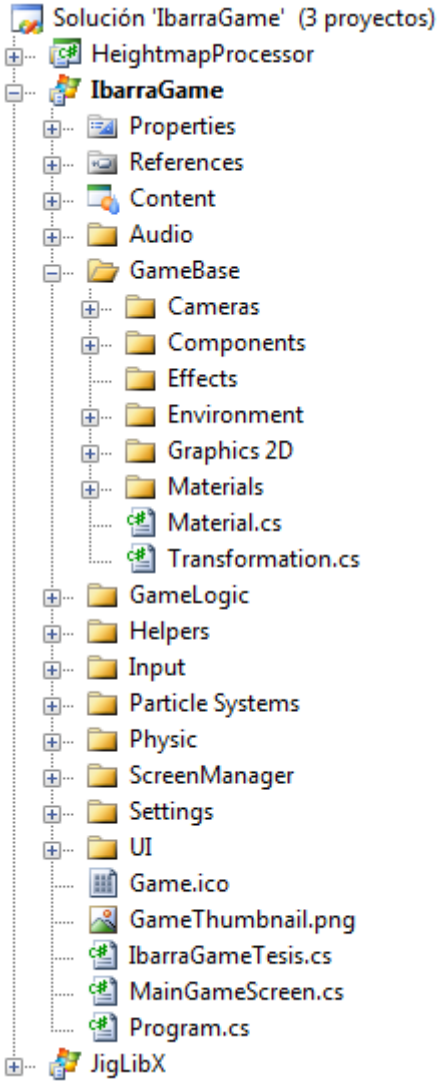
Antes de centrarse en la jugabilidad del videojuego, el motor de videojuegos debía encargarse de la comunicación de los diferentes módulos con los otros módulos pertenecientes al mismo proyecto, de aquí se parte primero al elemento más importante

dentro de un videojuego en 3D, la implementación del administrador de las cámaras que definirá el comportamiento desde la perspectiva del jugador y que elementos deben visualizarse a lo largo de la interacción aplicación – usuario.

Las cámaras utilizadas por los videojuegos desarrollados fueron una cámara en tercera persona con variantes en los parámetros que acepta para cambiar de perspectiva, una cámara en primera persona para la versión del juego multijugador y una cámara libre para viajar por todo el escenario y así ubicar los objetos de juego dentro del escenario.

La organización del proyecto se muestra en el siguiente gráfico:

Figura 28. Estructura de directorios del proyecto.



Fuente: El autor

A falta de información de cómo estructurar un proyecto de este tipo y alcance, la presente distribución y organización del proyecto fue la solución presentada por parte del autor del proyecto.

Como se notará en la carpeta GameBase está el administrador de cámaras, básicamente en este directorio se encuentra los componentes gráficos del motor desarrollado.

"La descripción de las clases y demás detalles del desarrollo de software se lo hará en el siguiente capítulo."

Como convicción a lo largo del desarrollo del proyecto se utilizó el idioma inglés, como se observa en el gráfico de la estructura de directorios, esto se adoptó por la fácil sintaxis que este idioma presenta, ya que si se lo hubiese hecho en español se tenía que seguir con las reglas gramaticales de este idioma, por ejemplo, los caracteres con tilde, caracteres como la ñ, etc.

Cabe mencionar que el motor de videojuegos construido es el resultado de la unión de varios componentes, si bien el gráfico anterior presenta la distribución lógica que se adoptó, pero esto no quiere decir que al inicio del proyecto esto ya estaba hecho, es decir, las necesidades del proyecto fueron implementados en el orden que el proyecto requería.

Con el anterior antecedente primero se pensó en el administrador de cámaras, y con un mundo que no tenía nada que ver con el proyecto se la iba probando, para luego ser mejorada, adaptada y por último, si cumplía con lo que se requería, se continuaba con el siguiente requerimiento.

El siguiente requerimiento fue el incorporar la entrada de datos a través del teclado, para lograr esto se decidieron mapear los botones del control de mando de XBOX y empatarlos con el teclado.

Figura 29. Mapeo de control de mando XBOX con el teclado

Fuente: El autor

Aunque la aplicación no pudo ser probada en alguna de las consolas soportadas por el framework, se estableció la compatibilidad a nivel de entrada de datos, para en un futuro probar la aplicación en una consola de XBOX sin mayor cambio en el código fuente.

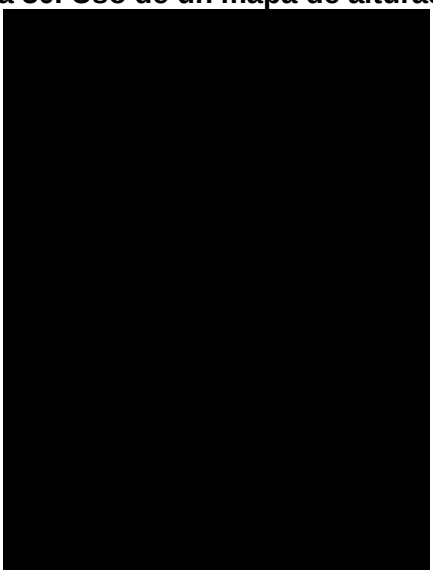
Una vez pensado en los componentes básicos del motor que hasta ahora han sido descritos, se continuó con el diseño del escenario, sin tomar en cuenta el motor de física

de utilizarse, se debió empezar con el terreno que formara parte del escenario. Para conseguir este fin se utilizó un procesador de contenido para hacer más extensible la Content Pipeline.

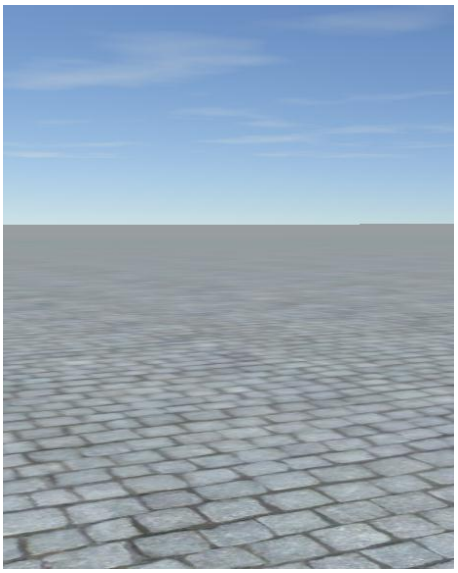
Para generar el terreno se parte de una imagen 2D en escala de grises, el cual servirá para generar un mapa de alturas dependiendo de la intensidad de color en la imagen, como solo tendrá variación de color de negro y blanco, las partes más claras tendrán mayor elevación que las partes obscuras de la imagen, la intensidad de cómo se procesar la imagen es configurable en el procesador de contenido.

EL procesador de mapa de alturas es un proyecto que se encuentra dentro de la solución del proyecto JigLib, se utilizó este debido que además era totalmente configurable y de fácil entendimiento, se integraba fácilmente al motor de física que se utilizó

Figura 30. Uso de un mapa de alturas



Textura que fue utilizada por el procesador de mapa de alturas.



Resultado de aplicar el mapa de alturas

Fuente: El autor

El resultado de aplicar el procesador de contenido al mapa de alturas es claramente visible, dentro de los parámetros que acepta están: el nivel de intensidad de las alturas, la escala del terreno, la escala de la textura a aplicar en el resultado que como se observa en la imagen, representa las calles con adoquín de la ciudad de Ibarra, y por último la ubicación de la textura a aplicar al terreno.

Luego de aplicar este procesador de contenido es el de una malla, que es resulta legible por la Content Pipeline de XNA a través de la clase Model de este framework.

Aunque el terreno que se utilizó en el proyecto no es exactamente idéntico al de la ciudad de Ibarra, pero fue la mejor solución que se pudo encontrar para cumplir con esta necesidad del videojuego, por suerte la porción de la ciudad que se utilizó en el videojuego no presenta mayores detalles de alturas, es decir, que prácticamente es una planicie.

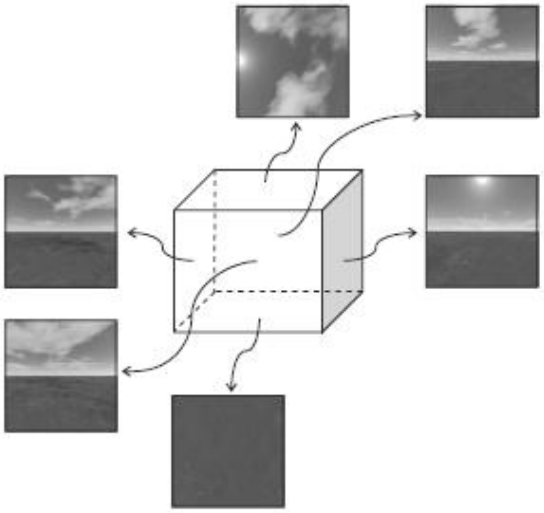
Dentro de las facilidades encontradas en la utilización de este procesador de mapas de alturas esta la integración con el motor de física que se utilizó en el proyecto JigLib, que ofrecía una clase con métodos para manipular la información de un mapa de alturas, como por ejemplo, la altura de un punto específico del mapa.

La principal intención de utilizar librerías ya existentes fue que en lo posible disminuir la complejidad del desarrollo del proyecto, y así acortar el tiempo del desarrollo del motor y centrarse en las características del videojuego.

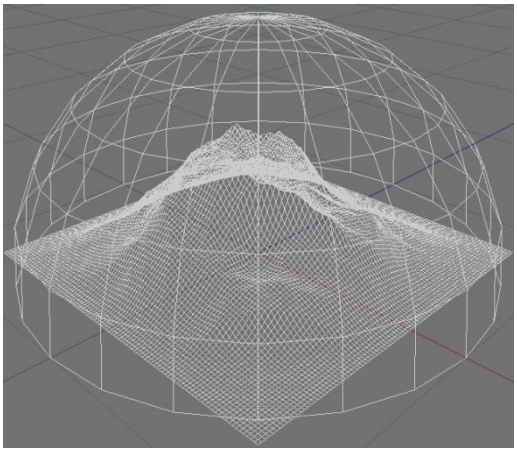
El escenario ya tenía un terreno pero, como es obvio el espacio restante era de color azul, fue el momento de incorporar un componente que remplace ese color por algo más representativo a la realidad, en este caso un cielo con nubes.

Para conseguir este objetivo y a través de una búsqueda en internet se encontró un componente que se comportaba la más cercano posible a lo que se requería, a continuación la respectiva explicación.

Figura 31. Uso de Skybox y Skydome



Skybox, con sus respectivas texturas



Skydome desde una perspectiva de renderizado wireframe

Fuente: El autor

Se necesitaba un fondo que oculte las partes que no tenían modelos ni texturas, es decir donde no se dibujaba nada, para esto existen dos opciones un Skybox y un Skydome, este último fue el que se lo implementó debido, a que se lo podía animar sin cambiar la visualización por parte del jugador y otras características más.

El componente que so lo utilizó se lo puede descargar desde la página oficial de codeplex con el nombre de AtmosphericScattering.

Este componente tenía incorporado algunas características que se requería dentro del proyecto como por ejemplo: nubes en movimiento con efectos que fácilmente se los puede modificar y personalizar.

Figura 32. Componente AtmosphericScattering en ejecución

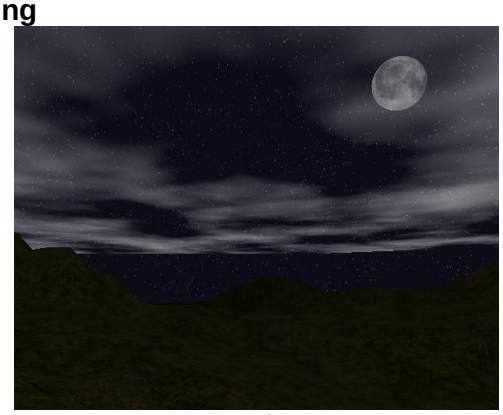
Fuente: El autor

Desde la perspectiva de renderizado wireframe se puede observar que el domo es una esfera recortada por la mitad, pero uno de los inconvenientes y que es claramente apreciada es la cantidad de vértices que se utiliza para formar la malla del domo, pero dicho inconveniente no representa un obstáculo para su utilización, puesto que la mayoría de computadores actuales si soportan el procesamiento de esta complejidad geométrica.

Este componente ofrecía una ventaja muy peculiar ante otros componentes del mismo tipo, y que vale mencionar que dentro de la comunidad de codeplex existen varios componentes Skydome, pero este era el más completo y era lo que el proyecto requería, esta ventaja era que permitía la transición de día y noche, con ciertas variantes en este comportamiento. Esta transición se la podía hacer de manera manual presionando ciertos botones para cambia de día a noche o viceversa, también ofrecía una transición automática a partir de parámetros como la velocidad de transmisión, ángulo en que el sol o luna debía transitar, y por último la posibilidad de capturar la hora del sistema y automáticamente simular si era de día o noche.

Figura 33. Componente AtmosphericScattering

Día



Noche

Fuente: El autor

Para dar un efecto más realista a este componente se implementó otro componente que funcione junto con la atmósfera del escenario.

Figura 34. Efecto fuego de lente en ejecución.

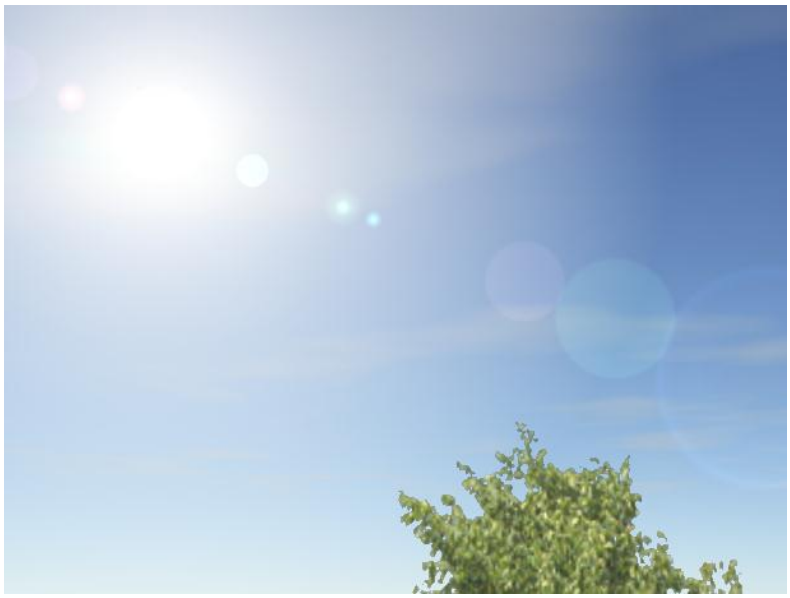
Fuente: El autor

Efecto fuego de Lente, y que básicamente trata de simular el efecto cuando una cámara visualiza al sol desde algún ángulo.

Este componente Fuego de Lente (LensFlare) se lo puede descargar de los ejemplos que ofrece Microsoft desde la página oficial de XNA.

Estos dos componentes fueron los utilizados para formar el ambiente y atmosfera del escenario, la principal ventaja de utilizar componentes es de facilitar las tareas al programador, y centrarse más en la lógica del videojuego. Y ya que estos componentes para ser utilizados deben tener el código fuente, fácilmente son personalizables e incorporar cambios en el comportamiento que inicialmente tiene, para luego irlos añadiendo a proyectos propios sin ninguna restricción.

Figura 35. Componentes que forman el parte del escenario y ambiente del videojuego



Fuente: El autor

Con el escenario ya ambientado y personalizado es necesaria la incorporación de personajes animados, sonido y video, que forma parte de la segunda parte del desarrollo del motor de videojuegos también se describe como fue la incorporación al videojuego.

3.2.8 Animación, sonido y video

Las animaciones 3D difieren de las animaciones en 2D, las animaciones 2D se las realiza a través de texturas 2D, en una secuencia lógica, para mostrar al usuario que el personaje tiene movimiento.

Figura 36. Secuencia de imágenes 2D de un personaje

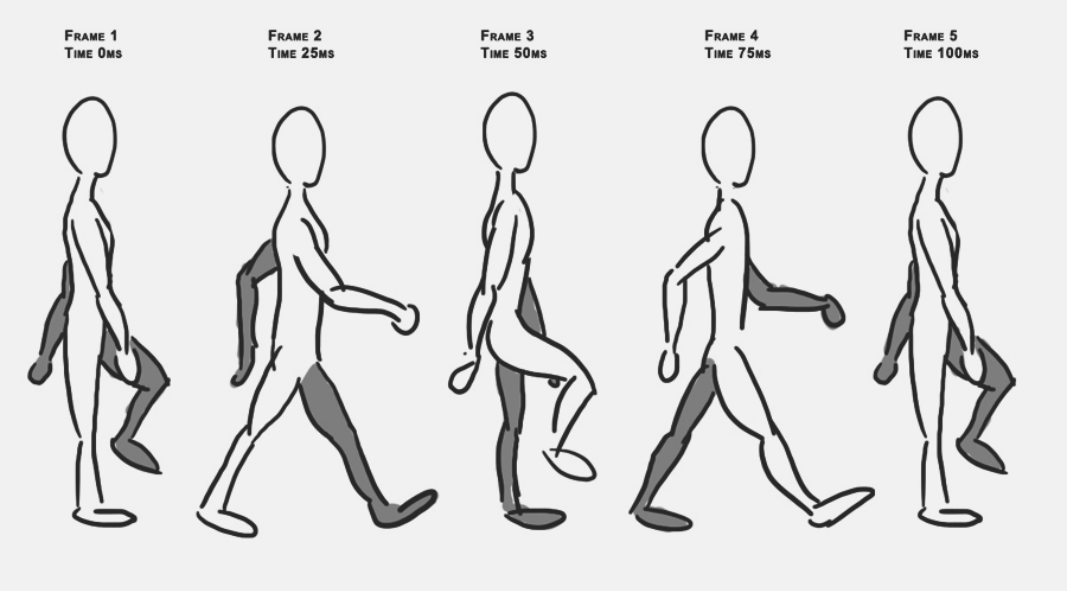
Fuente: El autor

Mientras las animaciones 3D son modelos animados a través de huesos que forman el esqueleto de dicho personaje, aunque también se pueden animar modelos 3D sin que

estos tengan un esqueleto, esto mediante el movimiento de las mallas que componen el modelo.

Existen principalmente dos formas de animar un modelo 3D: la animación por keyframes y la animación esqueletal.

Figura 37. Secuencia de caminar de una modelo 3D



Fuente: Libro Learning XNA 3.0

En esta animación de un personaje caminando, la malla del modelo es modificada en cada frame.

La animación por keyframes: consiste en almacenar una malla estática completa para representar cada posible posición intermedia del modelo, e ir cambiando sucesivamente la malla mostrada. Esto significa que, si queremos utilizar una animación de una persona caminando a 24 fps, en la cual cada paso dura 1 segundo, necesitaremos almacenar 24 mallas, cada una de ellas con su colección completa de vértices, normales, coordenadas de textura, etc.

También es posible diseñar solo unas pocas posiciones y hacer que el programa interpole los vértices para las posiciones intermedias. En cualquier caso, este proceso debería realizarse en la fase de carga (para conseguir una buena velocidad) y seguiríamos necesitando almacenar una malla distinta para cada uno de los frames de la animación.

La animación por keyframes puede ser muy rápida. Si tenemos muchos modelos idénticos simultáneamente en nuestro juego, es una buena opción. Sin embargo, en el caso habitual en el que existen muchos modelos diferentes, con diferentes animaciones, la animación por keyframes resulta ineficiente por ocupar una gran cantidad de memoria.

Puesto que XNA posee todas las clases necesarias para manejar modelos estáticos, este tipo de animación es muy fácil de implementar: por ejemplo, se puede almacenar cada frame de la animación como un objeto de la clase Model en un arreglo.

En el siguiente tipo de animación (animación esqueletal) también se utilizará la palabra “keyframe”, pero no debe confundir. Un keyframe en la “animación por keyframes” se refiere a cada una de las mallas que representa una posición del modelo. En la animación esqueletal, un keyframe es una matriz de transformación asociada a una serie de vértices en un instante concreto.

La animación esqueletal: es conceptualmente más compleja, pero más eficiente en el uso de la memoria.

Consiste en definir una estructura de “huesos” unidos jerárquicamente, formando un “esqueleto”. Cada hueso representa una parte móvil del modelo (un brazo, una mano, un dedo, etc). La organización jerárquica conlleva que, cuando se mueve el hueso del brazo, se vean arrastrados también los huesos de la mano y los dedos.

Los huesos del esqueleto están “conectados” por un punto con su hueso padre. En la animación esqueletal, cada keyframe de un hueso representa una transformación que indica el movimiento de ese hueso alrededor de su punto de conexión, en un instante determinado de la animación.

Cada vértice del modelo está asociado al menos a un hueso del esqueleto, de manera que al mover el hueso se mueve también el vértice. Los vértices pueden estar asociados a diferentes huesos, con diferentes “pesos”. De esta forma, el movimiento de un hueso puede arrastrar solo parcialmente un vértice. Así mismo, el movimiento final de un vértice puede ser la composición de los movimientos de diversos huesos.

La animación esqueletal resulta mucho más eficiente en memoria que la animación por keyframes, puesto que solo se almacena una matriz por cada hueso y keyframe, en lugar de almacenar todos los vértices con sus correspondientes coordenadas espaciales, coordenadas de textura, normales, etc. Por otro lado, puede resultar más lenta, dado que tenemos que multiplicar la matriz de transformación por las coordenadas de cada vértice (en lugar de redirigir un sencillo puntero, que es lo que haría al cambiar de posición en una animación por keyframes). Sin embargo, gracias a las operaciones en paralelo de las modernas tarjetas gráficas, esta multiplicación se hace tan rápido que, normalmente, los diseñadores se decantan por la animación esqueletal.

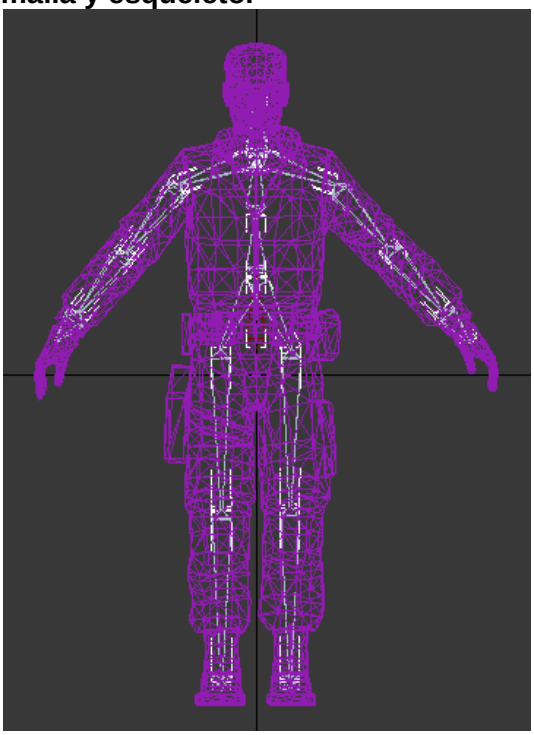
Estamos hablando de huesos como abstracciones que representan partes móviles de un modelo, pero en realidad un hueso solo es una forma de representar un grupo de vértices, al que se puede asociar una sucesión de transformaciones (giros) con un cierto peso.

Los modelos con animación de tipo esqueletal se suelen denominar “Skinned models”.

Los modelos que utilizan huesos se crean habitualmente en programas externos como Blender o Autodesk 3ds Max 2010.

XNA no tiene soporte nativo para animación esqueletal. El procesador por defecto de modelos de la Content Pipeline de XNA es capaz de extraer los vértices y huesos del modelo, pero descarta los datos de animación del modelo.

Figura 38. Modelo con malla y esqueleto.



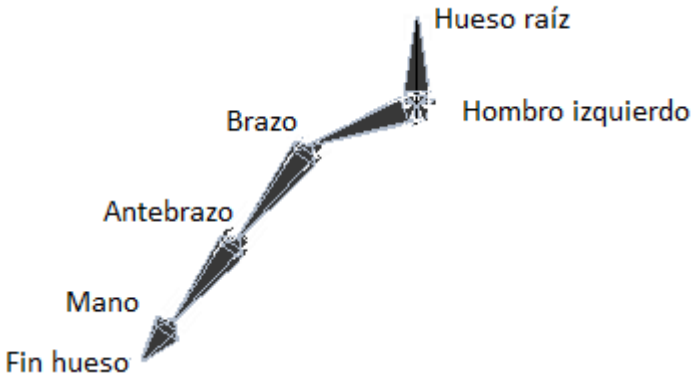
Fuente: El autor

Antes de empezar con la animación esqueletal en XNA, se debe explicar cómo los modelos con esqueletos esta construidos y como los huesos son representados y construidos.

Existen dos diferentes caminos para guardar el esqueleto de un modelo: usando huesos (bones) o usando junturas (joints). Por ejemplo, Autodesk 3ds Max 2010 representa un esqueleto usando huesos, mientras en Maya un esqueleto se representa usando junturas. Sin embargo, cuando el modelo es exportado a XNA en un formato compatible (.X o .FBX), esto no representa diferencias entre ellos y el esqueleto es representado por estos huesos.

Tanto, si se usa huesos para representar y guardar el esqueleto. Cada hueso guarda una posición y rotación inicial, definiendo donde y como está unido al hueso padre. Este también guarda su tamaño, que está definido como la distancia entre su posición y la posición del hueso hijo. Esta representación del hueso crea lo necesidad de tener un final para el hueso (de tamaño cero) que define el fin del esqueleto.

Figura 39. Huesos del brazo de un esqueleto



Fuente: El autor

La posición y orientación de cada hueso está relacionado jerárquicamente con el hueso padre. Por ejemplo: la orientación y posición de la mano está definido de acuerdo a la orientación y posición definidas por el antebrazo, el cual tiene su orientación y posición definida por el brazo, repitiendo el mismo proceso hasta alcanzar el hueso raíz. Con este concepto, se puede ver que modificando cualquier hueso en particular afectará todos los huesos descendentes. Si el hueso del hombro izquierdo es movido o rotado, todos sus descendientes también serán movido o rotados.

Para guardar el esqueleto, se necesita la configuración (orientación y posición) de cada uno de los huesos y la jerarquía de estos huesos dentro del esqueleto. La jerarquía es necesario cuando se renderiza el modelo, ya que luego se necesitara encontrara la posición 3D absoluta de cada vértice del modelo. Por ejemplo: la posición absoluta de un vértice del antebrazo es encontrada empezando desde la posición original del vértice, multiplicado por la configuración del hueso de antebrazo, el hueso del brazo, el hueso del hombro izquierdo, y el hueso raíz.

Afortunadamente, como cada hueso guarda esta configuración, XNA hace estos cálculos automáticamente.

La configuración de un hueso es guardada dentro de una matriz. La jerarquía del esqueleto es guardada en una lista de huesos, cada uno con su matriz y un vínculo a su hueso padre.

Para añadir soporte a la animación esqueletal en XNA se utilizó una librería de animación obtenida a través de internet de la comunidad de codeplex, que a continuación se la detallará, pero antes se debe hacer una aclaración importante.

En la comunidad de codeplex algunos de los proyectos son alojados junto con los códigos fuentes del proyecto, y cabe mencionar que la librería utilizada para el soporte de animación esqueletal si tenía el código fuente para su libre descarga, no obstante, en el proyecto a desarrollarse solo se utilizó los archivos .dll que son resultados de la compilación de los proyectos del procesador de contenidos que hace extensible la Content Pipeline de XNA.

Una biblioteca de vínculos dinámicos (DLL) es un archivo ejecutable que actúa como una biblioteca de funciones compartida. La vinculación dinámica proporciona a los procesos

una forma de llamar a una función que no forma parte del código ejecutable. El código ejecutable de la función está en un archivo DLL, que contiene una o más funciones que se compilan, vinculan y almacenan de forma independiente de los procesos que las utilizan. Los archivos DLL también facilitan el uso compartido de datos y recursos. Distintas aplicaciones pueden tener acceso simultáneamente al contenido de una única copia de un archivo DLL en la memoria.

XNAnimation

Versión actual: XNAnimation 0.7.0.0 BETA 3

Versión utilizada: XNAnimation 0.7.0.0 BETA 3

Fecha: Febrero 5 2010 a las 8:00 AM

Desarrollador: Bruno Evangelista programador de gráficos que actualmente está estudiando para una maestría Licenciado en Ciencias de la Computación en la Universidad Federal de Minas Gerais (UFMG), Brasil.

Ávido desarrollador XNA que han estado trabajando en algunas bibliotecas y juegos con XNA, uno de los autores Beginning XNA 3.0 Game Programming: From Novice to Professional.

Introducción

XNAnimation es una librería de animación esqueletal para XNA. Esta librería permite a desarrolladores la facilidad de manipular, reproducir, interpolar y combinar animaciones.

A partir de febrero de 2010, el desarrollo de XNAnimation ha continuado.

Características principales

- General
    - Fácil de usar.
        - Provee clases para manejar “skinned models”, esqueletos, clips de animación, keyframes y poses.
        - Provee controladores para manipular la reproducción de animaciones.
    - Alto rendimiento.
        - Interpolación de animaciones y mescla son hechas en la CPU.
        - Los “skinning models” son hechas en la CPU.
    - Bajo consumo de memoria.
        - Almacena los keyframes de translación, orientación y escala descompuestos.
        - Accione de esqueletos, animaciones y recursos de mallas.
    - Compatible con plataformas Windows y Xbox.
- Reproducción de animaciones.
    - Reproduce animaciones hacia adelante y viceversa.
    - Controles de velocidad de la animación y bucles.
    - Soporta modelos con hasta 80 huesos.
    - Soporta interpolación de keyframes lineal, cubica y esférica.

        - Almacena los keyframes de translación, orientación y escala descompuestos.
        - Interpola la translación, orientación y escala por separado para una mejor interpolación.
- Fusión de animaciones
    - Soporta el fundido entre clips de animación.
        - Permite suavizar las transiciones entre animaciones.
- Material del sistema
    - Soporta múltiples fuentes de puntos de luz (hasta ocho) con atenuación de la luz
    - Soporta materiales con propiedades emisivos, difusos y especulares.
    - Soporta texturas difusas, normales y especulares.
    - Reporta el perfil de vertex y pixel shader usados.
- XNAnimation procesador de modelos
    - Divide cualquier animación en una nueva serie de animaciones (basado en el timepo o keyframes)
    - Transforma toda la escena usando la rotación o escala.

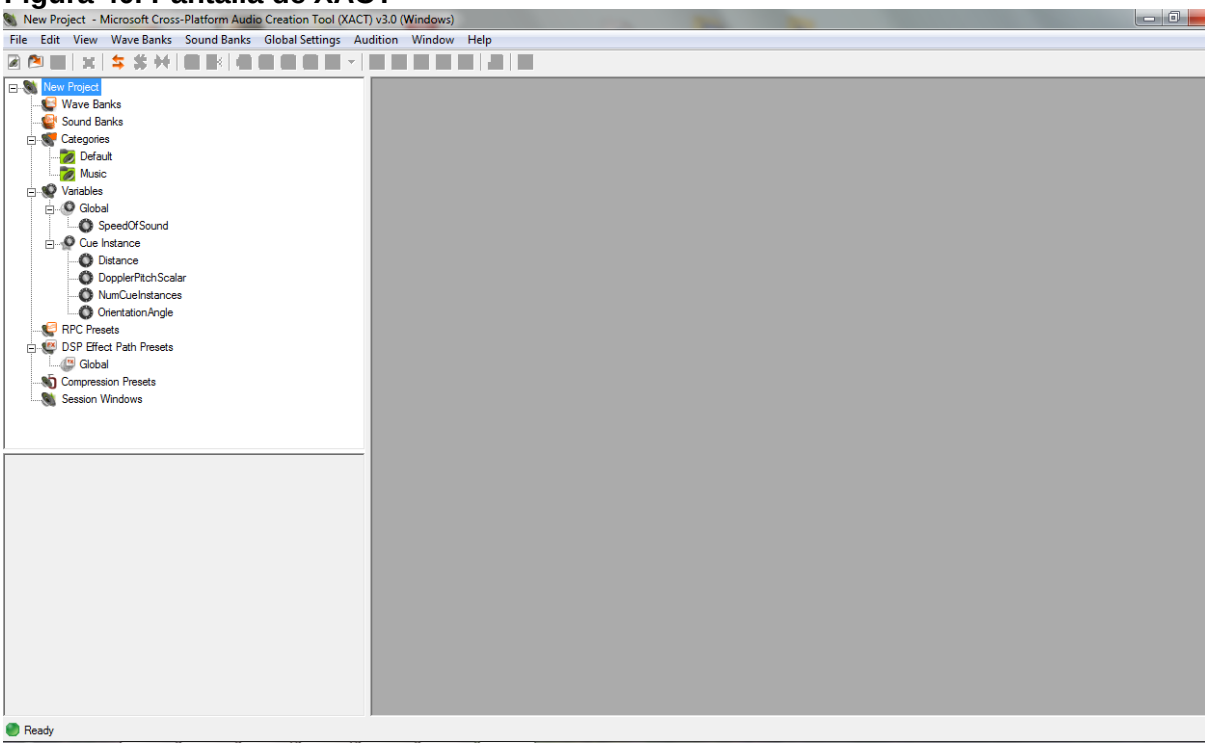
Fuente: http://xnanimation.codeplex.com/

La utilización de esta librería es sencilla, dentro de la página de descarga se pueden obtener ejemplos de cómo utilizar esta librería.

La implementación de sonido y como parte del motor de videojuegos se lo realiza a través de un administrador de un manejador de sonidos, el cual provee acceso a los sonidos dentro del banco de sonidos, para introducción sonido en la aplicación se debe conocer como XNA provee esta funcionalidad.

Con el framework XNA 3.1 hay un par de maneras diferentes de aplicar el sonido. En versiones anteriores de XNA, los desarrolladores utilizaban una herramienta llamada Microsoft Cross-Platform Audio Creation Tool (XACT) exclusivamente para el audio. Con el uso de XACT, los desarrolladores pueden crear las compilaciones de sonido que son procesados por la Content Pipeline y son ejecutados usando la API de sonido de XNA. Con XNA 3.1, XACT todavía se puede utilizar para aplicar audio para PC y Xbox 360. Sin embargo, ya que Zune no es compatible con el motor de XACT, XNA ha añadido una API para el desarrollo de Zune.

Figura 40. Pantalla de XACT



Fuente: El autor

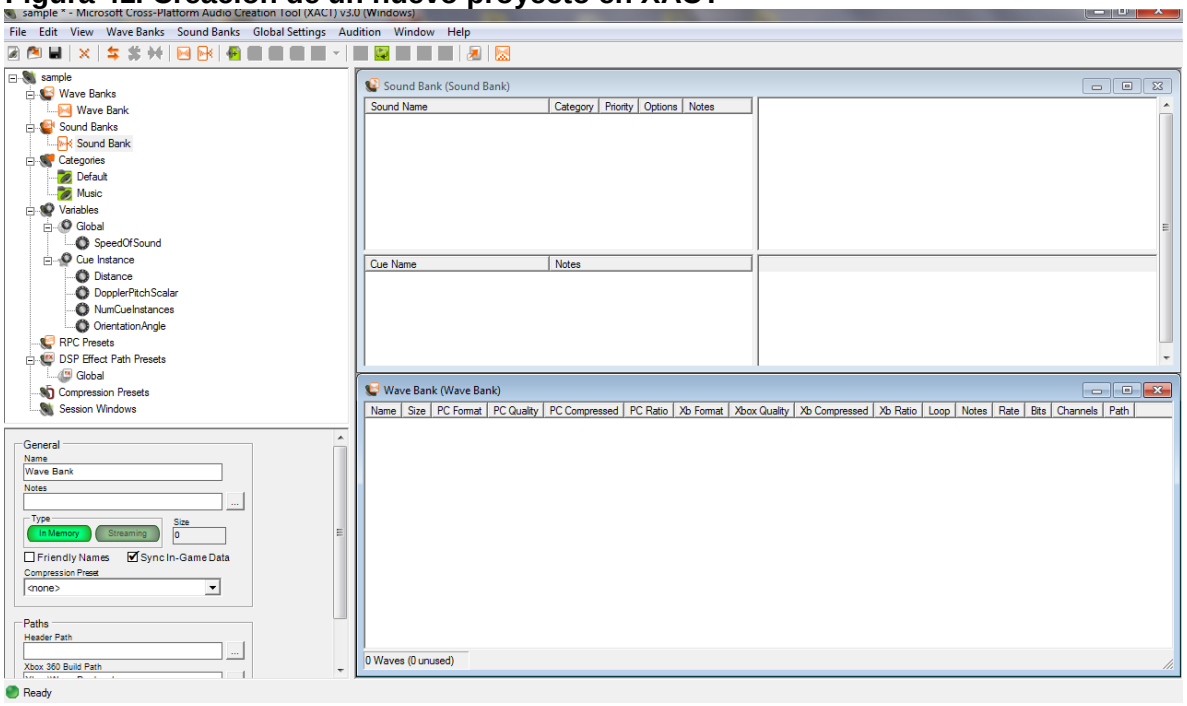
En muchos aspectos XACT proporciona un mini estudio de sonido. Se puede editar el volumen, tono, bucles, y otras propiedades de sonido sin escribir nada de código.

El método de XACT de implementación de audio ofrece más características que las disponibles usando la API simplificada de sonido, pero tiene una desventaja, es el método más complicado de aplicación de audio.

Para la implementación de sonido usando XACT, se necesita los archivos de audio en formato .wav los mismos que fueron recolectados a través de internet y de otros videojuegos para realizar este proyecto.

Los archivos de audio deben estar dentro del directorio de la Content Pipeline, pero no deben pertenecer como archivos multimedia que deben ser procesados, es decir, los archivos de audio deben ubicarse dentro de la Content Pipeline al que puede accederse desde el explorador de Windows, mas no pertenecer al proyecto y que a través de un procesador de contenidos integrarse al proyecto. Con el uso de XACT lo que debe añadirse a la Content Pipeline es el proyecto creado en XACT el cual contiene un banco de sonidos con sus efectos respectivos y los que pueden ser reproducidos mediante programación.

Figura 41. Creación de un nuevo proyecto en XACT

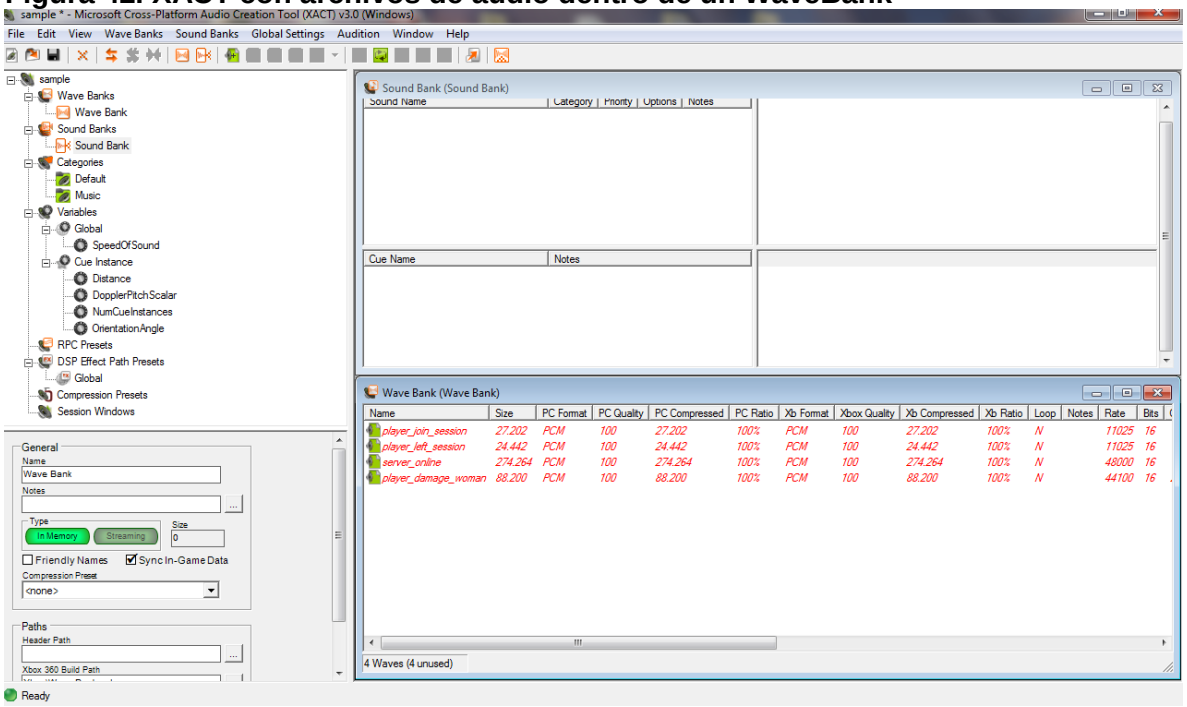


Fuente: El autor

Al crear un nuevo proyecto en XACT se debe crear Wave Banks y Sound Banks con nombres representativos, es aquí donde se alojaran los archivos de audio.

Los archivos de audio deben ser ingresados primero dentro de un Wave Bank, si el archivo es correcto este podrá será aceptado y reproducido en el proyecto.

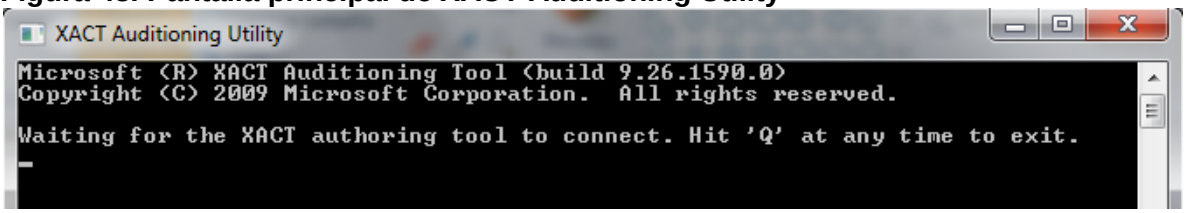
Figura 42. XACT con archivos de audio dentro de un WaveBank



Fuente: El autor

Los archivos de audio que fueron ingresados a esta herramienta pueden ser reproducidos desde esta, para ello es necesario abrir una utilidad de XACT y que fue instalada automáticamente cuando se instaló XNA, XACT Auditioning Utility, más que nada sirve para propósitos de desarrollo, ya que al aplicar efectos sobre sonidos es necesario escucharlos para verificar sus resultados.

Figura 43. Pantalla principal de XACT Auditioning Utility

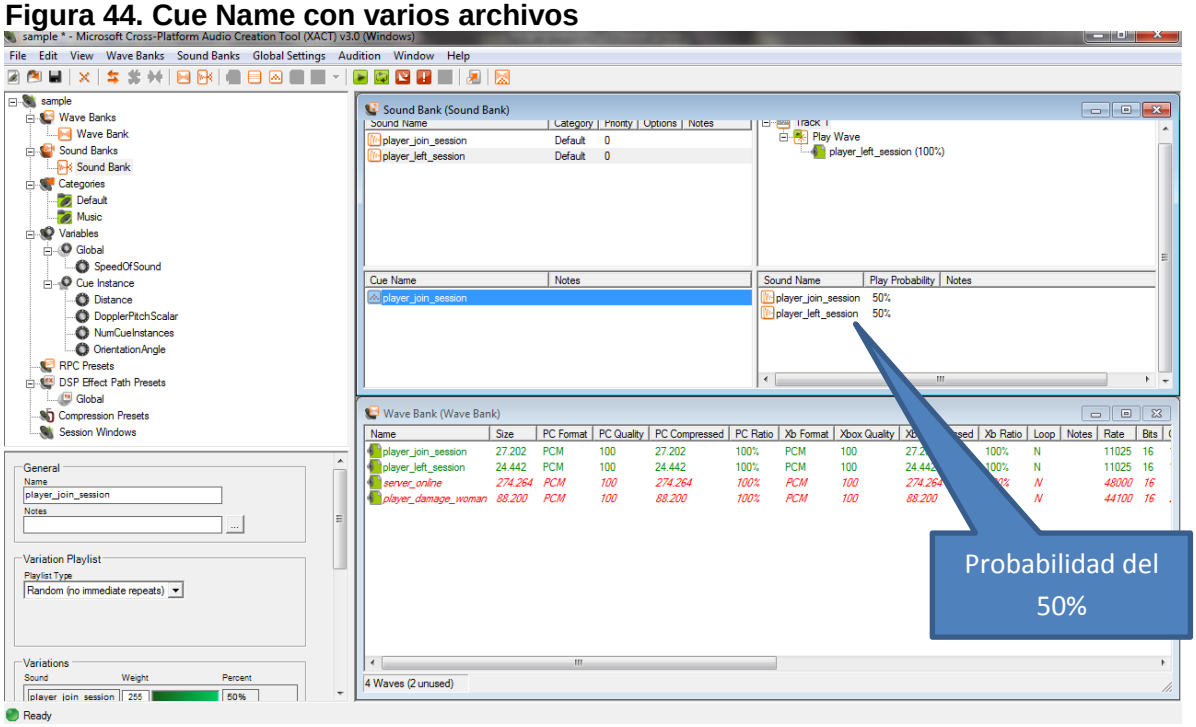


Fuente: El autor

Los archivos dentro de un Wave Bank pertenecerán a un Sound Bank, dentro de este se define el Sound Name y Cue Name, para hacer esto se debe arrastrar el archivo de audio desde el Wave Bank hasta el Sound Bank, y automáticamente se añade también a los Cue Name.

Cuando varios archivos de sonido pertenecen a un nombre de sonido, y que deberán ser reproducidos de acuerdo a una probabilidad, primero se debe ingresar los archivos desde el Wave Bank hasta el Sound Bank en Sound Name, se selecciona el Cue Name que contiene va a contener varios archivos de sonido y desde los Sound Name se selecciona el archivo y se arrastra al Cue Name respectivo.

Figura 44. Cue Name con varios archivos

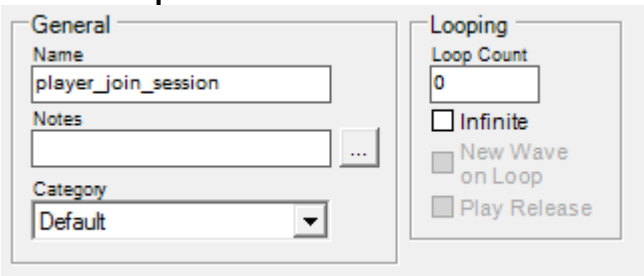


Fuente: El autor

Para que los archivos de audio sean reproducidos mediante código a través del manejador de sonidos perteneciente al motor de videojuegos desarrollado se debe hacer referencia al nombre del Cue Name donde está ubicado el archivo.

Los efectos de sonido son fáciles de implementar sin ingresar una sola línea de código, esta es una de las ventajas de la herramienta XACT.

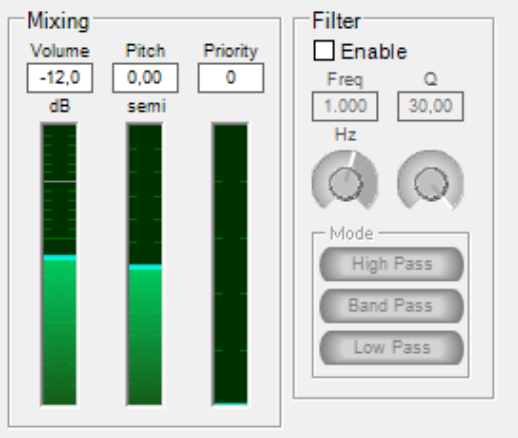
Figura 45. Efectos de sonido que ofrece XACT



Fuente: El autor

Para lograr el efecto de reproducción en bucle de un sonido, solo basta con activar el casillero Infinite de las opciones Looping.

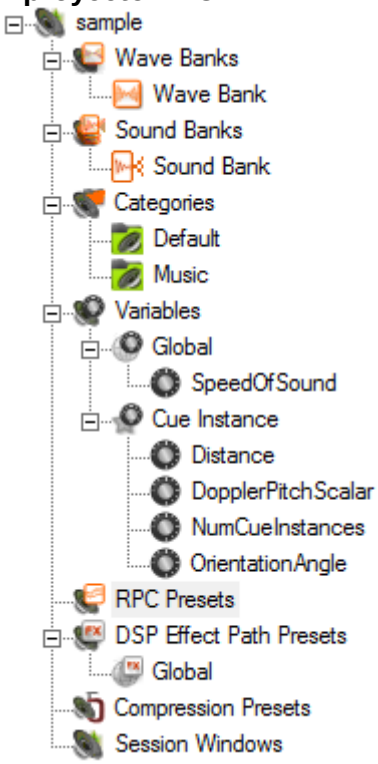
Figura 46. Efectos de sonido que ofrece XACT



Fuente: El autor

También ofrece control sobre las variaciones en el volumen del sonido.

Figura 47. Estructura de un proyecto XACT



Fuente: El autor

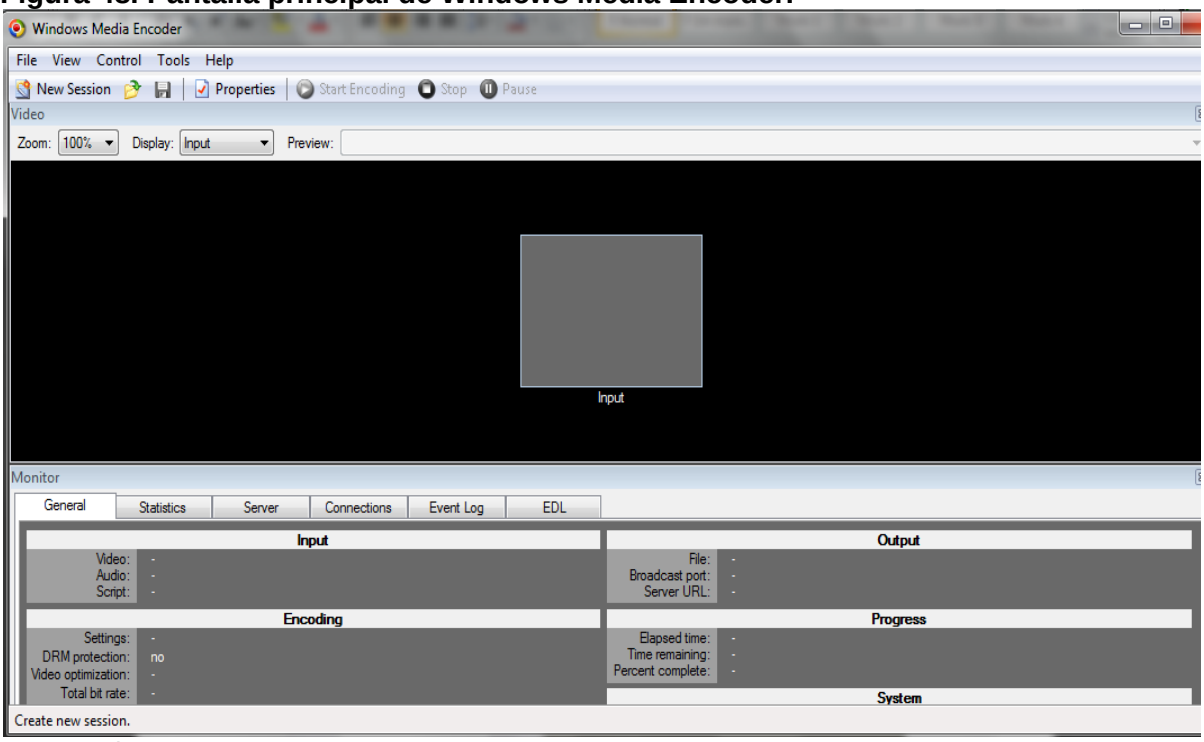
Como requisito antes de utilizar esta herramienta se debe conocer aspectos básicos de la edición de archivos de sonido, ya que como se podrá notar para la edición de variaciones del volumen del sonido de los archivos de audio, las unidades en las que se puede medir la intensidad del volumen esta en decibeles. A partir de este antecedente para la aplicación de efecto 3D sobre sonidos se debe tener un conocimiento más avanzado, por esta razón en el proyecto desarrollado no se implementó este efecto, pero se explicará cómo funciona:

El efecto de sonido 3D viene a ser el volumen del sonido reducido o aumentado según la distancia desde donde será escuchado, es decir, si la posición desde donde será escuchado el sonido, está lejos del emisor del sonido, el volumen del sonido será bajo, en el caso de que el emisor este cerca con respecto a la posición del receptor este será más alto. Algunas de las configuraciones para lograr este efecto se lo realiza dentro de la herramienta XACT pero también mediante la implementación en el código fuente del proyecto.

El soporte de video no es parte del motor de videojuegos, más bien es una funcionalidad de XNA en la versión 3.1.

El formato de video el cual es soportado por la Content Pipelines es .wmv, aunque a veces puede dar error al ser procesado dentro de la aplicación, para esto se debe aplicar algunas configuraciones con Windows Media Encoder, programa que puede ser descargado gratuitamente.

Figura 48. Pantalla principal de Windows Media Encoder.



Fuente: El autor

La incorporación de video en cualquier aplicación XNA es extremadamente sencilla, solo basta cargar el video desde la Content Pipeline en una clase que lo soporte en este caso Video y para manipular dicho contenido multimedia se lo realiza a través de la clase VideoPlayer.

3.2.9 Motor de Física

La implementación de física es uno de los aspectos más importantes en el desarrollo de un videojuego, ya que hace más realista la aplicación que se está desarrollando a través de la simulación de física y como afecta a los objetos que pertenecen al mundo del videojuego.

Para la implementación motor de física dentro del motor de videojuegos se utilizó JigLibX que se detalla a continuación:

JigLibX

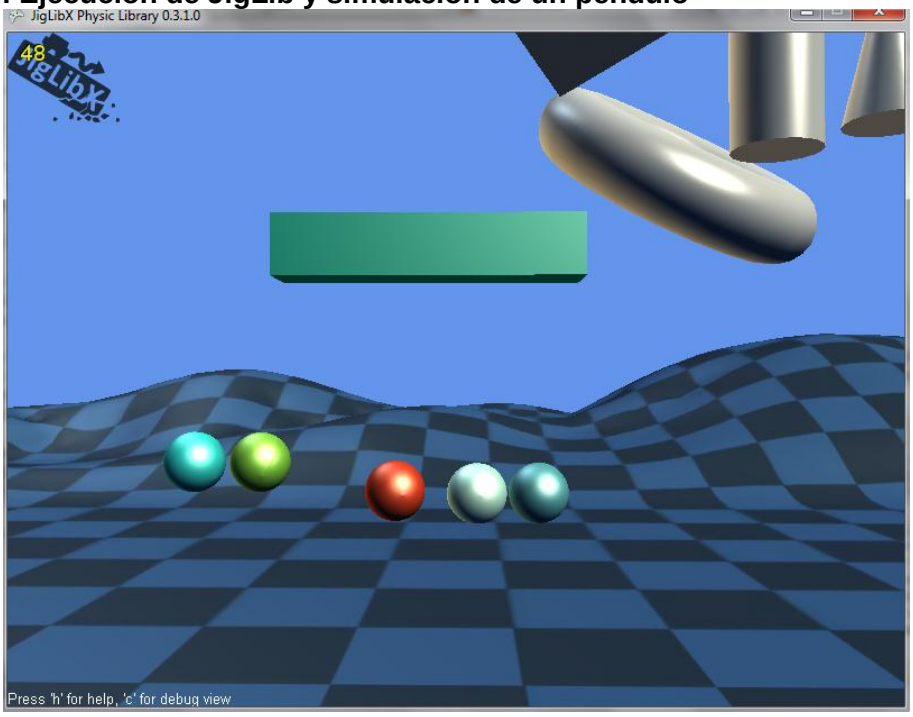
Versión actual: JigLibX 0.3.1 Beta

Versión utilizada: JigLibX 0.3.1 Beta

Fecha: Abril 8 2009 at 3:00 AM

Desarrollador: Thorben Linneweber

Figura 49. Ejecución de JigLib y simulación de un péndulo



Fuente: El autor

Introducción

JigLibX es un motor de física escrito en C # utilizando el framework de Microsoft XNA. Está basado en el motor de física JigLib y actualmente está siendo portado y ampliado. Tiene un sistema de colisiones y un motor de física de cuerpos rígidos que hace de JigLibX uno de los más preferidos motores de física de código abierto diseñado para su uso con XNA.

Conceptos

La dinámica sobre cuerpos rígidos es una especie de simulación utilizado en los juegos para dar el comportamiento de física sobre objetos. Para cada objeto que se desea simular, uno o varios cuerpos que están definidos, cada uno de ellos tiene una o más primitivas de colisión. Para partes móviles, se unen varias primitivas de colisión que formen la estructura del objeto, esto se lo realiza a través de junturas.

Las primitivas de colisión son formas básicas, tales como cajas, esferas, capsulas (un cilindro con puntas redondas, como algunos tipos de pastillas), rayos/ segmentos de línea, así como un poco de formas más avanzadas como planos, mapas de alturas y mallas de triángulos. Para describir el comportamiento aproximado de un cuerpo físico determinado (vehículos, columpios, puertas, etc.) es posible utilizar más de una forma y juntarlos. Esto se lo realiza mediante una máscara de colisión y la asignación a cada una de las partes del cuerpo.

Control de Flujo.

En la simulación, se debe crear una instancia del sistema de física. Este sistema es la parte del código que va a simular el movimiento de los cuerpos. El simulador de la física

sabe cómo tomar las colisiones y objetos rígidos de modo que el comportamiento sea lo más parecido a la realidad.

El simulador de física se integra a la aplicación llamando al método PhysicsSystem.Integrate() al que se pasa una cantidad de tiempo desde la última actualización del componente GameComponent.Update() como 1/100 segundos o 0.01 segundos, este parámetro sirve para sincronizar el flujo de tiempo de la aplicación con el simulador de física.

Características

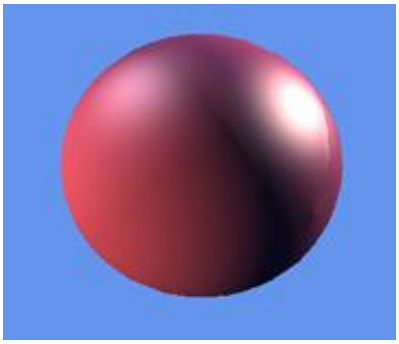
Las unidades utilizadas son néwtones, metros, néwtones-metros, kilogramos y segundos, también conocidos como SI (Sistema internacional) o en unidades métricas. Es posible escalar estas unidades para sistemas de medición diferentes, pero es más fácil convertir todo a un sistema métrico.

Si se desea incorporar un cuerpo para estar en el mundo, pero que no se mueva, se establece la propiedad Immovable en true para que la gravedad no afecte dicho cuerpo y este no sea jalado hacia abajo, en caso de existir un mapa de alturas, este objeto será arrastrado hacia el suelo.

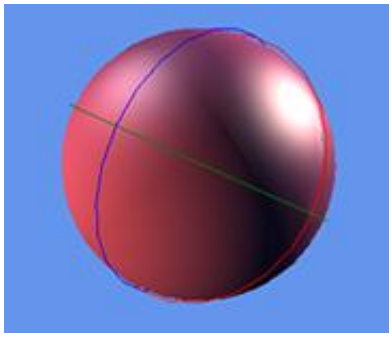
La cantidad de tiempo que debe ser pasado como parámetro al simulado de fisca debe ser pequeño para obtener buenos resultados de simulación. Un valor de 0.01 está bien, un valor demasiado pequeño y puesto que la simulación requiere de mucho cálculo haría que la ejecución sea lenta, un valor demasiado grande hará que la simulación sea menos estable. Teniendo en cuenta que el tiempo entre los fotogramas se actualicen y dibujen es a 60 Hz es de 0.017, que es algo superior a lo ideal, pero por lo general funciona bien.

En la simulación de vehículos se pude observar que es un objeto más completo en su conjunto. Este objeto no hace uso de esferas para las ruedas, sino que utiliza un cierto número de rayos, expulsados en la dirección de la rueda, para detectar el suelo.

Figura 50. Máscara de colisión



Modelo de una esfera



Modelo con una primitiva de colisión

Fuente: El autor

La integración de este motor de física al motor del videojuego como un componente que forme parte de este en primera instancia es sencilla, pero a lo largo del desarrollo del proyecto se vio en la necesidad de modificar el código de esta librería para que se adapte

a las necesidades del proyecto y que es necesario mencionar, para que en proyectos futuros se las toma en cuenta.

Un importante cambio se dio en la clase Car perteneciente al proyecto de JigLibX, para simular el comportamiento de un vehículo como una entidad física perteneciente al videojuego.

A primera vista la simulación de física sobre vehículos es aceptable, pero al probar algunas de las características propias de los vehículos se notó un error es su comportamiento, el cual era que cuando el vehículo al alcanzar su máxima velocidad este empezaba a tambalear y hacer movimientos erráticos. Para solucionar este problema fue necesario de cambios en el código, en las clases pertenecientes al objeto vehículo.

Antes de iniciar con el detalle de los cambios realizados en esta librería es importante conocer como la física influye sobre los objetos del mundo real, para esto a continuación se menciona las leyes del movimiento.

Las Leyes de Newton, también conocidas como Leyes del movimiento de Newton, son tres principios a partir de los cuales se explican la mayor parte de los problemas planteados por la dinámica, en particular aquellos relativos al movimiento de los cuerpos.

Primera ley de Newton o Ley de la inercia

Todo cuerpo persevera en su estado de reposo o movimiento uniforme y rectilíneo a no ser que sea obligado a cambiar su estado por fuerzas impresas sobre él.

Segunda ley de Newton o Ley de fuerza

El cambio de movimiento es proporcional a la fuerza motriz impresa y ocurre según la línea recta a lo largo de la cual aquella fuerza se imprime.

Tercera Ley de Newton o Ley de acción y reacción

Con toda acción ocurre siempre una reacción igual y contraria: o sea, las acciones mutuas de dos cuerpos siempre son iguales y dirigidas en sentido opuesto.

Fuente: Wikipedia

La clase Car dentro del proyecto JigLibX es la que contiene las demás subclases para formar el objeto vehículo estas son: chassis y wheel que como podrá notarse representa las partes más importantes de un vehículo, sin tomar en cuenta la mecánica de vehículos.

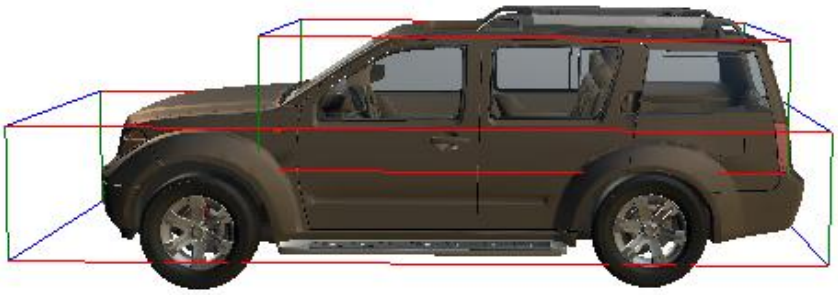
Los cambios más representativos se dieron en las clases car y wheel ya que es en estas clases donde se aplican las configuraciones sobre el comportamiento físico de un vehículo, la clase car se encarga de formar el objeto vehículo aplicando configuraciones propias de la física de vehículos como: fricción, ángulo de giro de las llantas, distancia de viaje, fracción de resistencia de las llantas, radio de las llantas, gravedad independiente del sistema físico, etc. Par resolver el problema de tambaleo del vehículo se notó que en la clase wheel como parámetro solo se ingresaba la fricción causada por las llantas

delanteras, a partir de esto se requirió el soporte para la fricción causada por las llantas traseras, con estos cambios aplicados en esta clase fue necesario modificar la clase car para aplicar las demás configuraciones que serían ingresados como parámetros al instanciar el objeto Car.

Con pequeños pero importantes cambios en el comportamiento físico de vehículos se arregló el problema, dando como resultado una mejor simulación de física.

Figura 51. Máscara de colisión aplicado a un vehículo

Vehículo modificado en el ejemplo JigLibX



Máscara de colisión para la integración con el sistema físico

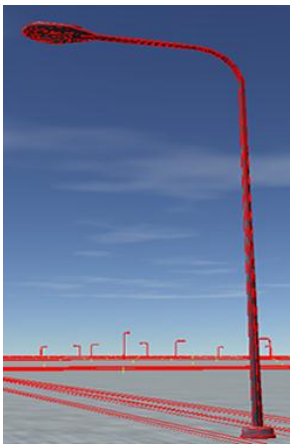
Fuente: El autor

A lo largo del desarrollo del proyecto todos los personajes que presentan en alguna forma parte de la jugabilidad tienen una primitiva de colisión.

Los objetos que formaban parte de los escenarios de los videojuegos desarrollados, pero que no eran objetos dinámicos, es decir no tienen movimiento como resultado de aplicar física sobre ellos estos pueden ser: paredes, puertas, escaleras, etc. se aplicó una primitiva especial para cuerpos rígidos.

Figura 52. Cuerpos rígidos

Parte del escenario del videojuego



Máscara física de cuerpos rígidos

Fuente: El autor

La máscara de colisión para los cuerpos rígidos no son formas básicas de las primitivas que se usan para emular física sobre objetos, sino que son marcaras construidas a partir del modelo, es decir, en otra parte de este documento se mencionó sobre el modelado 3D de objetos que pertenecerán al videojuego y que eran esto modelos que básicamente son mallas formada por vértices para formar polígonos, a partir de esta definición para la integración con el sistema de física, a cada modelo que sea estático o rígido se le extrae la información de estos vértices que son guardados en un diccionario de datos y a partir de este construir triángulos para luego crear una malla de colisión propia a partir del modelo. Como puede verse en el grafico anterior, la máscara física de un cuerpo rígido tiene similitud como si se lo estuviera viendo desde una perspectiva de renderización wireframe.

Los cálculos de colisiones sobre cuerpos rígidos son menores que los objetos dinámicos.

Figura 53. Parte de una escena física del videojuego desarrollado para un solo jugador

Fuente: El autor

Para que los personajes del videojuego se integren con el sistema de física se utilizó capsulas como primitivas de colisión, que como puede verse en el gráfico anterior, envuelven a los personajes, obviamente que las capsulas no son las mismas debido a que los personajes son de deferentes dimensiones.

Los objetos físicos que pertenecen al videojuego son fáciles de implementar en cualquier proyecto, pero se tiene que tener cuidado en la cantidad de estos y en las propiedades de las primitivas de colisión, ya que si se excede o si las propiedades están mal configuradas podría dar como resultado la ejecución de la aplicación con un número reducido de frames por segundo.

El presente proyecto no pretende adentrarse a fondo es aspectos avanzados de cómo desarrollar videojuegos, como se mencionó anteriormente es una tarea compleja donde intervienen varias personas e incluso en la actualidad empresas de desarrollo se dedican exclusivamente a esta actividad. La implementación de la física en el proyecto desarrollado fue una parte fundamental a tomarse en cuenta, y aunque al principio se intentó desarrollar un motor de física propio, esta idea fue inmediatamente desechada, por dos razones: la primera fue el nulo conocimiento de cómo representar física del mundo real en un programa de computadora y es importante mencionar que las formulas y ecuaciones matemáticas que son utilizadas para el cálculo en eventos físicos son demasiado complejas, la segunda razón era la de cumplir con el desarrollo del proyecto en el tiempo establecido, y ya que había un motor de física open-source disponible para su uso, que mejor que utilizar lo que ya ha sido desarrollado.

3.2.10 Inteligencia Artificial (IA)

Una de las principales características de los videojuegos actuales, es la inteligencia que presentan los objetos que interactúan de alguna manera con el jugador, objetos que no son controlados por el jugador, pero que responden de alguna manera al entorno del videojuego.

La Inteligencia Artificial, probablemente suena un poco complicado y al mismo tiempo interesante, desde el principio de la era de la computación, algunas investigaciones han propuesto y debatido caminos para hacer que los actos humanos puedan ser realizados por maquinas, el problema más grande con la entera línea de la ciencia de la Inteligencia Artificial es ¿qué es realmente la inteligencia?, pregunta que tal vez es difícil de contestar, numerosas otras preguntas surgen acerca de este tema: ¿cómo se define el típico comportamiento humano? ¿Qué formas de comportamiento humano constituyen inteligencia? ¿Qué formas de comportamiento humano podría ser replicados por una maquina?

Definir que es inteligencia es complicado, que acciones pueden ser interpretadas como inteligentes y si estas pueden ser copiadas por maquinas.

Por ejemplo en el tema de videojuegos, que una imagen se mueva de izquierda a derecha mientras continuamente reproduce una animación, esto no es muy inteligente, en otras palabras no hay inteligencia, quizás porque la imagen no hace otra cosa más que moverse.

Se denomina inteligencia artificial (IA) a la rama de las ciencias de la Computación dedicada al desarrollo de agentes racionales no vivos.

Para explicar la definición anterior, entiéndase a un agente como cualquier cosa capaz de percibir su entorno (recibir entradas), procesar tales percepciones y actuar en su entorno (proporcionar salidas), y entiéndase a la racionalidad como la característica que posee una elección de ser correcta, más específicamente, de tender a maximizar un resultado esperado. De acuerdo al concepto previo, racionalidad es más general y por ello más adecuado que inteligencia para definir la naturaleza del objetivo de esta disciplina.

Por lo tanto, y de manera más específica la inteligencia artificial es la disciplina que se encarga de construir procesos que al ser ejecutados sobre una arquitectura física producen acciones o resultados que maximizan una medida de rendimiento determinada, basándose en la secuencia de entradas percibidas y en el conocimiento almacenado en tal arquitectura.

Fuente: Wikipedia

Prueba de Turing

Es una prueba propuesta por Alan Turing para demostrar la existencia de inteligencia en una máquina. Fue expuesto en 1950 en un artículo (Computing machinery and intelligence) para la revista Mind, y sigue siendo uno de los mejores métodos para los defensores de la Inteligencia Artificial. Se fundamenta en la hipótesis positivista de que, si una máquina se comporta en todos los aspectos como inteligente, entonces debe ser inteligente.

La prueba consiste en un desafío. Se supone un juez situado en una habitación, y una máquina y un ser humano en otras. El juez debe descubrir cuál es el ser humano y cuál es la máquina, estándoles a los dos permitido mentir al contestar por escrito las preguntas que el juez les hiciera. La tesis de Turing es que si ambos jugadores eran suficientemente hábiles, el juez no podría distinguir quién era el ser humano y quién la máquina. Todavía ninguna máquina puede pasar este examen en una experiencia con método científico.

Una de las aplicaciones de la prueba de Turing es el control de spam. Dado el gran volumen de correos electrónicos enviados, el spam es, por lo general, enviado automáticamente por una máquina. Así la prueba de Turing puede usarse para distinguir si el correo electrónico era enviado por un remitente humano o por una máquina (por ejemplo por la prueba Captcha).

Inteligencia Artificial en el proyecto desarrollado

Durante la investigación de este tema, se pudo tener acceso al código fuente a uno de los videojuegos más populares que ha existido en la historia de este mercado, y que fue liberado, este es Quake III Arena, escrito en C++, por la empresa ID Software, la principal característica de este producto fue la avanzada inteligencia artificial aplicada a los personajes del videojuego, avanzada para la época, pero aunque se tuvo acceso a los

fuentes de este videojuego, no fue posible la implementación de los algoritmos de IA para el proyecto que se estaba desarrollando, principalmente por la complejidad de estos.

Los personajes que no son controlados por el jugador tienen inteligencia artificial básica, los personajes solo presentan 4 estados por llamarlos inteligentes, estos son:

- Deambulatorio
- Persecución
- Ataque
- Muerte

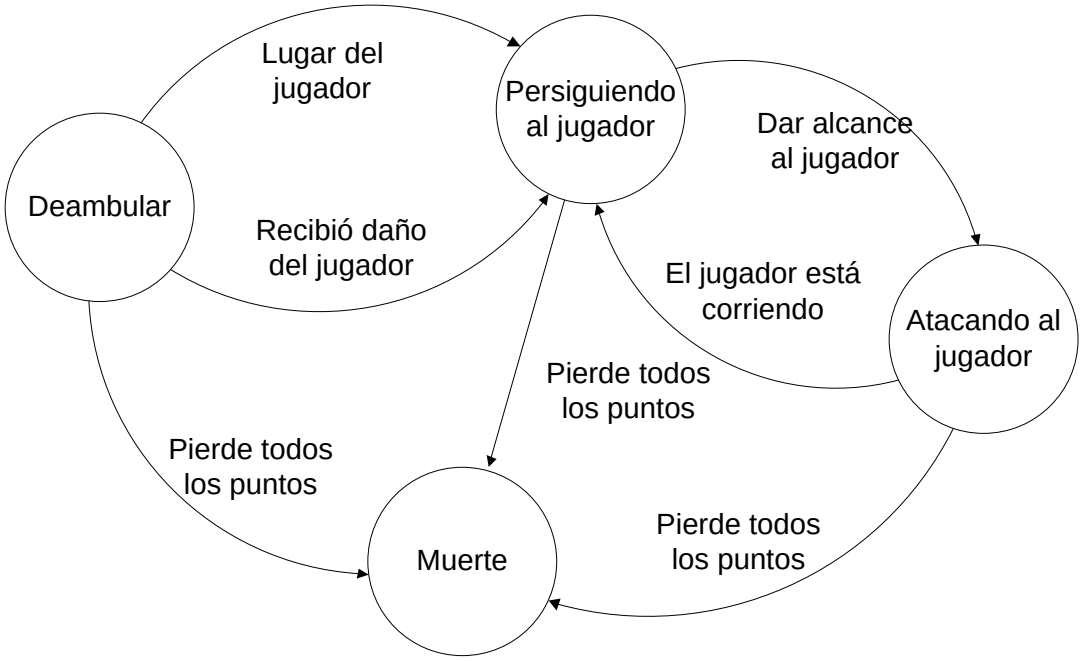
El estado deambulatorio inicialmente el personaje esta en este estado, las únicas razones para el cambio de estado son que el jugador este cerca del objeto de acuerdo a un rango, y que el jugador ataque a este personaje, en este caso el personaje, atacará al jugador.

En persecución luego de recibir daño, o estar cerca del jugador, este perseguirá al jugador, cabe mencionar que en este estado, el personaje no podrá esquivar obstáculos.

En ataque como todos los personajes atacan al jugador “mordiendo”, estos deben estar a una cierta distancia del jugador para atacar, si el jugador se mueve, los personajes cambiaran de estado a persecución.

Muerte si el personaje ha recibido el suficiente daño, este desaparecerá.

Figura 54. Diagrama de Inteligencia Artificial de los personajes no controlados



Fuente: El autor

La debilidad presentada en la inteligencia artificial aplicada a este proyecto y como se mencionó anteriormente es la falta de evasión de obstáculos por parte de los personajes controlados por el computador, razón por la cual y como se verá en la aplicación, si un objeto choca con el escenario físico, mientras está en movimiento, este puede mantenerse en esa posición, ya que no está programado para evadir objetos, en un futuro se puede implementar algoritmos eficientes para superar este inconveniente.

En este tema como se notará, no solo se debe tener conocimiento de programación en algún lenguaje, sino también, conocimiento en otros campos como: matemática, estadística, etc.

Para los cálculos de distancia entre el jugador y el objeto, interpolaciones de puntos para definir trayectorias, ángulos de giro entre el jugador y el oponente, etc.

3.2.11 Sistema de Partículas

El sistema de partículas se refiere a una técnica de gráficos por ordenador para simular ciertos fenómenos difusos, que son muy difíciles de reproducir con técnicas convencionales de representación. Ejemplos de estos fenómenos que suelen reproducirse utilizando sistemas de partículas incluyen incendios, explosiones, humo, agua en movimiento, las chispas, la caída de las hojas, las nubes, niebla, nieve, polvo, las colas de meteoros, pelo, piel, pasto, o efectos visuales abstractos.

Mientras que en la mayoría de los casos los sistemas de partículas se aplican en sistema de gráficos de tres dimensiones, los sistemas de partículas en dos dimensiones son usados en algunas circunstancias.

Por lo general la posición de un sistema de partículas y movimiento en el espacio 3D son controlados por lo que se conoce como emisor. El emisor actúa como la fuente de las partículas, y su ubicación en el espacio 3D determinando dónde se generan y de dónde proceden. Un objeto regular 3D, como un cubo o un avión, puede ser utilizado como una fuente de emisión. El emisor tiene un conjunto de parámetros de comportamiento de las partículas. Estos parámetros pueden incluir la tasa de aparición (cuántas partículas se generan por unidad de tiempo), el vector con la velocidad inicial de las partículas (la dirección en que se emiten desde su creación), el tiempo de vida de la partícula (la longitud de tiempo que cada partícula individual existe antes de desaparecer), color de las partículas, y muchos más. Es común que todos o la mayoría de estos parámetros puede ser "difuso" - en lugar de un valor numérico exacto, se establece un valor central y el grado de aleatoriedad permitido a cada lado del centro (es decir, la vida de la partícula promedio podría ser de 50 frames de más o menos el 20%). Cuando se utiliza un objeto como emisor, el vector velocidad inicial es a menudo dispuesto a ser normal a la cara de la persona (s) del objeto, por lo que las partículas parecen “spray”.

Un típico bucle de actualización de un sistema de partículas se pueden separar en dos fases distintas, la actualización de parámetros / etapa de simulación y la etapa de procesamiento.

Durante la etapa de simulación, el número de nuevas partículas que se debe crear se calcula sobre la base de la tasa de aparición y el intervalo entre las actualizaciones, y cada uno de ellos se genera en una posición específica en el espacio 3D basado en la posición del emisor y la zona de aparición especificado. Cada uno de los parámetros de la partícula (es decir, la velocidad, color, etc.) se inicializa de acuerdo a los parámetros del emisor. En cada actualización, todas las partículas existentes son revisadas para ver si se han excedido su tiempo de vida, en cuyo caso se retiran de la simulación. De lo contrario, la posición de las partículas y otras características avanzadas están basadas en algún tipo de simulación física, que puede ser tan simple como la traslación de su posición actual, o tan complicado como la realización de los cálculos de trayectoria físicamente con precisión que tengan en cuenta las fuerzas externas (gravedad, la fricción , el viento, etc.) Es común para llevar a cabo algún tipo de detección de colisiones entre las partículas y especificar los objetos 3D en la escena para hacer que las partículas reboten o de otra manera interactuar con los obstáculos en el medio ambiente. Las colisiones entre partículas se usan muy poco, ya que son computacionalmente costosas y no es realmente útil para la mayoría de las simulaciones.

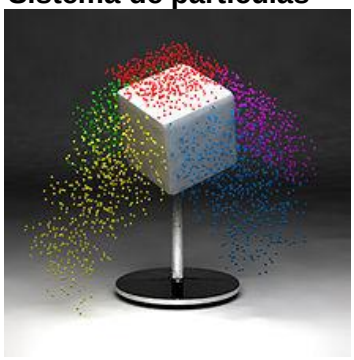
Después de la actualización, cada partícula se representa, por lo general en forma de un quad billboarded texture (es decir, un cuadrilátero que está siempre de cara al espectador). Sin embargo, esto no es necesario, una partícula puede ser traducida como un solo píxel en pequeños y limitados ambientes.

Los sistemas de partículas pueden ser animados o estáticos, es decir, el tiempo de vida de cada partícula puede ser distribuido en el tiempo o renderizado todo de una vez. La consecuencia de esta distinción es la diferencia entre la aparición de “nieve” y la aparición de “pelo”.

El término “sistema de partículas” en sí mismo a menudo se trae a la mente sólo el aspecto de animación, que se usa comúnmente para crear simulaciones de partículas en movimiento como chispas, lluvia, fuego, etc. En estas implementaciones, cada frame de la animación contiene cada partícula en una posición específica en su ciclo de vida, y cada partícula ocupa un lugar único en el espacio.

Sin embargo, si el ciclo de vida completo de la partícula de cada uno se representa al mismo tiempo, el resultado es partículas estáticas, filamentos de material que muestran la trayectoria de las partículas en general, en lugar de partículas puntuales. Estos filamentos se pueden utilizar para simular el pelo, piel, pasto, y materiales similares. Los hilos se pueden controlar con los vectores de velocidad de la misma, campos de fuerza, los índices de aparición, y los parámetros de la desviación que las partículas de animación deben obedecer. Además, los espesores prestados de los hilos se pueden controlar y, en algunas implementaciones se puede variar a lo largo de la cadena. Las diferentes combinaciones de parámetros pueden impartir rigidez, flacidez, pesadez, nitidez, o cualquier otra propiedad. Los hilos también pueden utilizar la asignación de texturas para variar el color de la tira, la longitud, u otras propiedades en toda la superficie del emisor.

Figura 55. Sistema de partículas



Un cubo emitiendo 5000 partículas animadas obedeciendo una fuerza gravitacional en la dirección negativa de Y

El mismo cubo con el emisor renderizando partículas estáticas

Fuente: Wikipedia

El desarrollo de un sistema de partículas puede resultar de lo más complejo pero se pueden lograr efectos interesantes. Para conseguir este objetivo se utilizó una librería que resuelve esta complejidad, esta librería no está disponible en codeplex, tiene su propia página web, es de uso libre pero no está disponible el código fuente.

DPSF (Dynamic Particle System Framework)

Versión actual: 1.5.3.0 – Noviembre 10, 2010

Versión utilizada: 1.5.0.0

Desarrollador: Daniel Schroeder mientras era estudiante en la Universidad de Regina, bajo la supervisión del Dr. Howard Hamilton.

Introducción

DPSF (Framework dinámico de sistema de partículas) es una herramienta libre para programadores para la creación de sistemas de partículas a la medida en XNA de una forma rápida y sencilla.

Incorpora efectos de partículas en proyectos propios en cuestión de minutos utilizando las clases o plantillas por defecto.

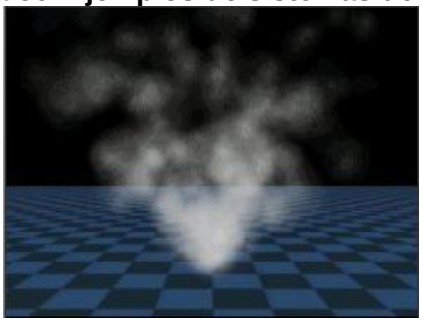
A diferencia de otros sistemas de partículas API / bibliotecas, DPSF es flexible y permite a través de código personalizar el comportamiento en el sistema de partículas, no se tiene limitación al usar solo los parámetros provistos por el framework, se puede crear y controlar propiedades propias haciendo cualquier cosa que se pueda imaginar.

Características

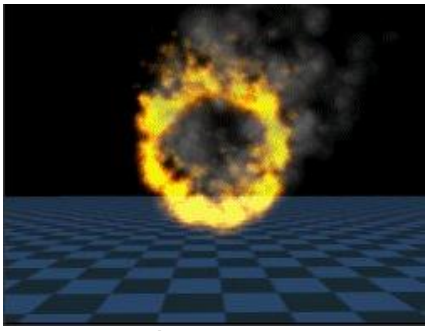
- Común funcionalidad del sistema de partículas es siempre, como la gestión de las partículas en la memoria, el dibujo, y el emisor.
- Plantillas para empezar que se proporcionan para hacer la creación de nuevos sistemas de partículas rápida y fácil.

- Se escribe el código de sistema de partículas, lo que le permite crear cualquier tipo de efectos sobre el sistema de partículas que se desee. La imaginación es el límite.
- Permite al sistemas de partículas crear en cuestión de minutos efectos personalizados utilizando la configuración predeterminada del sistema de partículas que se proporciona; Sólo cambiando valores establecidos en los parámetros, como la posición, velocidad, aceleración, rotación, fuerza externa, inicio / final de color, etc.
- El sistema de partículas por defecto puede ser extendido, lo que le permite ofrecer toda la funcionalidad requerida adicional.
- Control total sobre el sistema de partículas y sus partículas.
- Permite efectos innovadores a ser creados rápida y fácilmente a través del uso de eventos.
- Soporta el dibujo de partículas como píxeles, Sprites (utilizando un SpriteBatch), Point Sprites, Quads, y Textured Quads.
- Fácil de integrar en proyectos existentes, sólo tiene que añadir una referencia a un archivo dll.
- Uso de built-in Effect (es decir, shaders), así como efectos personalizados.
- Modificar el efecto por defecto (es decir, shaders) para crear nuevos efectos personalizados de forma rápida y sencilla.
- Un administrador del sistema de partículas es provisto para hacer la actualización y dibujado de muchos sistemas de partículas más fácil.
- Una clase de animaciones se proporciona para crear fácilmente las partículas de animación.
- Fácil de crear una secuencia de imágenes, o gifs animados que muestra animaciones de partículas en el sistema de partículas.
- Los sistemas de partículas pueden aplicarse como DrawableGameComponents si es necesario.
- Funciona en Windows y Xbox 360 (en la actualidad no probados Zune).
- Completa documentación de la API se proporciona en el archivo de ayuda, así como en la documentación de ayuda en línea.
- Tutoriales y código fuente se proporcionan en el instalador. Los tutoriales (menos el código fuente) están también disponibles en la documentación de ayuda en línea.

Figura 56. Ejemplos de sistemas de partículas



Sistema de partículas para simular humo



Sistema de partículas para simular fuego y humo

Fuente: El autor

Con la utilización de esta librería de sistema de partículas y con la ayuda de los tutoriales que lo acompañan, su integración en el proyecto resulto sencilla, algunos de los sistemas de partículas utilizados se listan a continuación.

Figura 57. Disparos y humo de las armas utilizadas por parte del jugador.

Figura 58. Peligro al acercarse a un enemigo.

Figura 59. Interacción con el jugador.

Figura 60. Daño en vehículos

Figura 61. Explosión de objetos

Estos son algunos de los sistemas de partículas integrados en el proyecto, como ya se mencionó un sistema de partículas sirve para ambientar un escenario. La fácil utilización de esta librería permitió añadir efectos especiales al videojuego y esa fue la principal característica de su utilización, con poco código en las plantillas de los sistemas de partículas se puede conseguir efectos avanzados.

La falta del código fuente de la librería de sistema de partículas no fue un impedimento para su utilización, la amplia documentación y ejemplos que trae consigue desde donde

es descargado hizo que su integración dentro del motor videojuegos del proyecto sea sencilla y rápida.

3.2.12 Soporte de Red

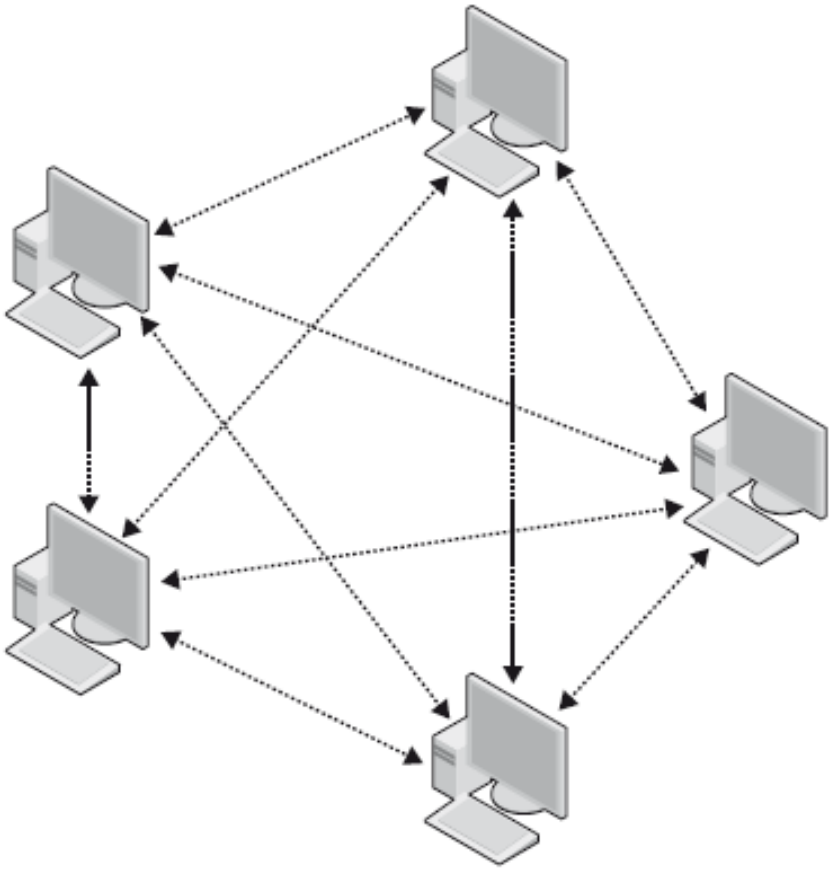
Para el desarrollo de la versión multijugador del videojuego desarrollado se partió del motor ya construido y que sirvió para el desarrollo de esta versión.

Antes de comenzar con la descripción del soporte de red para uno de los videojuegos desarrollado es necesario mencionar algunos conceptos que serán utilizados a lo largo de esta sección.

Una red de computadoras, también llamada red de ordenadores, es un conjunto de equipos conectados por medio de cables, señales, ondas o cualquier otro método de transporte de datos, que compartan información (archivos), recursos (CD-ROM, impresoras, etc.), servicios (acceso a internet, e-mail, chat, juegos), etc.

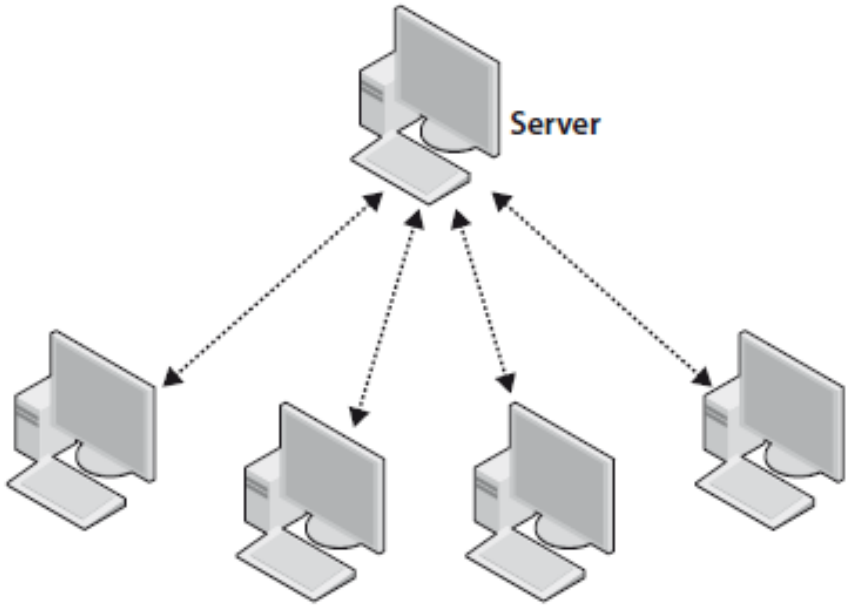
Topologías de red

Figura 62. Típica red punto a punto, todas las computadoras interactúan con las demás.



Fuente: El autor

Figura 63. Típica red cliente/servidor, todos los mensajes son enviados y recibido a través del servidor.



Fuente: El autor

Para los videojuegos, las topologías más comunes son las conexiones punto a punto (todos los mensajes se envían a todo el mundo por parte de cada máquina), y la topología cliente/servidor (cada máquina-jugador envía los datos a un servidor central, y este redistribuye la información). Se tiene que tener en cuenta que el servidor puede ser una máquina dedicada, o una de las máquinas de los propios jugadores, que también puede actuar como tal.

Es una decisión de diseño importante seleccionar la tipología de red, ya que por ejemplo, la tipología punto a punto puede llegar a consumir todo el ancho de banda disponible si hay muchos jugadores conectados.

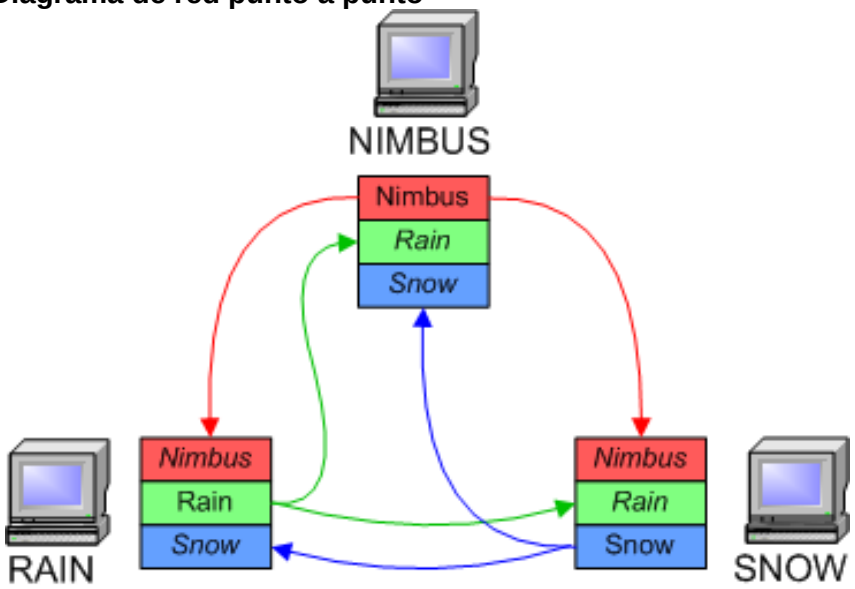
Red punto a punto

En un juego con topología punto a punto, el trabajo de actualización de la lógica del juego, movimiento de objetos, la física, comprobación de colisiones, etc. se distribuye en todas las maquinas en la sesión de red. Para evitar confusiones, cada objeto de juego debe ser actualizado por una sola maquina a la vez. Muy a menudo, una máquina controla todos los objetos pertenecientes a sus jugadores locales, en juegos sofisticados que incluyen otros objetos o personajes con IA que no están bajo el control directo del jugador, los objetos pueden ser compartidos para equilibrar la carga de trabajo de actualización de forma uniforme entre las máquinas disponibles.

Una vez que una máquina ha actualizado todos sus objetos locales, se envía un paquete de red que describe su último estado a todos los demás en la sesión. Las otras máquinas nunca tratarán de actualizar los objetos que ellos no controlan: en cambio, leen los paquetes de la red y actualizan sus copias locales del estado de los objetos.

Este diagrama muestra el flujo de datos para un juego de igual a igual con tres equipos en la sesión:

Figura 64. Diagrama de red punto a punto



Fuente: Ejemplo proporcionado por Microsoft en el sitio oficial de XNA.

Se puede ver que, aunque cada uno de los tres equipos tiene una copia local de la información de estado para cada uno de los tres jugadores, este estado sólo fluirá hacia fuera del ordenador que lo controla. Por ejemplo, el estado del jugador Nimbus, que se muestra en rojo, es siempre actualizada por la maquina NIMBUS, y la remitió a Rain y Snow, mientras que el estado del jugador Rain, que se muestra en verde, se actualiza en la maquina RAIN, entonces se transmite a Nimbus y Snow.

Una red punto a punto tiene varias ventajas sobre la topología cliente/servidor:

- Juegos punto a punto por lo general requieren menos ancho de banda que los juegos cliente / servidor.
- El ancho de banda de red y carga de trabajo de procesamiento de la CPU son compartidas entre los equipos, en lugar de apilar carga de trabajo en un único servidor.
- Dado que los objetos locales están siempre actualizados a nivel local, los jugadores no verán ningún tipo de retraso en el juego al responder sus peticiones.
- Debido a que el estado de simulación se distribuye en varios equipos, en los juegos punto a punto se puede apoyar la migración del anfitrión con más facilidad que cuando se utiliza la topología cliente/servidor.

Pero también tiene algunas desventajas:

- Como los equipos son diferentes, la actualización de las diferentes piezas del estado del juego es independiente, en los juegos punto a punto son más propensos a sufrir de problemas de consistencia en las decisiones que se tomen con base en la información del estado que no ha sido perfectamente sincronizados. Por ejemplo, si dos jugadores llegan a un power-up, al mismo tiempo, puede ser difícil para un juego punto a punto resolver con precisión cuál de ellos fue el primero.

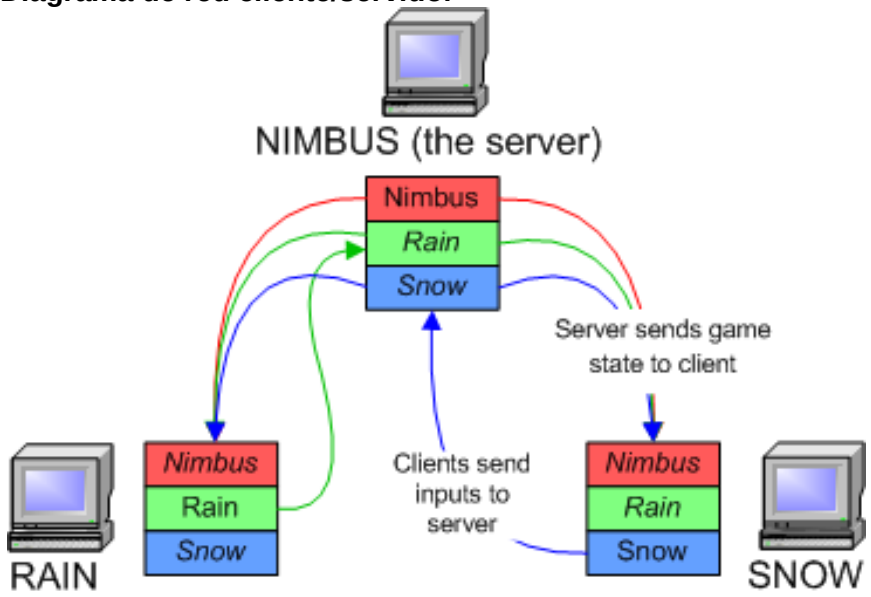
- Es más fácil para los hackers modificar el ejecutable de un juego punto a punto. Cuando se utiliza una topología cliente/servidor, cambiando el código del juego del cliente tiene un efecto menor ya que el cliente no es directamente responsable de actualizar el estado de simulación de juego.

Red cliente/servidor

En un entorno de juego cliente/servidor, el trabajo de actualización de la lógica del juego, movimientos de objetos, la física, la comprobación de colisiones, y así sucesivamente, se ejecuta por completo en un único equipo que ha sido elegido como el servidor. Con el sistema de redes del Framework XNA, el servidor es un equipo regular que también está jugando en la sesión. Esto es diferente a algunos juegos de Windows que utilizan un ordenador independiente dedicado como su servidor.

Esta figura muestra el flujo de datos para un cliente/servidor de juego con tres equipos en la sesión:

Figura 65. Diagrama de red cliente/servidor



Fuente: Ejemplo proporcionado por Microsoft en el sitio oficial de XNA

Como se puede ver cómo todos los datos de la red se enrutan a través del anfitrion, Nimbus, y no siempre los paquetes se envían directamente entre los dos clientes, Rain y Snow.

El equipo anfitrión también se utiliza como el servidor. Esto no siempre tiene que ser el caso, como los juegos son libres de elegir cualquier equipo como su servidor (o ninguno en absoluto si se está utilizando una red punto a punto). Por ejemplo, un juego que desee examinar los tiempos de ping entre todos los equipos de la reunión, y elegir el de menor promedio de ping como su servidor. El servidor no significa nada especial para el Framework XNA, por lo que los juegos son libres la máquina que será su servidor. El anfitrión, en cambio, tiene un especial tratamiento en su significado. El anfitrión es el equipo que creó originalmente el período de sesiones, y es responsable de la publicidad

a través de matchmaking para otras máquinas que pueden encontrarse y unirse a él. El anfitrión también es responsable de decidir cuándo cambiar los estados de juego. Si se habilita NetworkSession.AllowHostMigration, un nuevo anfitrión se seleccionará automáticamente si el anfitrión anterior deja la sesión (de lo contrario, la sesión terminará cuando el juego acaba). El anfitrión siempre es elegido por el XNA Framework, pero el servidor es elegido por el juego en sí.

Una red Cliente / servidor tiene varias ventajas sobre la topología alternativa punto a punto:

- Debido a que la simulación total del juego corre un una sola computadora, los juegos cliente/servidor son menos propensos a sufrir de problemas de consistencia.
- Debido a que todas las decisiones se toman por parte de un solo servidor, es más difícil para los hackers engañar al modificar los archivos ejecutables del juego del cliente.

Pero también tiene algunas desventajas:

- Los juegos cliente/servidor por lo general requieren más ancho de banda de red que los juegos punto a punto. El servidor debe enviar información sobre cada jugador para cada cliente, enviando así un total de *(playerCount * (playerCount − 1))* descripciones de estado. En comparación, en un juego punto a punto, cada equipo sólo envía su propio estado a los demás puntos de red, enviando así un total de *(playerCount - 1)* descripciones de estado.
- El ancho de banda de red y la carga de trabajo que debe ser procesada por la CPU provocaría una caída en el servidor, que fácilmente puede convertirse en un cuello de botella en el procesamiento.
- La red de ida y vuelta desde y hacia el servidor puede hacer que los jugadores en los equipos cliente a experimentar el retraso en el juego al responder sus peticiones. (El servidor no sufren este retraso, por lo que es una "ventaja del anfitrión", comportamiento que puede ser observado en muchos juegos cliente/servidor.)
- Debido a que el estado de simulación sólo existe en un solo equipo, el soporte de migración del servidor es más difícil que estar usando una red.

Independientemente de la topología seleccionada, las rede de computadores siempre presentan algunos problemas:

Latencia se denomina latencia a la suma de retardos temporales dentro de una red. Un retardo es producido por la demora en la propagación y transmisión de paquetes dentro de la red.

El problema de la latencia es un problema inevitable, la latencia siempre estará ahí, por muy buena conexión que se tenga, y por más que se optimice el juego, la red es algo que no es controlado por una persona (sobre todo si no es una red local). Por ello hay que tomar decisiones importantes durante la fase de diseño de la lógica de los juegos: por ejemplo, qué se hará si durante varios ciclos del juego no se conoce la nueva posición de

los demás jugadores, una posible aproximación sería presuponer que los demás jugadores mantienen la misma velocidad y dirección en cada instante, respecto al último dato conocido de los mismos. Aunque obviamente esto puede dar lugar a imprecisiones, puede reducir los efectos de "parpadeo", o "desaparición" de jugadores en la pantalla.

El framework XNA no tenía soporte para red en la versión 1.0, para solucionar se podía hacer uso de la librería System.Net en Windows, también no se podía tener acceso a la red sobre un consola de Xbox.

A partir de la versión 2.0 se agregó en la API de XNA soporte de red de alto nivel construido sobre Xbox Live y Games for Windows – LIVE con soporte de hasta 31 jugadores por sesión.

Figura 66. Pantalla de Games for Windows – LIVE

Fuente: El autor

Para crear y probar un juego en red es posible hacerlo a través de:

- Usando un sistema de enlace
- Solamente trabaja sobre una red local.
- Xbox requiere una subscripción a Creators Club.
- PC no requieren ninguna subscripción.

Para jugar un juego en red

- Uso de LIVE PlayerMatch
- Trabaja sobre internet (incluyendo una NAT de recorrido).
- Xbox y PC ambos requieren una subscripción LIVE Gold y Creators Club

Al utilizar XNA en el desarrollo de juegos con soporte de red se facilitan algunas de las tareas más comunes pero complejas para este tipo de aplicaciones como:

- Búsqueda y unión a sesiones de red.
- Sincronización de la lista de jugadores.

- Protocolo UDP confiable.
- Transmisión de vos.
- Migración del anfitrión.
- Simulación de latencia de red y pérdida de paquetes.
- Compresión de datos que son enviados en la red.
- Mitigar la latencia en la red con algoritmos de predicción e interpolación.

Los problemas que no se pueden controlar ya que son provocados por los medios físicos que componen la red básicamente son tres:

- Ancho de banda
- Latencia
- Perdida de paquetes

Considerando estos problemas la programación de un videojuego en red puede ser algo difícil, ya que se tiene que tomar en cuenta cómo afectará el rendimiento de la red y del juego, los paquetes que viajaran a través de la red, con este antecedente y antes de centrarse en la lógica del videojuego se debe responder las siguientes interrogantes que fueron planteadas al inicio del desarrollo del proyecto por el autor:

- ¿Qué topología se debe escoger?
- ¿Qué requerimientos debe tener un servidor de juegos?
- ¿Cómo debe ser el comportamiento del juego?
- ¿Qué información debe enviarse a los clientes de la red?
- ¿Qué información es de más importancia que otra?
- ¿A qué momento se deben actualizar el mundo del videojuego?
- ¿Qué información deben actualizar los jugadores locales?

Con la respuesta a estas preguntas se consiguió una amplia visión de cómo será el desarrollo del videojuego y como este se comportará. Estas preguntas serán contestadas a continuación:

La selección de la topología de red a utilizarse es lo más importante a tomar en cuenta a la hora de desarrollar un videojuego de este tipo. Con las ventajas y desventajas de cada una de las topologías anteriormente mencionadas se decidió implementar una topología cliente/servidor, pero el factor que más influyó en esta decisión fue que, dentro de los objetivos planteados era llevar el videojuego multijugador a Internet para que cualquier cliente que no sea parte de la red local se pueda unir a un servidor previamente configurado.

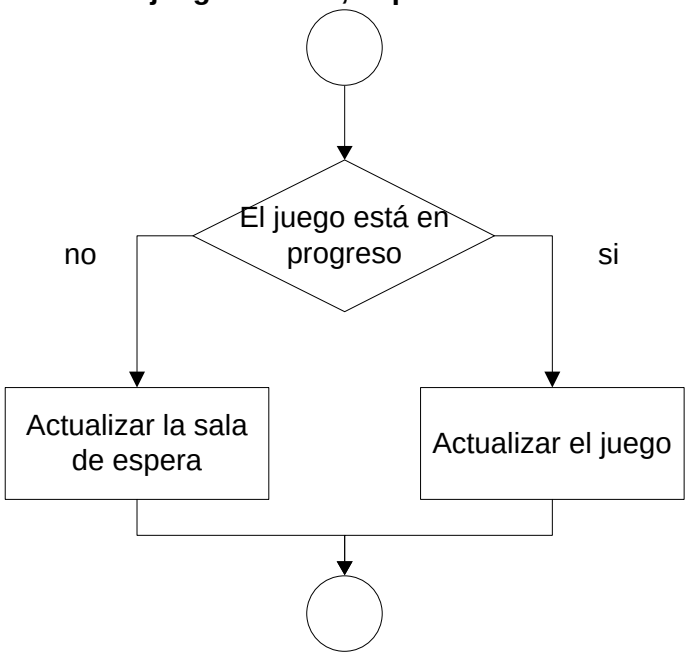
En un videojuego en red y gracias al API de red de XNA el servidor de juegos puede ser cualquier usuario de la red, independientemente de las capacidades de hardware y software que este puede tener, el único requerimiento es que pueda ejecutar el videojuego.

En la definición del proyecto se propuso que el videojuego será al estilo Quake III Arena para la versión multijugador con esto se define los pasos a seguir desde la creación de la sesión hasta la unión de un jugador a la sesión:

- La red debe estar previamente configurada y asegurar que funcione, es decir, que todas las maquinas que pertenecen a la red pueden comunicarse con las otras.
- La aplicación debe estar instalada en cada una de las máquinas y probar que se ejecute correctamente, caso contrario no podrá ser cliente de la sesión, sin tomar en cuenta cuál de las maquinas será el servidor ya que la aplicación es la misma para todos.
- El servidor será el encargado de configurar el mundo del videojuego e inicializarlo, además será el responsable de publicar esta sesión para que los demás clientes puedan unirse a ella.
- Los clientes deben buscar la sesión de red, en caso de haber más de una sesión deberá seleccionar la sesión y posteriormente unirse a ella.
- Cuando se cumpla la condición de victoria el servidor será el encargado de notificar que el juego ha terminado y de quien ha ganado.

Estas son todas las tareas vistas desde una perspectiva general que han sido realizadas dentro del videojuego. Como se podrá observar los clientes se unen a un juego en progreso, esta característica tuvo que ser programada ya que XNA no provee soporte para este mecanismo de juego, a continuación se explica que mecanismo de juegos en red utiliza XNA.

Figura 67. Ciclo de un videojuego en XNA, dependiendo si esta en progreso o no.



Fuente: El autor

Como se puede ver en el diagrama anterior, el servidor es quien inicia y crea la sesión, pero antes de iniciar el juego, los clientes deben ingresar a la sesión y marcarse como listos, una vez que todos los clientes de la sesión estén listos, el servidor puede inicializar el juego, el criterio para ganar la partida es definida de acuerdo al tipo de juego.

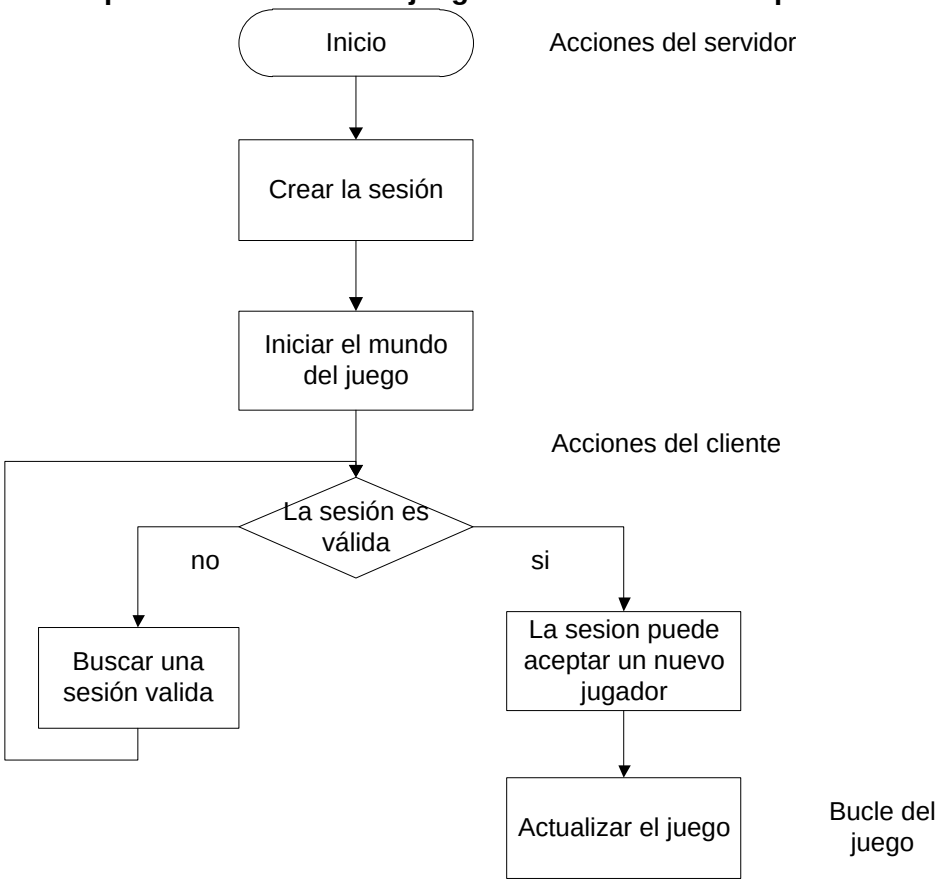
Ahora se explica cómo el mecanismo que se implementó, modificando el comportamiento por defecto de XNA para aplicaciones en red, a través de código para que soporte el ingreso de clientes en un juego en progreso.

Dentro de un videojuego en progreso y como única restricción por parte de este videojuego para la unión de clientes en una sesión o juego en progreso, es que el juego este activo y que no haya sido ganado, es decir que el juego sea válido tanto en su ejecución como en su jugabilidad.

El framework XNA ofrece sincronización en la lista de clientes de la sesión lo cual ayudo en gran parte a hacer más extensible el mecanismo de unión para juegos en red, la lista de clientes de la sesión de red de una maquina será la misma y en el mismo orden de las demás, y como aclaración con respecto en la lista de clientes de una sesión, el servidor siempre tendrá el índice 0 dentro de la sesión.

La salida de jugadores cuando el videojuego este en progreso, que puede deberse a fallas en la conexión o por voluntad del usuario, no hará que la aplicación falle, en estos casos XNA provee métodos para capturar estos eventos, cuando esto suceda, las listas de jugadores se actualizarán, se borraran todos los datos en el videojuego del usuario que abandono la partida, ya no se actualizará la información que manejaba dicho usuario, no tendrá representación gráfica dentro del videojuego, etc. además que cuando pase esto, se notificará a todos los miembros de la sesión que un participante ha salido.

Figura 68. Comportamiento del videojuego desarrollado con soporte de red.



Fuente: El autor

Algunas de las opciones que tiene el servidor antes de crear la sesión, no fueron activadas, como puede verse en la pantalla de menú de configuración del servidor de videojuegos, estas son, elegir el número máximo de participantes, simular que la sala de juegos está llena, hacer del servidor de videojuegos un servidor dedicado, etc. opciones de configuración que no se vio en la necesidad de programarlos pero que vale mencionarlos, la utilización de estas y demás opciones quedan a criterio de quien opte por activarlas, con las ventajas y desventajas que estas pueden presentar.

Modificando el mecanismo por defecto de XNA para la unión de nuevos jugadores e inicialización del juego se puede crear un nuevo comportamiento que fue el requerido por la aplicación.

Este mecanismo de soporte para la unión de nuevos jugadores en un juego en progreso, tiene una característica y ventaja ante el otro mecanismo antes mencionado y el que es por defecto de XNA, esta es que no se tiene que esperar que los jugadores se marquen como listos antes de iniciar el juego dando como resultado que cualquier jugador se una a la sesión en cualquier momento.

Dependiendo del tipo de videojuego a desarrollarse, se tendrá que seleccionar que tipo de mecanismo será el mejor a implementar, obviamente que el mecanismo implementado en esta versión del videojuego fue la mejor opción y la que se adaptaba a las necedades del proyecto, una ejemplo claro donde implementar el mecanismo por defecto de XNA se manifiesta en los videojuegos de competencia o estrategia, donde un jugador tiene que derrotar a otro, en este caso se debe esperar que todos los jugadores que pertenecen a la sala del juego ingresen a la sesión, marcarse como listos y posteriormente el servidor iniciar el videojuego.

Una vez definido el comportamiento del videojuego, es necesario jerarquizar la información que viajara a través de la red, es decir, que información la cual viaja a través de la red es necesaria que llegue a su destino para su procesamiento, como se mencionó antes, la latencia y perdida de paquetes siempre estará presente dentro de una red de computadores.

Tradicionalmente, los juegos había que preocuparse por:

- Los paquetes no se entreguen.
- Los paquetes se entregan en el orden incorrecto.
- Daño en los paquetes de datos.
- Los paquetes están manipulados por tramposos.
- Lectura accidental de paquetes de algún otro programa.
- Lo paquetes de datos pueden ser examinados cuando son transmitidos.

El Framework XNA ayuda con todos estos problemas

- Los paquetes no se entreguen. UDP confiable (opcional)
- Los paquetes se entregan en el orden incorrecto. Entrega en orden (opcional)
- Daño en los paquetes de datos. Paquetes seguros
- Los paquetes están manipulados por tramposos. Paquetes seguros
- Lectura accidental de paquetes de algún otro programa. Paquetes seguros

- Lo paquetes de datos pueden ser examinados cuando son transmitidos. Paquetes seguros

Los paquetes de datos no son más que los datos que se envían de un computador a otro.

La información que debe enviarse debe ser seleccionada de acuerdo a la función de actualización de los jugadores que pertenecen a la sesión, es decir, un jugador debe enviar datos de sí mismo a los demás para que estos sepan qué hacer con los jugadores que no son locales para actualizarlos en su juego local. Sin embargo debido a que el paquete que es enviado por el jugador local a la red este también lo recibe en este caso, el jugador local no debe procesar este paquete ya que, el jugador local es el encargado de actualizar su propia información.

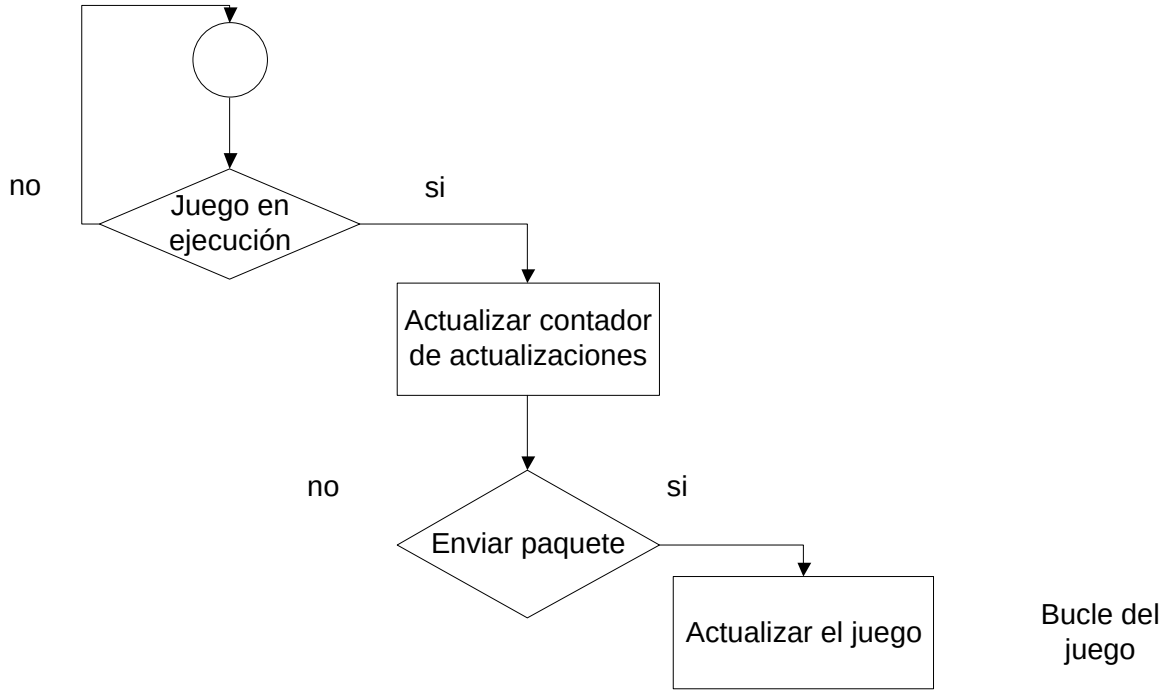
En la programación de videojuegos con soporte de red se debe priorizar la información y analizar qué información es de más importancia que otra, XNA dentro del API para el soporte de red tiene opciones especiales a la hora de enviar un paquete estas son:

- **None.** Envía los datos sin garantía. Los paquetes de este tipo se pueden entregar en cualquier orden, con la pérdida de paquetes de vez en cuando. Esta es la opción más eficiente en términos de ancho de banda de red y el uso de los recursos de la máquina. Sin embargo, se recomienda sólo en situaciones en las que el juego puede recuperarse de la pérdida de paquetes ocasional.
- **Reliable.** Envía los datos con la entrega confiable, pero no en un orden especial. Los paquetes de este tipo se vuelven a enviar hasta que lleguen a su destino. Estos pueden llegar fuera de orden.
- **InOrder.** Envía los datos con un orden garantizado, pero sin la entrega confiable. En ocasiones, los paquetes de este tipo no se entregan. Sin embargo, los paquetes entregados siempre llegan en el orden en que se envían. Se utiliza esta opción en situaciones donde los valores transmitidos cambian constantemente. Las versiones antiguas nunca llegan después de una versión más reciente.
- **ReliableInOrder.** Envía los datos con fiabilidad y llegada en el orden que fueron enviados originalmente. Los paquetes de este tipo se vuelven a enviar hasta que lleguen y ordenados internamente. Esto significa que llegan en el mismo orden en que fueron enviados. En términos de uso de ancho de banda de la red, esta es la opción más fuerte y más costosa. Se la utiliza únicamente cuando el arribo y orden de los datos son esenciales. Comúnmente, un juego utiliza esta opción para un pequeño porcentaje de paquetes. La mayoría de los datos del juego son enviados usando None o Reliable.
- **Chat.** Marca que este paquete contiene datos de chat, como una cadena de mensaje de jugador a jugador ingresado con el teclado. Para cumplir con las normas internacionales, se debe enviar esos datos sin cifrado de paquetes. Por lo tanto, se debe utilizar esta opción para marcarlo. Para mantener la seguridad, otros datos de juego no debe usar esta opción. Es aceptable y eficiente para mezclar los datos cifrados y sin cifrar. Si se envía los paquetes con y sin esta opción en un frame único, tanto los datos cifrados y no cifrados se combinan en un paquete único. Esta opción se puede combinar con uno o ambos de estas opciones Reliable e InOrder. Cuando se solicita la entrega en orden para paquetes de chat, estos tienen un orden relativo para otros paquetes de chat, pero pueden llegar fuera de orden con respecto a otros los no datos de chat.

Con estas opciones, con su respectiva definición y funcionamiento es preciso clasificar la información de acuerdo a su importancia para la red, obviamente optimizando el consumo de ancho de banda sin afectar la ejecución del videojuego, en el transcurso del desarrollo de esta versión del videojuego en su mayoría se utilizó la opción InOrden y como se menciona en las definiciones de cada una de la opciones, en un porcentaje menor se utilizó la opción ReliableInOrder solo cuando los datos debían ser garantizados que lleguen y en orden, en el videojuego se implementó un chat en el cual obviamente se utilizó la opción Chat.

Antes de enviar los paquetes a la red se debe analizar a qué momento deben enviarse esta información, para no inundar la red con mucha información, este aspecto fue considerado y la solución fue de la siguiente manera:

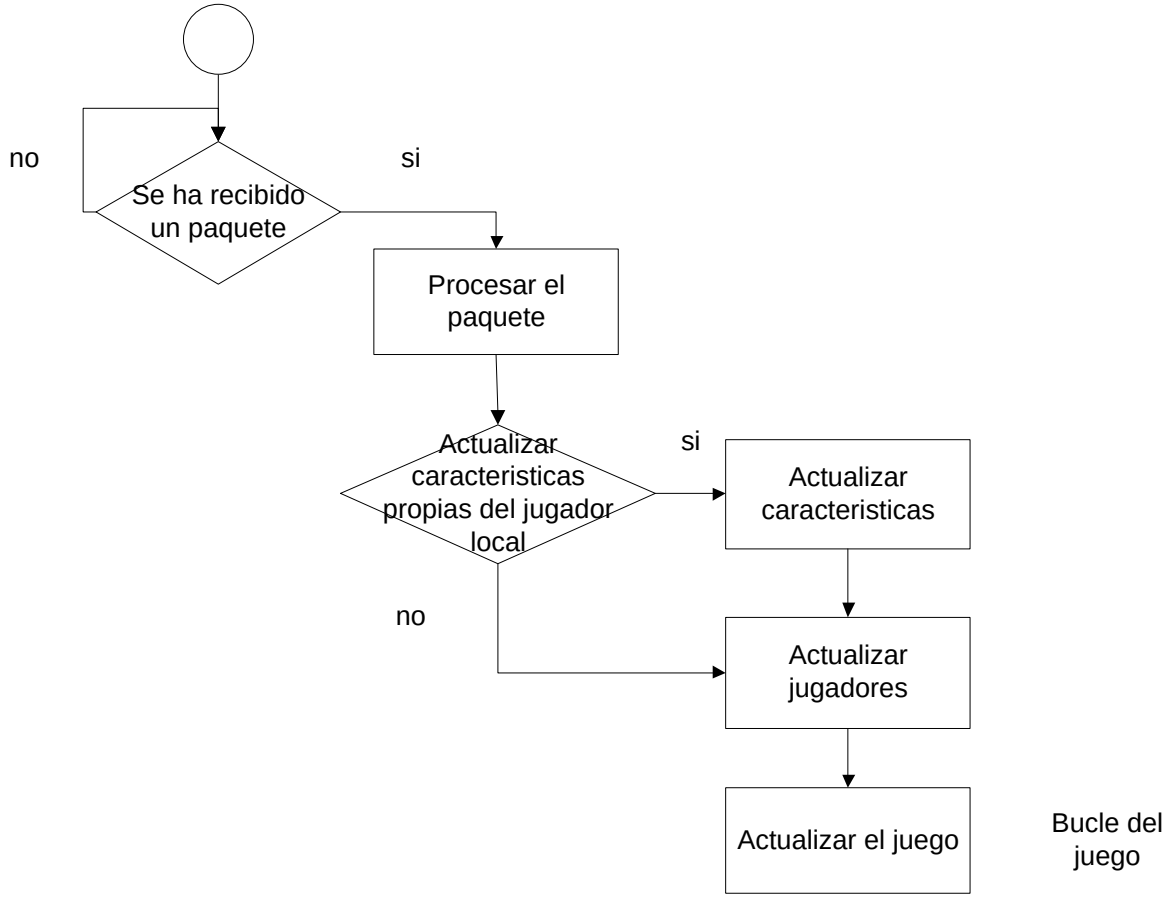
Figura 69. Diagrama de actualización del mundo del videojuego



Fuente: El autor

Este diagrama de envió de información a la red, solo es para la actualización del jugador local para que los demás clientes de la red sepan de su estado y demás características. Como se puede observar se utiliza un contador para saber en qué momento se deben enviar los paquetes, este contador depende del número máximo de jugador que se configura en el servidor del juego. El contador empieza en cero y si no es momento de enviar el paquete para la actualización del jugador local en la sesión de red de los demás jugadores, este se incrementar hasta alcanzar el criterio para que se cumpla la condición que como ya se indicó, depende del número máximo de jugadores.

Figura 70. Diagrama de actualización de un jugador



Fuente: El autor

En el gráfico anterior se notará que al recibir un paquete y según lo que se debería hacer con el paquete este es procesado, en este caso es un paquete de actualización de los jugadores de la sesión, si bien se pregunta si se debe actualizar las características propias del jugador local este igualmente actualiza los jugadores de la sesión esto es porque en este paso solo se actualiza un atributo propio del jugador mas no representa llamadas a métodos.

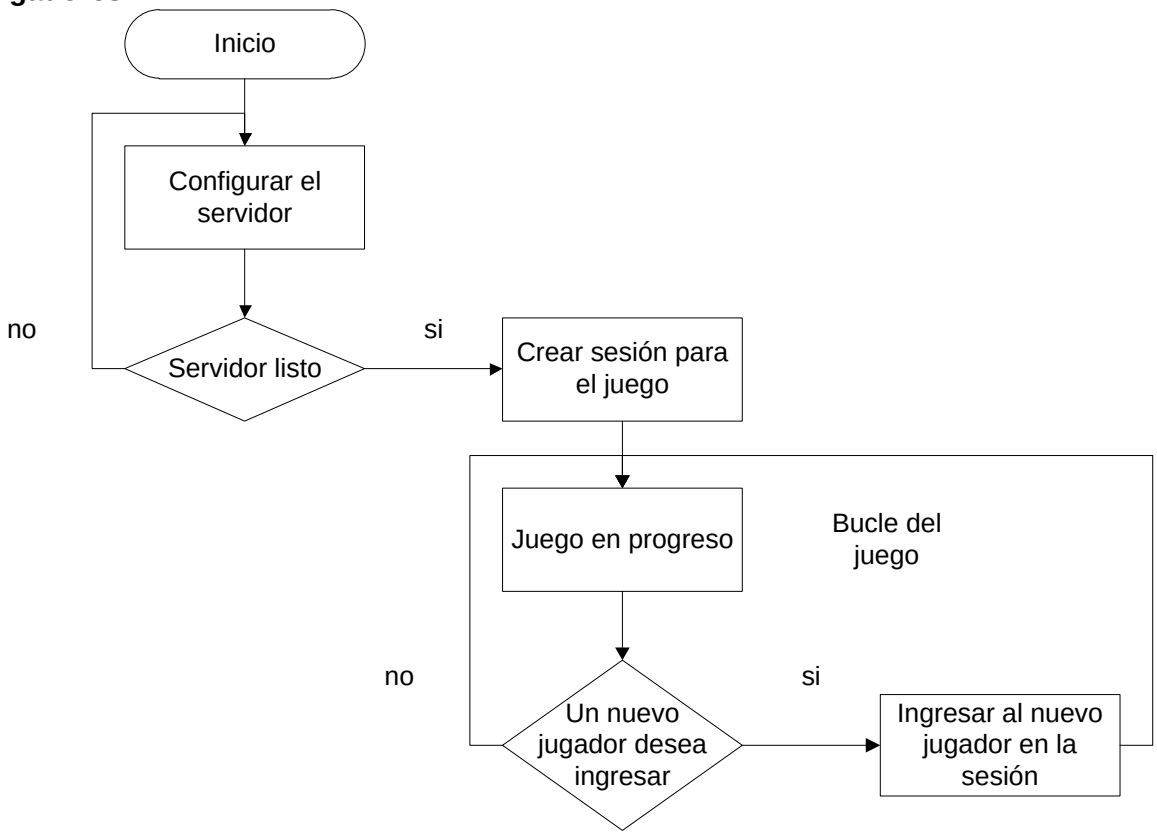
En el desarrollo del videojuego se tomó muy en cuenta que la información que fluiría a través de la red no debía inundarla, con esta consideración se obvio la incorporación de entidades físicas al mundo del videojuego, ya que si se lo hubiera hecho, el servidor sería el encargado de notificar a los clientes de la red el estado de estas entidades físicas y también considerar a qué momento se debía de enviar esas notificaciones, sin embargo aunque no se lo hizo se concluyó que para una red local él envió de paquetes de actualización del mundo del videojuego a los demás clientes de la sesión de red se lo debía hacer cada 30 frames desde la última actualización del videojuego., es decir comenzar con un contador encerado con el valor cero, en cada llamada al método Update incrementar este contador, una vez que se cumplió el criterio para él envió de datos, empaquetar dicha información para luego ser envidada.

Dentro de un videojuego en red existe mucha información fluyendo en la red y que los clientes deben procesar, en este paso cuando el cliente recibe la información y que debe

procesar se debe seleccionar que información actualizar para que no influya en la jugabilidad del jugador local, es este punto donde los paquetes recibidos no deben afectar al jugador local, es decir, la información que es recibida no debe modificar las características propias de un jugador en este caso el local.

En general el videojuego en red desarrollado se comporta de la siguiente manera, de acuerdo al mecanismo implementado:

Figura 71. Diagrama general según el mecanismo para el ingreso de nuevos jugadores.

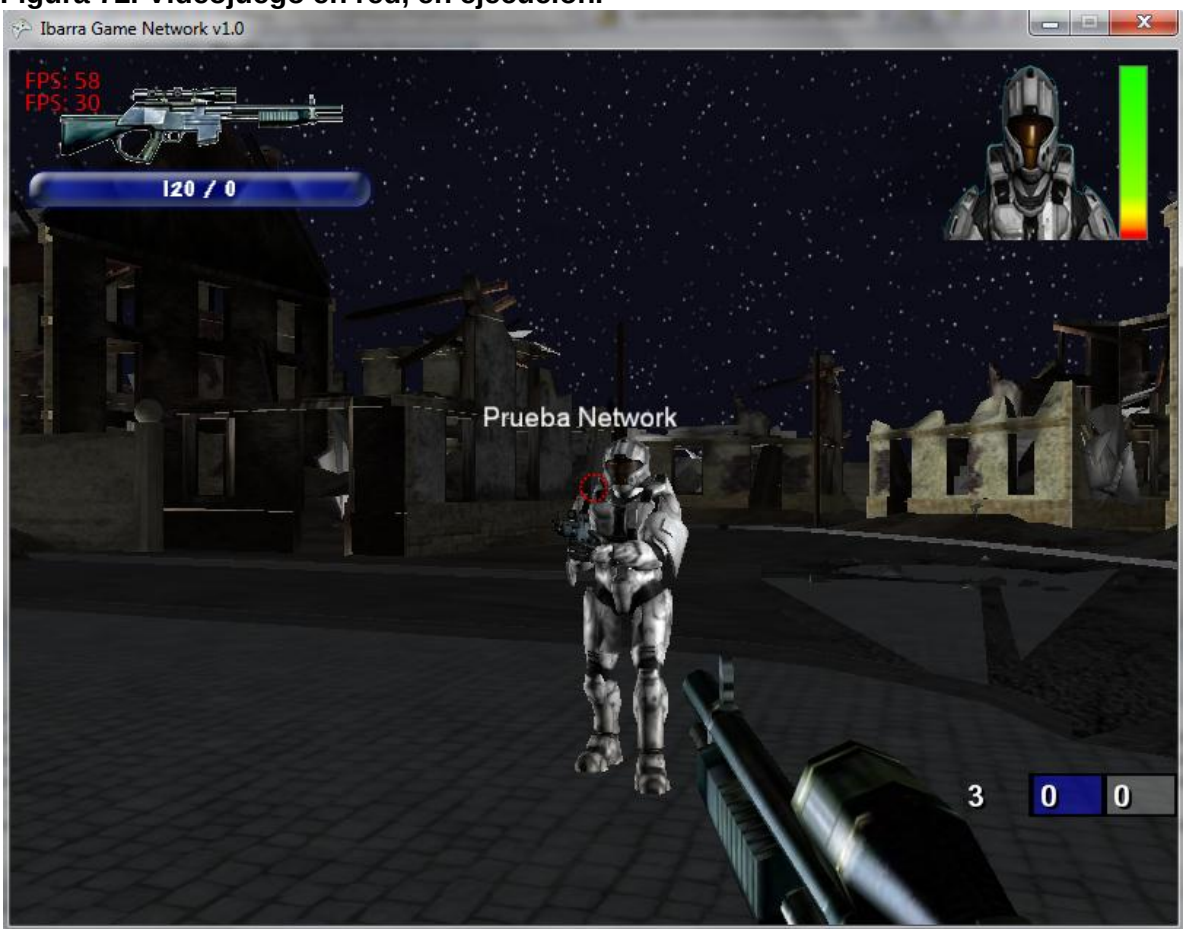


Fuente: El autor

Como ya se mencionó anteriormente el objeto más importante dentro de un videojuego es la cámara, con esto es fácil saber que el jugador local es la única autoridad para modificar el comportamiento de la cámara y de otras características propias del jugador local.

Para concluir con esta capitulo, es importante mencionar que el desarrollo de videojuegos en red puede ser de lo más complejo y quizás el resultado al final del desarrollo no sea el esperado, en este caso se cumplió en su totalidad todos los objetivos planteados antes de desarrollar el proyecto, sin embargo, esto se logró en primer lugar a la facilidad ofrecida por XNA para el soporte de red y por último a la amplia documentación encontrada en páginas oficiales de Microsoft para desarrolladores interesados en esta tecnología, junto con Starter Kits que pueden ser estudiados para ayudar a desarrollar proyectos propios ya que cuentan con su código fuente que puede ser reutilizado.

Figura 72. Videojuego en red, en ejecución.



Fuente: El autor

3.2.13 Detección de colisiones

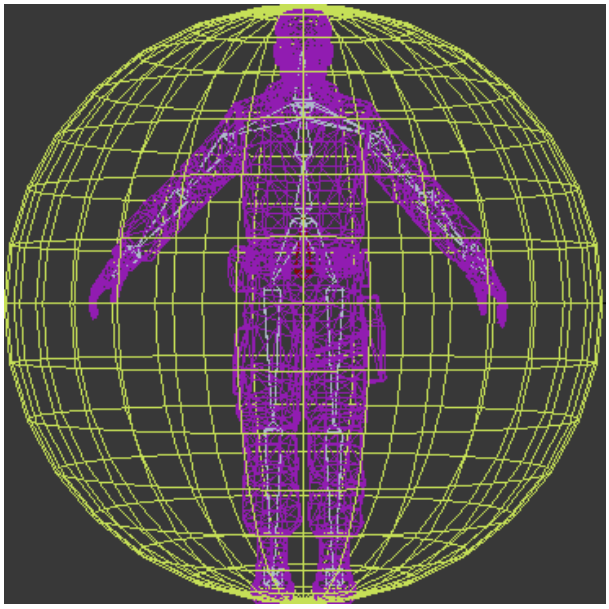
Dentro de la jugabilidad de los videojuegos desarrollados en este proyecto, y que según su complejidad puede ser de lo más fácil o lo más difícil de implementar, esto es la detección de colisiones.

Se puede comprobar las colisiones entre los objetos de la escena con algunos enfoques diferentes. Una manera es comprobar la intersección entre la totalidad de sus triángulos. Este método es el más preciso, pero es también el que más cálculos utiliza. Por ejemplo, para probar la colisión entre dos mallas de triángulos con 2.000 cada uno, que tendría que hacer 2.000 × 2.000 pruebas de choque. Esto es más de lo que generalmente se pueden permitir.

Los objetos que forman parte de escenario, si bien se mencionó anteriormente tienen mallas de colisión y que se integran al motor de física del motor de videojuegos implementado, esto no es suficiente para implementar colisiones entre los demás objetos del escenario, como por ejemplo, cuando el jugador está apuntando a un enemigo y este dispara, el enemigo debe recibir cierto daño dependiendo del arma utilizada, en este caso no se utiliza la malla de colisión del enemigo.

Los volúmenes para la detección de colisiones se los puede implementar de diferentes maneras, estas pueden ser con esferas o cubos que rodean al personaje, que al intersecar con otros objetos o elementos del juego, se produce la colisión. Sin embargo, cada una de estas presenta sus respectivas desventajas, si se utilizase una esfera para la detección de colisiones, la intercesión con otro elemento no sería de lo más óptimo, debido al volumen de esta figura.

Figura 73. Esfera de colisión sobre un modelo 3D



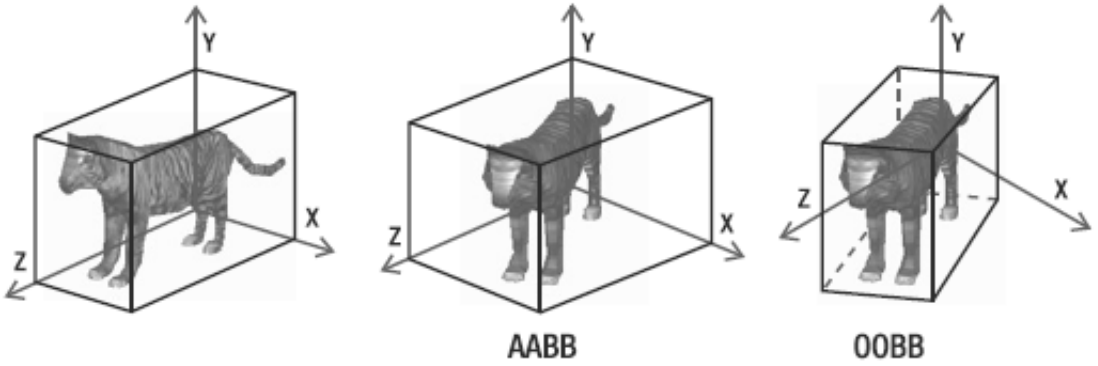
Fuente: El autor

Para la construcción de la esfera de colisión, se parte del modelo 3D, recorriendo los vértices de la malla del modelo, a partir de estos se crea una lista de vértices que sirven como parámetro para un método de la clase BoundingSphere, que retorna un volumen de colisión en este caso una esfera.

El otro camino y el que fue adoptado por parte del desarrollador, fue la construcción de cajas como volúmenes de colisión, pero con algunas variantes que se las muestra a continuación:

Se puede construir una caja que se utilizará para la colisión alineada con los ejes del mundo. En este caso, la caja es llamada axis-aligned bounding box(AABB). Una de las ventajas de la AABB es que la prueba de colisión es muy sencilla. Sin embargo, la AABB no puede girar, debido a que necesita guardar sus ejes alineados con los ejes del mundo. Si la caja reservada para la colisión se orienta con los ejes unidad, estos son llamados object-oriented bounding box(OOBB). Una prueba de colisión con un OOBB es más lento que uno con un AABB, pero el OOBB proporciona una caja que siempre tiene la misma orientación que la unidad.

Figura 74. Creación de un AABB y OOBB uno para una unidad con dos orientaciones diferentes.



Fuente: Libro Learning XNA 3.0

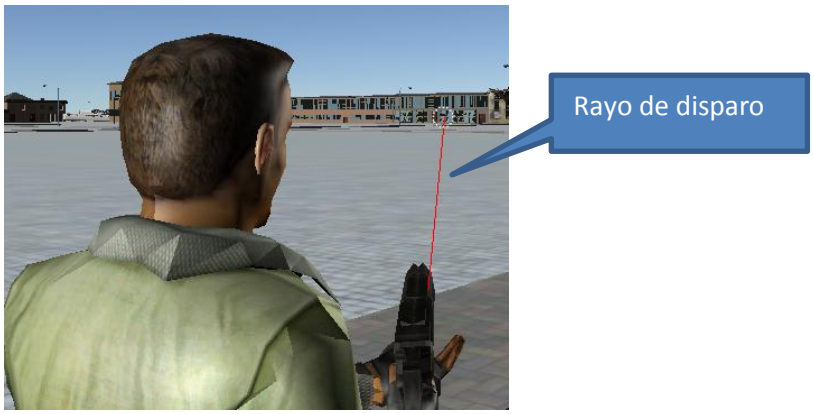
Cada uno de los modelos que intervienen en la jugabilidad del videojuego tiene una esfera y una caja de colisión, cada una de estas para un propósito diferente dentro del escenario.

Figura 75. Caja de colisión AABB para una modelo del videojuego.

Fuente: El autor

Dentro del género de los videojuegos a desarrollarse y que abarca de forma general la jugabilidad de estos es que se trata de juegos de disparos independientemente de la perspectiva del jugador, debido a esto la mayor parte de las pruebas de colisiones con el escenario del videojuego y el jugador, se lo realiza con un rayo que se inicia en el canon de disparo del arma hasta que interseque algún objeto, en caso de disparar, si el rayo interseca algún objeto que responda a esta acción, este ejecutara dicha acción, por ejemplo, si el jugador dispara a un enemigo y el rayo de disparo interseca la caja de colisión del enemigo, este recibirá algún daño dependiendo del arma que disparo.

Figura 76. Rayo de disparo.



Fuente: El autor

Se tiene que tomar en cuenta que aspectos del escenario son necesarios presentar al usuario del videojuego, como se puede observar, en el escenario se dibuja el rayo de disparo, sin embargo por cuestiones de estética y presentación se lo puede omitir, pero en este caso se lo mostrará al usuario ya que se lo considera un elemento del arma, además de ser útil para la depuración en tiempo de ejecución ya que se puede observar donde hace las pruebas de colisión con el escenario.

Al implementar los algoritmos de detección de colisiones con los objetos del escenario, es importante tener en cuenta el rendimiento en los cálculos utilizados para hacer ejecutar esta función, por ejemplo, si el escenario tiene un número mayor de 200 objetos que no intervienen en la jugabilidad del videojuego, pero que es necesario hacer las pruebas de colisión con estos objetos, los cálculos podrían demandar más consumo de memoria, dando como resultado un desempeño lento en la ejecución del videojuego. Para solucionar este inconveniente, y cabe mencionar que en el desarrollo de este proyecto si se dio, fue que las pruebas de colisión solo se debían hacer con los objetos que se estén dibujando en pantalla, eliminado lo que no esté dentro de este rango, con esto se logró que el rendimiento sea bueno dentro de los que considero tolerable.

Debido a que el proyecto se lo desarrollo en un computador normal (sin modificación de software ni hardware), el rendimiento del videojuego se vio limitado por esto, sin embargo, uno de los objetivos del proyecto fue que la aplicación pueda funcionar en la mayoría de computadores sin casi ninguna restricción, lo cual fue cumplido, un videojuego con un desempeño rápido está entre los 55 y 60 frames por segundo siendo lo óptimo 60 frames por segundo, en el proyecto desarrollado se obtuvo los siguientes resultados:

Proyecto	Desempeño
Videojuego para un solo jugador	45 – 55 fps
Videojuego multijugador	55 – 60 fps

Tabla de rendimiento del proyecto desarrollado

La PC donde se realizó el diseño y desarrollo tiene las siguientes características:

- Procesador Intel Core(TM)2 Duo CPU 2.0 GHz
- 4 GB RAM
- Windows 7 Ultimate 64 bits.
- Intel Graphics Media Accelerator 4500MHD

Las características anteriores no presentan mayormente restricciones ya que actualmente las computadoras ofrecen todas estas características.

Y como podrá verse en la tabla anterior el desempeño de los videojuegos es considerablemente buena y esto puede cambiar según donde la aplicación sea probada.

La aplicación desarrollada no está optimizada por completo en cuanto a rendimiento a nivel de código, es decir, cuando fue programada la aplicación no se consideró las posibles opciones para mejorar el rendimiento y aprovechar lo que se pudo haber mejorado en más prestaciones graficas por parte de la aplicación ya que estas consumen más recursos tanto de hardware como de software.

Para mejorar el rendimiento, como opción se podría utilizar de hilos de ejecución, para ejecutar tareas en paralelo, y no en cascada como la aplicación actualmente está concebida, opción que en un futuro se podría implementar, además de otras maneras de optimizar recursos sin perjudicar la calidad del producto.

CAPITULO IV.

DESARROLLO DEL SISTEMA

4.1 Documentos de diseño

Antes de iniciar con la descripción de la metodología de desarrollo de software es necesario incluir en esta sección un documento específico para este tipo proyectos, este es el documento de diseño para los videojuegos desarrollados en este caso Ibarra Game v1.0 e Ibarra Game Network v1.0, en estos documentos se explican de manera general los alcances de cada uno de estos.

Desarrollado con tecnología

Microsoft®
Your potential. Our passion.™

Framework

xna

Documento de diseño para:

Ibarra Game v1.0

The Local City Dead

Escrito por Wilmer Carrera

Versión # 1.0

martes, 22 de enero de 2013

Tabla de Contenidos

Diseño de la historia

En un principio el videojuego a desarrollarse y que fue presentado a modo de propuesta, presentaba demasiada complejidad no solo por el desarrollo de los personajes del videojuego sino también las características de jugabilidad que inicialmente se concibieron, por esta razón, en el transcurso del desarrollo del sistema, este adoptó otra historia pero sin cambiar los objetivos a obtener.

El presente proyecto tiene un cierto parecido con los videojuegos de la saga de Resident Evil desarrollado por la compañía CAPCOM. El jugador tiene que completar ciertos objetivos pertenecientes a cada escenario del videojuego, la meta de este tipo de videojuego es cumplir con las misiones que son indicadas para lograr la victoria del mismo.

El videojuego a desarrollarse tiene la siguiente trama:

El escenario donde se desenvolverá la historia será la ciudad de Ibarra, la ciudad que no tiene población, porque todos han huido de esta y los que se quedaron murieron debido a la aparición de criaturas malvadas que destruyeron y acabaron con la ciudad, estas criaturas aparecieron por experimentos secretos llevados a cabo por grupos terroristas radicados en la ciudad de Ibarra y que durante mucho tiempo no fueron descubiertos, pero cuando lo hicieron ya era demasiado tarde.

Estos experimentos realizados sobre restos humanos e incluso en otros animales dieron como resultado que los muertos revivan y que dichos animales cambien de forma y comportamiento, estas criaturas agresivas e incapaces de sentir no se detienen ante nada.

El Gobierno con ayuda internacional decide acabar con esta plaga desatada en esta ciudad y para evitar que se propague, la única solución es destruir la ciudad por completo.

Una persona (el jugador), y el único sobreviviente deberá escapar de la ciudad antes de sucumbir con esta, pero para logarlo deberá conseguir armas para defenderse y reparar el único medio encontrado para huir del destino de la ciudad.

Versión 1.0

Esta versión incluye algunas características que fueron pensadas para ofrecer más entretenimiento al jugador.

1. En las primeras tres misiones el jugador tendrá que recolectar las únicas tres armas que le servirá para completar los objetivos requeridos.
2. Se incorporó la posibilidad de tomar un vehículo para una mejor travesía a través del escenario.
3. Al iniciar el videojuego el jugador será situado en el parqueadero situado en la parte de atrás del Ilustre Municipio de Ibarra.
4. Las próximas misiones serán con el objeto de reunir piezas de un reparar un helicóptero que es el único medio que puede utilizar para escapar.

Versión 1.5

Esta es la primera versión donde se hace una mayor revisión acerca de la historia del videojuego.

Los cambios son los siguientes:

1. El diseño del mundo del juego tendrá que ser con un enfoque de destrucción, tétrico y que indique un lugar sombrío, misterioso, etc.
2. Los monstruos aparecerán de tal forma que no se indique al jugador de donde aparecen.
3. Algunos objetos que obstruyan las calles de la ciudad serán puestos estratégicamente par lograra una mayor percepción de la historia del videojuego.
4. El jugador para lograr sobrevivir de los ataques del que es objeto por parte de las criaturas del videojuego, deberá destruir a estos para luego incrementar sus puntos de vida.

Versión 2.0

Esta es la última versión presentada en el desarrollo de este documento, y presenta pequeños cambios que afectaran a la historia del videojuego.

Los cambios son los siguientes:

1. Además de incrementar los puntos de vida, el jugador también podrá incrementar las municiones del arma que esté usando.
2. Se incorpora la posibilidad de que el jugador pueda tomar el control de más de un vehículo que este situado en el escenario.

Características del juego

Filosofía

Punto filosófico #1

Este videojuego trata de hacer que el jugador aplique un poco de destreza y habilidad para completar los objetivos de cada misión, fundamentalmente entra en una competencia entre salvar su vida o incluirse dentro del destino de la ciudad.

Punto filosófico #2

El presente videojuego solo podrá ser utilizado en PC con un sistema operativo Windows que tenga previamente instalado el framework .NET 3.5 y XNA 3.1 framework, con la última versión de DirectX de preferencia, estas restricciones son propias de la aplicación a desarrollarse.

Punto filosófico #3

Además la razón de utilizar una ciudad real es también de promover turísticamente la misma, ya que el jugador cuando se desplace por el escenario podrá visualizar su ubicación, con el nombre de las calles que también serán reales.

Preguntas comunes

¿Qué es el juego?

Este juego será presentado como proyecto de fin carrera, presentada como tesis para la obtención del título de Ingeniero de Sistemas Computacionales.

¿Por qué crear este juego?

Este juego es creado para incursionar dentro de la industria del videojuego, no con un proyecto que entre en competencia con algún producto comercial, pero si para conocer las bases iniciales de cómo hacer software de entreteniendo de este tipo, y proponerlo como una nueva línea de desarrollo para los profesiones de software.

¿Dónde tendrá lugar el juego?

El juego tendrá como escenario el centro de la ciudad de Ibarra – Ecuador y las misiones utilizarán lugares conocidos de dicha localización.

¿Qué se puede controlar?

El jugador tomará el control sobre el personaje para cumplir con los objetivos de la misiones, además podrá controlar vehículos que serán ubicados dentro del escenario.

¿Cuántos caracteres se puede controlar?

Solo podrá controlar un solo personaje.

¿Cuál es el principal enfoque?

El juego pretende adentrar al jugador dentro de un mundo con enemigos y monstros.

¿Qué tiene de diferente?

Aunque tiene similitud con una saga conocida dentro del mercado de los videojuegos, este juego será el primero desarrollado en el centro de estudios donde será presentado, además se lo pretende distribuir de manera gratuita a nivel de ciudad y según su popularidad, se espera llegar a nivel de provincia y país.

Características provistas

Características Generales

Mundo equivalente con la realidad

Enemigos y monstruos con inteligencia artificial básica.

Gráficos 3D

Resolución totalmente configurable

Color 32-bit

Jugabilidad

Juego de acción en tercera persona

Sonido de acuerdo a las acciones del jugador

Efectos de partículas sobre algunos objetos.

Información del estado del jugador durante la ejecución del videojuego

Interfaz intuitiva y agradable.

Los objetivos serán presentados en texto 2D para indicar que debe hacer para cumplir la misión.

Los objetos que el jugador no pueda controlar tendrán inteligencia artificial básica.

El progreso de jugador se guardada automáticamente.

La partida podrá ser guardada para luego ser cargada desde donde se lo guardó por última vez.

La entrada de datos será a través del teclado.

El mundo del juego

Generalidades

El mundo del juego será representado por una ciudad.

Característica del mundo #1

La ciudad utilizada está compuesta por casas y como referente se tomo el mapa de la ciudad de Ibarra (real) que consta de parques, y demás ornamenta urbana que lógicamente debido a la complejidad del diseño se obviaron algunos detalles.

Característica del mundo #2

El espacio restante del mundo será ocupado para la ambientación del mismo como cielo, nubes, etc.

El mundo físico

Generalidades

El motor de física se encargara de la simulación de física de los objetos del videojuego.

Localizaciones clave

Las armas, vehículos y demás objetos que serán puestos en el escenario serán de acuerdo al diseño del mundo que como ya se mencionó será una ciudad.

Travesía

El jugador deberá moverse a través del mundo como bien le convenga, siempre y cuando cumpla con los objetivos del mismo.

Escala

La escala será la utilizada en el modelador 3D que se utilizó para el desarrollo del presente proyecto, la cual fue en metros.

Objetivos

Cada nivel tendrá su propio objetivo a cumplir para poder acceder al siguiente nivel.

Clima

El clima del videojuego será de acuerdo a cada nivel, cabe destacar que no tendrán efectos de lluvia o nieve, pero si se incorporara niebla.

Día y Noche

La transición de día y noche será en tiempo real, es decir, según la hora del sistema, este representara el día o noche.

Tiempo

Las misiones a completar por el jugador no presentan límite de tiempo.

Sistema de renderizado

Generalidades

La renderización del juego está a cargo del framework que se utilizó para el desarrollo del presente proyecto.

Renderizado 2D/3D

El renderizado 2D y 3D se utilizó shaders provistos por el framework.

Cámara

Generalidades

La cámara ofrecerá una visualización en tercera persona.

Detalles de la cámara #1

La cámara perseguirá al jugador.

Motor del videojuego

Generalidades

El motor de videojuegos fue desarrollado por el autor del proyecto.

Detalles del motor del videojuego #1

El motor de videojuego es básico por lo que no puede ser comparado con un motor profesional.

Detección de colisiones

Las colisiones serán provistas por los objetos que forman el escenario físico ya que estas al utilizar primitivas geométricas ayudan a la detección de colisiones, además también algunos de los objetos del juego e incluso el jugador tienen una esfera y caja de colisión que fue necesario en algunas operaciones.

Modelos de Iluminación

Generalidades

La iluminación es básica debido a lo básico del motor desarrollado.

Detalle de modelos de iluminación #1

Para la iluminación se utiliza el tiempo del videojuego, ya sea de día o de noche.

Detalle de modelos de iluminación #2

El jugador siempre tendrá iluminación a través de puntos de luz, especificado por la posición de la cámara.

El diseño del mundo

Generalidades

El diseño del mundo será en 3D.

Detalle del diseño del mundo #1

El mundo del videojuego tendrá efectos provistos por un sistema de partículas.

Detalle del diseño del mundo #2

En lo posible se tratará de asemejar el mundo virtual con la realidad.

Personajes del juego

Generalidades

Los personajes del videojuego a excepción del jugador serán de aspecto tétrico para representar de mejor manera el comportamiento de monstruos.

Creando un personaje

Los personajes fueren obtenido a través de internet y mediante el modelador 3D escogido para el proyecto se le introdujo características para representar un personaje del videojuego.

Enemigos y monstros

Los enemigos y monstros aparecerán de manera conveniente a lo largo del videojuego, el jugador con las armas que tiene y según el poder de daño de estas puede destruir a estos adversarios.

Interfaz de Usuario

Generalidades

A lo largo de la ejecución del videojuego, en la pantalla se mostrara información importante del estado del jugador como, puntos de vida, la cantidad de munición, ubicación, etc.

Detalle de la interfaz de usuario #1

La información presentada al usuario fácil de entender y asimilar.

Detalle de la interfaz de usuario #2

La representación de la interfaz de usuario será utilizando gráficos 2D y donde corresponda a través de texto legible y en español.

Armas

Generalidades

Las armas serán modernas y estarán ubicadas de manera estratégica dentro del mundo del videojuego.

Detalles de las armas #1

Cada una de las armas tendrá capacidad diferente con respecto a la munición, diferente poder de daño, cantidad máxima de balas y tendrán la capacidad de recargar en caso de tener un excedente de munición.

Detalles de las armas #2

Si no se cuenta con munición no se podrá utilizar el arma, para recargar munición el jugador deberá de eliminar un adversario para lograr este objetivo.

Música, puntaje y efectos de sonido

Generalidades

El videojuego consta de sonido y música pero no tiene efectos de sonido.

Introducción de sonido

Para la introducción de sonido en el videojuego se utilizó XACT que está dentro de las herramientas ofrecidas por XNA.

Sonido 3D

Aunque XNA si provee la reproducción de sonido 3D esto no podrá ser implementado en el videojuego a desarrollarse.

Diseño de sonido

El sonido depende de las acciones del jugador como caminar, disparar, morir, etc, también cada escenario tiene su propio audio que es reproducido en bucle.

Juego para un solo jugador

Generalidades

El jugador podrá controlar un solo personaje del videojuego y en caso de tomar un móvil del mundo el jugador también controlara este.

Detalles del videojuego para un solo jugador #1

La cámara del videojuego siempre perseguirá al jugador.

Historia

El jugador siendo el único sobreviviente debe reparar el único medio de transporte para salvarse antes de que la ciudad donde esta sea destruida y para conseguir esto debe en lo posible mantenerse con vida ya que la ciudad está llena de peligros.

Horas de juego

Esto depende de la habilidad del jugador, se considera que un jugador avanzado y que ha tenido experiencia con otros videojuegos similares podrá tener 2 horas de entretenimiento, en contraste con un jugador inexperto se considera por lo menos 2 días hasta completar los objetivos a perseguir.

Condiciones de victoria

El jugador debe completar con éxito todas las misiones del videojuego.

Renderizado de personajes

Generalidades

Cada personaje será implementado de acuerdo a la librería de animación utilizada por el motor de videojuegos.

Detalles del renderizado de personajes #1

Los personajes tendrán efectos visuales básicos, como puntos de luz, luz direccional, color difuso, etc.

Detalles del renderizado de personajes #2

Los efectos visuales aplicados a los personajes.

Documento de diseño para:

Ibarra Game Network v1.0

The Arena Network Dead

Escrito por Wilmer Carrera

Versión # 1.0

martes, 22 de enero de 2013

Tabla de Contenidos

Diseño de la historia

El presénte videojuego no presenta historia, pero es necesario especificar de qué trata el videojuego, cabe mencionar que este proyecto está basado en el videojuego comercial Quake III Arena desarrollado por ID Software en 1999.

El juego permitirá a los jugadores, cuyas computadoras están conectadas a una red o a Internet, disputar partidas entre sí, en tiempo real. Usa una estructura de arquitectura cliente-servidor que requiere que los clientes de todas las computadoras de los jugadores se conecten a un solo servidor. El objetivo en *Quake* es moverse a través de todo el campo de batalla eliminando (*fragueando*, del inglés *frag*) a los jugadores enemigos y anotándose puntos basándose en los objetivos del tipo de juego. Cuando los puntos de vida de un jugador llegan a cero, el avatar del jugador es eliminado (*fragueado*); luego el jugador reaparece en otro punto del mapa y sigue jugando con sus puntos de vida restaurados, pero sin las armas ni ítems que recolectó anteriormente.

Características del juego

Filosofía

Punto filosófico #1

Este videojuego trata de hacer que el jugador aplique un poco de destreza y habilidad para cumplir con los objetivos del videojuego, fundamentalmente entra en una competencia con otros jugadores de la sesión de red, aniquilando a otros jugadores o esperar ser destruido.

Punto filosófico #2

El presente videojuego solo podrá ser utilizado en PC con un sistema operativo Windows previamente con el framework .NET 3.5 y con XNA 3.1 framework, con la última versión de DirectX de preferencia, estas restricciones son propias de la aplicación a desarrollarse. Además debe tener algún dispositivo que le permita la comunicación con una red de computadores.

Punto filosófico #3

Además los escenarios para el videojuego tendrán lugares conocidos de la ciudad de Ibarra.

Preguntas comunes

¿Qué es el juego?

Este juego será presentado como proyecto de fin carrera, presentada como tesis para la obtención del título de Ingeniero de Sistemas Computacionales.

¿Por qué crear este juego?

Este juego es creado para incursionar dentro de la industria del videojuego, no con un proyecto que entre en competencia con algún producto comercial, pero si para conocer las bases iniciales de cómo hacer software de entreteniendo de este tipo, y proponerlo como una nueva línea de desarrollo para los profesiones de software.

¿Dónde tendrá lugar el juego?

El juego utilizara lugares conocidos de la ciudad e Ibarra como escenarios.

¿Qué se puede controlar?

El jugador tomara el control sobre el personaje que debe controlar para cumplir con los objetivos de la partida iniciada por el servidor.

¿Cuántos personajes se pueden controlar?

Solo podrá controlar un solo personaje que será el jugador.

¿Cuál es el principal enfoque?

El juego pretende adentrar a los jugadores ingresados en la sesión de red a una competencia, donde el más hábil será quien gane la partida.

¿Qué tiene de diferente?

Aunque tiene similitud con una saga conocida dentro del mercado de videojuegos, este juego será el primero desarrollado con soporte de red en el centro de estudios donde será presentado, además se lo pretende distribuir de manera gratuita a nivel de ciudad y según su popularidad, se espera llegar a nivel de provincia y país.

Características provistas

Características generales

Gráficos 3D

Resolución totalmente configurable

Color de 32-bit

Características multijugador

Hasta 16 conexiones dentro de una LAN y hasta 4 sobre internet.

Fácil configuración del servidor de juegos.

Fácil búsqueda de sesiones de red.

Capacidad de envió de menajes a través de un módulo de chat.

Capacidad de comunicación a través de voz.

Jugabilidad

Juego de acción en primera persona

Sonido de acuerdo a las acciones del jugador

Efectos de partículas sobre algunos objetos.

Información del estado del jugador durante la ejecución del videojuego

Interfaz intuitiva y agradable.

El mundo del juego

Generalidades

Los escenarios donde se disputaran las partidas serán lugares conocidos de la ciudad de Ibarra.

Característica del mundo #1

Cada escenario tendrá objetos de juego que serán ubicados de manera estratégica para la interacción con los jugadores.

Característica del mundo #2

Los escenarios tendrán efectos especiales propios.

El mundo físico.

Generalidades

El motor de física se encargara de la simulación de física de los objetos del videojuego.

Localizaciones clave

El mundo solo tendrá armas que el jugador puede tomar y que serán ubicados según el escenario.

Travesía

El jugador deberá moverse a través del mundo como bien le convenga, siempre y cuando cumpla con los objetivos del mismo.

Escala

La escala será la utilizada en el modelador 3D que se utilizó para el desarrollo del presente proyecto, la cual fue en metros.

Objetivos

El único objetivo será quien anote más puntos aniquilando a los demás jugadores.

Clima

El clima del videojuego será de acuerdo a cada escenario.

Día y Noche

La transición de día y noche será en tiempo real, es decir, según la hora del sistema, este representara el día o noche.

Tiempo

El tiempo no influirá para que alguno de los jugadores consiga la vitoria.

Sistema de renderizado

Generalidades

La renderización del juego está a cargo del framework que se utilizó para el desarrollo del presente proyecto.

Renderizado 2D/3D

El renderizado 2D y 3D se utilizó shaders provistos por el framework.

Cámara

Generalidades

La cámara ofrecerá una visualización en primera persona.

Detalles de la cámara #1

La cámara perseguirá al jugador.

Motor del videojuego

Generalidades

El motor de videojuegos fue desarrollado por el autor del proyecto

Detalles del motor del videojuego #1

El motor de videojuego es básico por lo que no puede ser comparado con un motor profesional.

Agua

A través de un shader solo el tercer escenario presenta agua como parte del ambiente de escena.

Detección de colisiones

Las colisiones serán provistas por los objetos que forman el escenario físico ya que estas al utilizar primitivas geométricas ayudan a la detección de colisiones, además también algunos de los objetos del juego e incluso el jugador tienen una esfera y caja de colisión que fue necesario en algunas operaciones.

Modelos de Iluminación

Generalidades

La iluminación es básica debido a lo básico del motor desarrollado.

Detalle de modelos de iluminación #1

Para la iluminación se utiliza el tiempo del videojuego, ya sea de día o de noche.

Detalle de modelos de iluminación #2

El jugador siempre tendrá iluminación a través de puntos de luz, especificado por la posición de la cámara.

El diseño del mundo

Generalidades

El diseño del mundo será en 3D.

Detalle del diseño del mundo #1

El mundo del videojuego tendrá efectos provistos por un sistema de partículas.

Detalle del diseño del mundo #2

En lo posible se tratara de asemejar el mundo virtual con la realidad.

Personajes del juego

Generalidades

Cada jugador tendrá un personaje que puede controlar.

Creando un personaje

Los personajes fueren obtenido a través de internet y mediante el modelador 3D escogido para el proyecto se le introdujo características para representar un personaje del videojuego.

Enemigos y monstros

Para esta versión en red, en el presente videojuego no se incorpora enemigos.

Interfaz de Usuario

Generalidades

A lo largo de la ejecución del videojuego, en la pantalla se mostrara información importante del estado del jugador como, puntos de vida, la cantidad de munición, de quien recibe daño, etc.

Detalle de la interfaz de usuario #1

Al presionar una tecla especifica se activa le entrada de mensajes para ser enviados a todos los jugadores de la red, esto representara un chat en la sala de juego.

Detalle de la interfaz de usuario #2

El servidor estará en capacidad de ingresar comandos programados que afectaran a los clientes, dichos comando podrán ser visualizados solo si la maquina donde se los escribe es el servidor.

Armas

Generalidades

Las armas serán modernas y estarán ubicadas de manera estratégica dentro del mundo del videojuego.

Detalles de las armas #1

Cada una de las armas tendrá capacidad diferente con respecto a la munición, diferente poder de daño, cantidad máxima de balas y tendrán la capacidad de recargar en caso de tener un excedente de munición.

Detalles de las armas #2

Las armas podrán ser recolectadas de acuerdo a cada escenario del videojuego.

Música, puntaje y efectos de sonido

Generalidades

El videojuego consta de sonido y música pero no tiene efectos de sonido.

Introducción de sonido

Para la introducción de sonido en el videojuego se utilizó XACT que está dentro de las herramientas ofrecidas por XNA.

Sonido 3D

Aunque XNA si provee la reproducción de sonido 3D esto no podrá ser implementado en el videojuego a desarrollarse.

Diseño de sonido

El sonido depende de las acciones del jugador como caminar, disparar, morir, recibir daño, etc. también cada escenario tiene su propio audio que es reproducido en bucle.

Juego un solo jugador

N/A

Juego Multijugador

Generalidades

Uno de los jugadores deberá crear la sesión de red y convertirse en el servidor de videojuegos, los demás usuarios que serán los clientes se unirán a esta sesión.

Máximo de jugadores

El videojuego estará en capacidad de aceptar un máximo de 16 conexiones por sesión creada.

Servidor

La topología del videojuego es Cliente/Servidor.

Personalización

Solo el servidor será quien configure los aspectos del videojuego y los clientes se regirán a esta, como el escenario, el puntaje máximo para alcanzar la victoria.

Internet

La conexión a internet se lo realizara a través de clientes VPN, debido a que el contar con un servidor dedicado seria muy costoso.

Sitios de juego

Habrá un único servidor ejecutando el videojuego con salida a internet, el protocolo utilizado será TCP/IP ya que es el protocolo con el cual trabaja XNA.

Persistencia

En una red de computadores los problemas siempre se presentan, pero XNA provee facilidad en el desarrollo de aplicaciones en red. La persistencia estará a cargo del framework, en el sentido de envío y recepción de datos pero el programador deberá analizar qué datos son los más importantes, que no deben perderse y como deben llegar a los clientes para que estos la utilicen.

Guardado y restauración del juego

N/A.

Renderizado de personajes

Generalidades

Cada personaje será implementado de acuerdo a la librería de animación utilizada por el motor de videojuegos.

Detalles del renderizado de personajes #1

Los personajes tendrán efectos visuales básicos, como puntos de luz, luz direccional, color difuso, etc.

Detalles del renderizado de personajes #2

Los efectos visuales aplicados a los personajes dependen del motor de videojuego.

4.2 Metodología de desarrollo de software

Existen varias metodologías de desarrollo de software, pero al inicio del proyecto y como mejor opción para cumplir los objetivos planteados se decidió elegir la metodología de desarrollo RUP, debido principalmente por la generación de varios artefactos durante todas las fases que involucra esta metodología, artefactos que ayudaran desarrollo de este tipo de software.

El **Proceso Racional Unificado** (*Rational Unified Process* en inglés, habitualmente resumido como **RUP**) es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización.

Principios de desarrollo

El RUP está basado en 6 principios clave que son los siguientes:

- **Adaptar el proceso**

El proceso deberá adaptarse a las necesidades del cliente ya que es muy importante interactuar con él. Las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

- **Equilibrar prioridades**

Los requisitos de los diversos participantes pueden ser diferentes, contradictorios o disputarse recursos limitados. *Debe encontrarse un equilibrio que satisfaga los deseos de todos*. Gracias a este equilibrio se podrán corregir desacuerdos que surjan en el futuro.

- **Demostrar valor iterativamente**

Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados

- **Colaboración entre equipos**

El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requisitos, desarrollo, evaluaciones, planes, resultados, etc.

- **Elevar el nivel de abstracción**

Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes o marcos de referencia (frameworks) por nombrar algunos. Esto evita que los ingenieros de software vayan directamente de los requisitos a la codificación de software a la medida del cliente, sin saber con certeza qué codificar para satisfacer de la

mejor manera los requisitos y sin comenzar desde un principio pensando en la reutilización del código.

Un alto nivel de abstracción también permite discusiones sobre diversos niveles y soluciones arquitectónicas. Éstas se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con el lenguaje UML.

- **Enfocarse en la calidad**

El control de calidad no debe realizarse al final de cada iteración, sino en **todos** los aspectos de la producción. El aseguramiento de la calidad forma parte del proceso de desarrollo y no de un grupo independiente.

Ciclo de vida

El ciclo de vida RUP es una implementación del desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una baseline (Línea Base) de la arquitectura.

Durante la fase de inicio las iteraciones hacen mayor énfasis en actividades de modelado del negocio y de requisitos.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la baseline de la arquitectura, abarcan más los flujos de trabajo de requisitos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la baseline de la arquitectura.

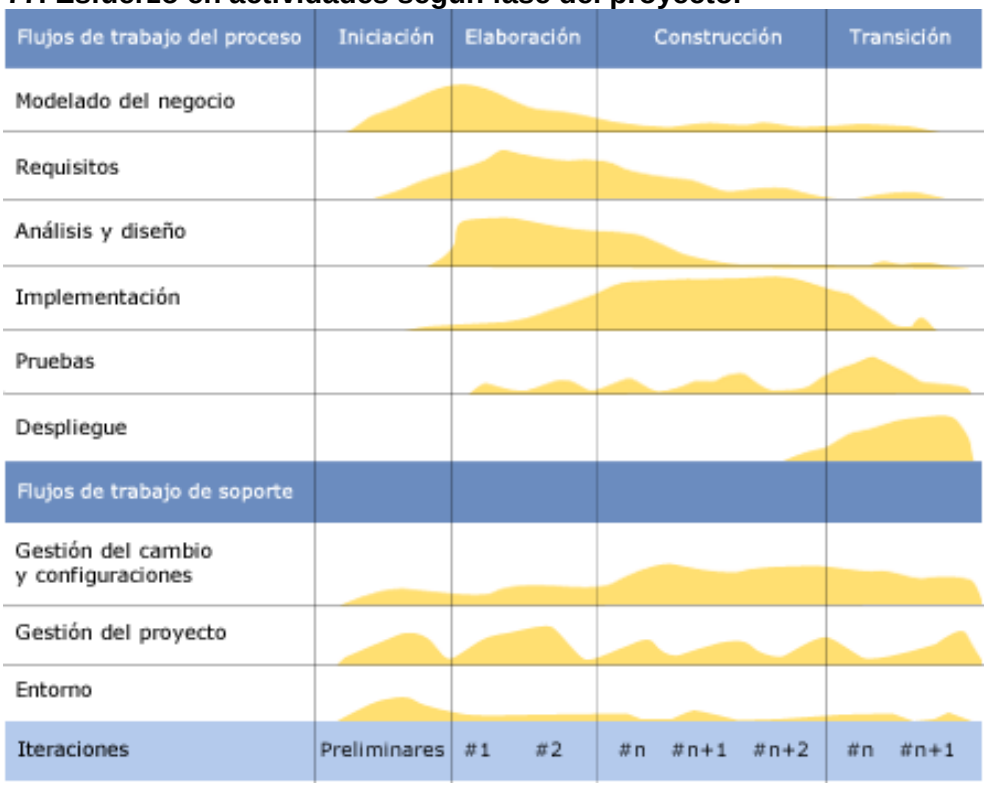
En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se selecciona algunos Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

Como se puede observar en cada fase participan todas las disciplinas, pero que dependiendo de la fase, el esfuerzo dedicado a una disciplina varía.

Figura 77. Esfuerzo en actividades según fase del proyecto.



Fuente: Sitio web oficial de RUP

Fases

Fase	Objetivos	Puntos de Control
Inicio	Definir el alcance del proyecto. Entender qué se va a construir.	Objetivo del proyecto.
Elaboración	Construir una versión ejecutable de la arquitectura de la aplicación. Entender cómo se va a construir	Arquitectura de la Aplicación.
Construcción	Completar el esqueleto de la Aplicación con la funcionalidad. Construir una versión Beta.	Versión Operativa Inicial de la Aplicación
Transición	Hacer disponible a la aplicación para los usuarios finales Construir la versión Final	Liberación de la versión de la Aplicación

La metodología RUP es más apropiada para proyectos grandes (Aunque también pequeños), dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en varias etapas. El presente proyecto fue desarrollado por una sola persona, obviamente, todas las etapas fueron desarrolladas por el autor del proyecto.

4.3 Desarrollo de software RUP

Los documentos presentados en esta sección son el resultado de seguir las fases de la metodología presentada por RUP.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Glosario

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
24/04/2011	1.0	Elaboración del glosario de los términos utilizados en el los artefactos generados para la metodología.	Wilmer Carrera

Tabla de Contenidos

Glosario

1. Introducción

Presenta cualquier información que tal vez el lector necesite entender en esta sección. Este documento es usado para definir terminología específica. Explica términos que tal vez no sean familiares para el lector como la descripción de los casos de uso u otros documentos. Frecuentemente este documento puede ser usado como un diccionario de datos informal.

1.1. Propósito

Definir términos utilizados en los documentos generados para el proyecto.

1.2. Ámbito

Este documento aplica a todos los artefactos elaborados.

2. Definiciones

- **2D** algo es bidimensional si tiene dos dimensiones, por ejemplo, ancho y largo, pero no profundidad. Los planos son bidimensionales, y sólo pueden contener cuerpos unidimensionales o bidimensionales.
- **3D e**n geometría y análisis matemático, un objeto o ente es tridimensional si tiene tres dimensiones. Es decir cada uno de sus puntos puede ser localizado especificando tres números dentro de un cierto rango.
- **Capcom Co., Ltd.** es una empresa japonesa desarrolladora y distribuidora de videojuegos. Fue fundada en 1979 como *Japan Capsule Computers*, una empresa dedicada a la fabricación y distribución de máquinas de videojuegos. Su actual nombre es el resultado de unir **Cap**sule **Com**puters.
- **Desarrollador de videojuegos** es un desarrollador de software (ya sea un individuo o una empresa) que crea videojuegos para diversas plataformas (videoconsola o computadora personal).
- **Direct3D** es parte de DirectX (conjunto de bibliotecas para multimedia), propiedad de Microsoft. Consiste en una API para la programación de gráficos 3D. Está disponible tanto en los sistemas Windows de 32 y 64 bits, como para sus consolas Xbox y Xbox 360.
- **Entretenimiento** es una diversión con la intención de fijar la atención de una audiencia o sus participantes. La industria que proporciona entretenimiento es llamada industria del entretenimiento.

- **Framework** es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de *software* concretos, con base en la cual otro proyecto de *software* puede ser organizado y desarrollado.

- **Industria de los videojuegos** es el sector económico involucrado en el desarrollo, la distribución, la mercadotecnia y la venta de videojuegos y del hardware asociado. Engloba a docenas de disciplinas de trabajo y emplea a miles de personas alrededor del mundo. La industria de videojuegos ha experimentado en los últimos años altas tasas de crecimiento, debido al desarrollo de la computación, capacidad de procesamiento, imágenes más reales y la estrecha relación entre películas de cine y los videojuegos, con lo cual los consumidores reconocen los títulos más pronto. En la década de 2000, los videojuegos han pasado a generar más dinero que la del cine y la música juntas, como en el caso de España. La industria de videojuegos generó 57.600 millones de euros durante 2009 en todo el mundo.

- **Inteligencia artificial** (**IA**) se denomina a la rama de las ciencias de la Computación dedicada al desarrollo de agentes racionales no vivos.

- **Inteligencia** es el término global mediante el cual se describe una propiedad de la mente en la que se relacionan habilidades tales como las capacidades del pensamiento abstracto, el entendimiento, la comunicación, el raciocinio, el aprendizaje, la planificación y la solución de problemas.

- **Konami Corporation** es una empresa de desarrollo de juguetes, cartas coleccionables, anime, tokusatsu, máquinas de monedas y videojuegos. Fue fundada en 1969 como un negocio de reparación de jukeboxes en Osaka, Japón, por Kagemasa Kozuki, quien es todavía su presidente y CEO. El nombre "Konami" es una conjunción de los nombres Kagemasa **Ko**zuki, Yoshinobu **Na**kama, Hiro **M**atsuda, y Shokichi **I**shihara, quienes fueron los socios de Kozuki y los fundadores originales de Konami Industry Co., Ltd en 1973. Konami también significa "olas pequeñas".

- **Lenguaje Unificado de Modelado** (LUM o **UML**, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

- **Microsoft Corporation** es una empresa multinacional de origen estadounidense, fundada el 4 de abril de 1975 por Bill Gates y Paul Allen. Dedicada al sector de la informática, con sede en Redmond, Washington, Estados Unidos. Microsoft desarrolla, fabrica, licencia y produce software y equipos electrónicos.

- **Microsoft XNA** (**X**NA's **N**ot **A**cronymed, XNA no es un acrónimo) es un conjunto de herramientas con un entorno de ejecución administrado proporcionado por Microsoft que facilita el desarrollo de juegos de ordenador y de gestión.

- **Motor de videojuego** es un término que hace referencia a una serie de rutinas de programación que permiten el diseño, la creación y la representación de un videojuego.

- **OpenGL** (**Open G**raphics **L**ibrary) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

- **Pixel**, plural **píxeles** (acrónimo del inglés *picture element*, "elemento de imagen") es la menor unidad homogénea en color que forma parte de una imagen digital, ya sea esta una fotografía, un fotograma de vídeo o un gráfico.

- **Resident Evil** (*Residente maligno*, traducido oficialmente como *El huésped maldito*), conocida como ***Biohazard*** (バイオハザード *Baiohazādo*?, Riesgo biológico) en Japón, es una serie de videojuegos y una franquicia de medios, entre los que se incluyen cómics, novelas, películas y coleccionables como figuras de acción, guías de estrategia y otras publicaciones.[1] Desarrollados por Capcom y creados por Shinji Mikami, se han vendido cerca de 40 millones de videojuegos de esta serie, a mayo de 2009.[2] Toda la saga, tanto juegos como en lo demás, tiene la misma trama: un peligroso virus se ha propagado por error o intencionalmente por la corporación Umbrella, éste al infectar a los humanos los transforma en zombies o infectados.

- **Shaders:** la tecnología shaders es cualquier unidad escrita en un lenguaje de sombreado que se puede compilar independientemente.

- **Tarjeta gráfica**, **tarjeta de vídeo**, **placa de vídeo**, **tarjeta aceleradora de gráficos** o **adaptador de pantalla**, es una tarjeta de expansión para una computadora u ordenador, encargada de procesar los datos provenientes de la CPU y transformarlos en información comprensible y representable en un dispositivo de salida, como un monitor o televisor. Las tarjetas gráficas más comunes son las disponibles para las computadoras compatibles con la IBM PC, debido a la enorme popularidad de éstas, pero otras arquitecturas también hacen uso de este tipo de dispositivos.

- **Test de Turing** (o **Prueba de Turing**) es una prueba propuesta por Alan Turing para demostrar la existencia de inteligencia en una máquina. Fue expuesto en 1950 en un artículo (*Computing machinery and intelligence*) para la revista *Mind*, y sigue siendo uno de los mejores métodos para los defensores de la Inteligencia Artificial. Se fundamenta en la hipótesis positivista de que, si una máquina se comporta en todos los aspectos como inteligente, entonces debe ser inteligente.

- **Unidad central de procesamiento** o **CPU** (por el acrónimo en inglés de *central processing unit*), o simplemente el **procesador** o **microprocesador**, es el

componente del computador y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos.

- **Unidad de procesamiento gráfico** o **GPU** (acrónimo del inglés *graphics processing unit*) es un procesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos y o aplicaciones 3D interactivas. De esta forma, mientras gran parte de lo relacionado con los gráficos se procesa en la GPU, la CPU puede dedicarse a otro tipo de cálculos (como la inteligencia artificial o los cálculos mecánicos en el caso de los videojuegos).

- **Videoconsola** es un sistema electrónico de entretenimiento para el hogar que ejecuta juegos electrónicos (videojuegos) que están contenidos en cartuchos, discos ópticos, discos magnéticos o tarjetas de memoria.

- **Videojuego** o **juego de vídeo** es un software creado para el entretenimiento en general y basado en la interacción entre una o varias personas y un aparato electrónico que ejecuta dicho videojuego.

- **XAudio**, debido a la integración de Xbox 360 y Microsoft Windows, Microsoft está alentando activamente a los desarrolladores para emigrar las nuevas aplicaciones hacia el equivalente de las APIs de audio de Xbox tales como XAudio y XACT. XAudio es una API exclusiva para Xbox diseñada para el procesamiento de señales digitales, sin embargo, XAudio 2 es una API de audio común multiplataforma (Windows y Xbox) de bajo nivel, propuesta como el reemplazo de DirectSound.

- **Xbox 360** es la segunda videoconsola de sobremesa producida por Microsoft, fue desarrollada en colaboración con IBM y ATI. Fue lanzada en Norteamérica, Japón, Europa y Australia entre 2005 y 2006. Su servicio Xbox Live permite a los jugadores competir vía online y descargar contenidos como juegos arcade, demos, trailers, programa de televisión y películas. La Xbox 360 es la sucesora directa de la Xbox, y compite actualmente contra la PlayStation 3 de Sony y la Wii de Nintendo como parte de las videoconsolas de séptima generación.

- **XInput** es una API para los controladores de juegos de la siguiente generación. Ha sido introducido con el lanzamiento de la videoconsola XBox 360. Tiene la ventaja de que es más sencillo de usar que DirectInput. Además, XInput es compatible con DirectX en sus versiones 9 y superiores.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Lista de Riesgos

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
17/11/2010	1.0	Identificación de riesgos más importantes en la fase inicial del proyecto	Wilmer Carrera

Lista de Riesgos

La calificación de los riesgos presentados a continuación es del 1 al 10

Ranking	Descripción del Riesgo e Impacto	Estrategia de reducción del riesgo
10	Complejidad en el desarrollo de videojuegos	Incrementar esfuerzos tratando de cumplir los objetivos planteados, más no comparar lo que se va a desarrollar con un producto comercial.
9	Falta de hardware capaz de soportar el diseño y programación de gráficos tridimensionales.	Utilizar hardware actual y común para ofrecer la mayor compatibilidad con otras máquinas donde se instalará la aplicación.
8	Los componentes de un videojuego son demasiados como para ser afrontados por una sola persona.	En lo posible, tratar de utilizar lo que ya está hecho, a través de internet e incluso de otros videojuegos existentes en el mercado.
8	Nulo conocimiento en el desarrollo de software de entretenimiento.	Auto capacitación a través de libros y de internet.
7	Debido a la complejidad de desarrollar este tipo de software, el proyecto no podría ser terminado en el tiempo establecido.	Los desfases son permisibles, siempre y cuando no sobrepase el tiempo límite para la presentación del proyecto.
7	Escasa información existente sobre cómo desarrollar videojuegos, utilizando el framework de Microsoft XNA.	Antes de iniciar el proyecto es necesario conocer el framework y haber realizado proyectos pequeños para ver de mejor manera el proyecto a desarrollarse.
6	El número de usuarios concurrentes en la versión en red del videojuego sobrepasen los límites funcionales determinados.	Especificar que el número máximo de conexiones que soporta un sistema de red utilizando este framework está limitado por el framework más no por el sistema.
6	Demasiado tráfico de información a través de la red, que puede dar como resultado, el colapso del sistema.	Priorizar que tipo de información debe enviarse a atreves de la red y que debe ser procesada por los destinatarios.
5	Problemas de latencia y pérdida de datos a través de la red.	Existen problemas propios de la red que son muy difíciles de tratar, pero en lo posible tratar de mitigar estos problemas.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Documento de Arquitectura de Software

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
08/03/2011	1.0	Elaboración del documento de Arquitectura de Software.	Wilmer Carrera

Tabla de Contenidos

Documento de Arquitectura de Software

1 Introducción

1.1 Propósito

El presente documento provee una comprensiva arquitectura general del sistema, usando un número de diferentes vistas de arquitectura a describir diferentes aspectos del sistema.

1.2 Alcance

Este documento se aplica al proyecto a desarrollarse como tesis para la obtención del título de Ingeniero de Sistemas Computacionales.

1.3 Definiciones, Siglas y Abreviaturas

Ver Glosario

1.4 Referencias

- Glosario

2 Representación de la Arquitectura.

El presente documento presenta la arquitectura como una serie de vistas, casos de uso, vista de procesos, vista de despliegue y vista de implementación. Los modelos han sido desarrollados usando una herramienta libre para el diseño de diagramas, y el lenguaje UML.

3 Objetivos y Restricciones de la Arquitectura.

Existen requerimientos y restricciones importantes para la definición de la arquitectura:

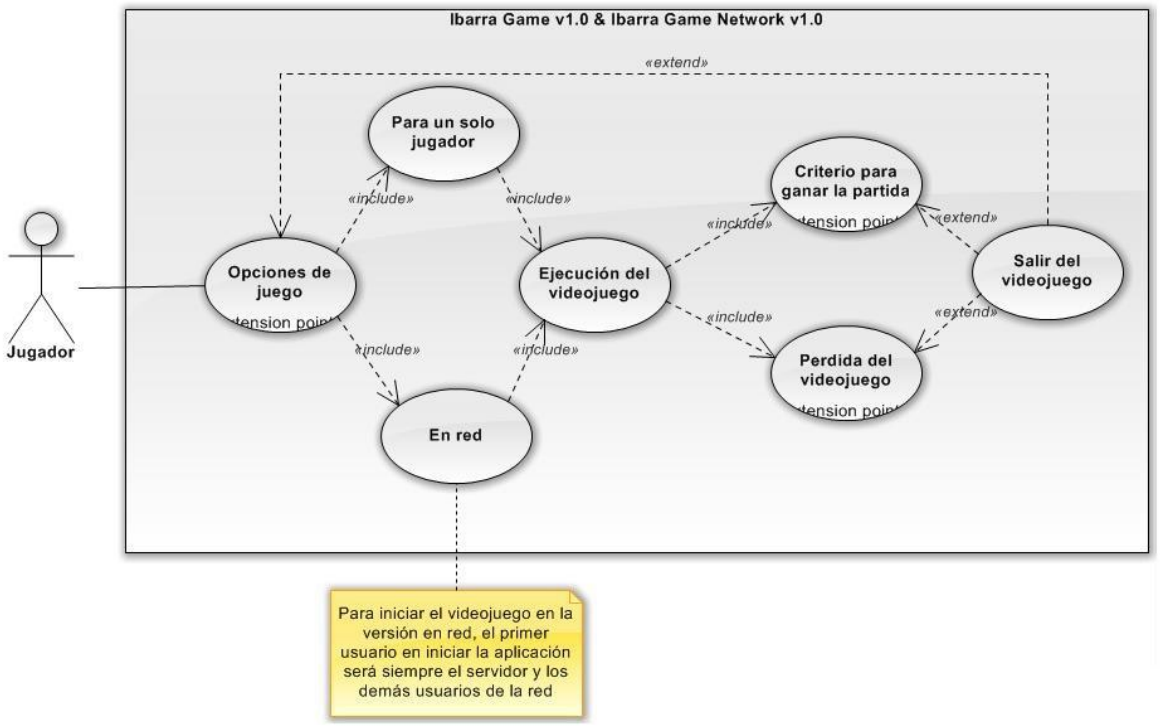
- Los computadores donde se instalará la aplicación deben tener instalado el framework .NET 3.5 y XNA 3.1 en su versión de redistribución provisto desde la página de descargas de Microsoft, para el correcto funcionamiento de la aplicación.

- La aplicación utiliza algunas fuentes que en caso de no estar instaladas en los computadores, estas serán instaladas automáticamente por la aplicación.

- Todos los requerimientos descritos en el documento de visión deben ser tomados en consideración para el desarrollo de la arquitectura definida.

4 Vista de Casos de Uso

4.1 Modelo de casos de uso



4.2 Prioridad de casos de uso

	Caso de uso	Prioridad para el negocio	Prioridad Técnica
1	Opciones de juego	Media	Alta
2	Para un solo jugador	Alta	Alta
3	En red	Alta	Alta
4	Ejecución del videojuego	Alta	Alta
5	Criterio para ganar la partida	Alta	Alta
6	Pérdida del videojuego	Alta	Alta
7	Salir del videojuego	Media	Media

4.3 Descripción de casos de uso más relevantes

4.3.1 Opciones de juego

Al inicio de la aplicación dependiendo de cuál sea: la versión para un solo jugador o para multijugador, se presentará opciones de configuración para la misma, cabe mencionar que solo las opciones para la versión multijugador están habilitadas, para la otra aplicación solo forman parte del diseño mas no representan ninguna funcionalidad.

4.3.2 Para un solo jugador

En la versión para un solo jugador no hay opciones para escoger, al iniciar la aplicación, esta iniciara la aplicación para un solo jugador.

4.3.3 Modo de ejecución en red

La aplicación iniciara las opciones previamente escogidas, si este es el servidor, creara el escenario y demás configuraciones para la partida y creara la sesión, para que los demás usuarios se unan al juego, caso contrario simplemente el usuario escogerá la sesión en caso de haber más de una y se unirá a esta.

4.3.4 Ejecución del videojuego

Realiza todo el proceso lógico para el desarrollo del videojuego dependiendo de la versión que se esté utilizando. Donde se desarrolla toda la lógica del producto, motivo por el cual, es la parte más importante del proyecto.

4.3.5 Criterio para ganar la partida

Dependiendo de la versión, existe un motivo por el cual el jugador gane el videojuego, en el caso de un solo jugador, será completar todos los objetivos del videojuego, en la versión multijugador, ganará cuando complete el puntaje máximo para la victoria.

4.3.6 Pérdida del videojuego

Independientemente de la versión del videojuego el jugador perderá cuando se acaben los puntos de vida que inicialmente serán 100.

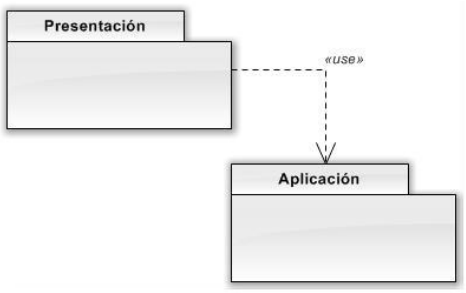
4.3.7 Salir del videojuego

En las opciones del videojuego existe la posibilidad de salir de la aplicación, y también como opción cuando la aplicación está en ejecución.

5 Vista Lógica

La vista lógica del proyecto a desarrollarse comprende 2 paquetes principales: Presentación y Aplicación

5.1 Paquetes arquitectónicos de diseño



5.1.1 Presentación

Los usuarios accederán a la aplicación a atreves de una interfaz con menús de selección de configuración para luego comenzar la aplicación.

5.1.2 Aplicación

Programa de escritorio donde se desplegara los objetos y entidades del mundo 3D contenidas en el videojuego.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Documento de Visión

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
17/11/2010	1.0	Creación del documento de visión	Wilmer Carrera

Tabla de Contenidos

Documento de Visión

1 Introducción

1.1 Propósito

El propósito de este documento es analizar y definir a alto nivel las necesidades y características de la aplicación a desarrollarse previa la obtención del título con el tema "Desarrollo de un videojuego con texturas y modelos 3D, multiusuario, en un entorno cliente/servidor (internet) utilizando el framework de Microsoft XNA".

1.2 Alcance

El presente documento de visión se aplica a los dos proyectos a realizarse derivados del tema citado anteriormente, cabe resaltar que son proyectos diferentes pero pertenecen a la misma aplicación.

1.3 Definiciones, Siglas y Abreviaturas

Ver Glosario

1.4 Referencias

- Glosario
- Resumen de los requerimientos de los interesados.
- Resumen del modelo de Casos de Uso.

2 Posicionamiento

2.1 Oportunidad del negocio

El presente proyecto de fin de carrera servirá para la obtención del título de Ingeniero de Sistemas, tesis de investigación sobre el desarrollo de videojuegos utilizando el framework de Microsoft XNA.

2.2 Definición del problema

El problema de	Escaso desarrollo de software de entretenimiento en nuestro país y no contar con centros de estudio especializado en esta actividad.
Que afecta a	Nadie en particular
El impacto de ello es	El mercado nacional de consumidores de software de entretenimiento está copado por productos internacionales.
Una solución exitosa debería	Incursionar en esta actividad que es una de las actividades económicas más rentables a nivel mundial.

2.3 Definición de la Posición del Producto

Para	Cualquier persona
Quien	Desee un producto de entretenimiento.
Los productos Ibarra Game v1.0 & Ibarra Game Network v1.0	Son productos de entreteniendo implementado sobre un computador.
Que	Será original ya que estará basado en escenarios de la ciudad de Ibarra.
Diferente de	Videojuegos desarrollados por empresas internacionales.
Nuestro producto	Es nacional y será el comienzo para incursionar en esta actividad.

3 Descripción de los interesados y usuarios

3.1 Resumen de los interesados

Nombre	Descripción	Responsabilidades
Director de tesis	Responsable en la revisión del proyecto a desarrollarse	Monitorear el avance del proyecto. En lo posible prestar ayuda a la culminación del proyecto.
Responsable del proyecto	Responsable del proyecto por parte del	Responsable de la investigación para culminar el proyecto con los

	autor del tema de tesis.	objetivos planteados. Responsable del análisis, diseño y desarrollo del proyecto. Responsable del correcto funcionamiento del proyecto. Reportar cualquier inconveniente que surja durante el desarrollo del proyecto al respectivo director de tesis.

3.2 Resumen de los usuarios

Nombre	Descripción	Responsabilidades
Usuario del sistema	Persona quien desee probar la aplicación.	En caso de la aplicación en red la única responsabilidad es por parte de quien inicie el servidor de juegos.

3.3 Entorno de usuario.

- Cualquier persona que pruebe la aplicación será un usuario del sistema, con esto se pretende tener una idea de cómo será acogida la creación de más sistemas de este tipo en la provincia y según los resultados en un futuro, a nivel nacional.
- En la versión en red del videojuego un usuario será el responsable de iniciar el juego con las configuraciones respectivas, y los demás usuarios se unirán a la respectiva sesión.
- La configuración del servidor de juegos está dada por las siguientes actividades:
    - Escoger el personaje que representará el jugador (el servidor también representa un jugador).
    - Configurar la sesión de red antes de ser creada.
    - Escoger el escenario de juego y el límite de frags (muertes) para ganar la partida.
    - Iniciar el servidor de juegos.
- El servidor tiene opciones para manipulas a los clientes como:
    - Ejecutar comandos programados en la aplicación.
    - Apagar la máquina de un cliente registrado en sesión.

- Reiniciar la máquina de un cliente registrado en sesión.
- Hibernar la máquina de un cliente registrado en sesión.
- Enrutar la información producida por la sesión de red a todos los clientes registrados.

Perfiles de los Interesados

3.3.1 Director de tesis

Representante	Mauricio Rea
Descripción	Responsable en la revisión del proyecto a desarrollarse
Tipo	Director
Responsabilidades	Monitorear el avance del proyecto. En lo posible prestar ayuda a la culminación del proyecto.
Criterio de éxito	Proporcionar el apoyo necesario en el desarrollo del proyecto.
Implicación	Revisor del desarrollo.
Entregable	N/A
Comentarios	Mantener una relación constante con el desarrollo del proyecto.

3.3.2 Responsable del proyecto

Representante	Wilmer Carrera
Descripción	Responsable del proyecto por parte del autor del tema de tesis.
Tipo	Estudiante
Responsabilidades	Responsable de la investigación para culminar el proyecto con los objetivos planteados. Responsable del análisis, diseño y desarrollo del proyecto. Responsable del correcto funcionamiento del proyecto. Reportar cualquier inconveniente que surja durante el desarrollo del proyecto al respectivo director de tesis.
Criterio de éxito	Terminar el proyecto con los objetivos planteados.

Implicación	Jefe de proyecto
Entregable	N/A
Comentarios	

3.4 Perfiles de usuario

3.4.1 Usuario del sistema

Representante	N/A
Descripción	Persona quien desee probar la aplicación.
Tipo	N/A
Responsabilidades	En caso de la aplicación en red la única responsabilidad es por parte de quien inicie el servidor de juegos.
Criterio de éxito	N/A
Implicación	N/A
Entregables	N/A
Comentarios	

3.5 Necesidades de los interesados y usuarios

Las presentes necesidades, solo son por parte de los interesados ya que el presente proyecto a desarrollarse al no ser un producto de tipo necesario para las personas, estas no presentan necesidades.

Necesidades	Prioridad	Inquietudes	Solución actual	Solución propuesta
Establecer una nueva opción para los desarrolladores de software.	Alta	La falta de motivación de afrontar un reto.	Escaso desarrollo de software de entretenimiento.	A partir de la investigación a desarrollarse, establecer un precedente de cómo desarrollar videojuegos.
Incursionar en una de las actividades más rentables a nivel mundial.	Alta	La dificultad de competir con empresas internacional	El mercado nacional está lleno de productos	Empezar a producir este tipo de software y según su aceptación,

		es.	internacionales.	seguir creciendo.
Elaborar un videojuego que sea del agrado de quien lo utilice.	Alta	El mercado de consumidores es muy exigente.	N/A	Desarrollar una aplicación que aunque no pueda competir con otra aplicación del mismo tipo hecha por empresas dedicadas a esta actividad, sea intuitiva y agradable.

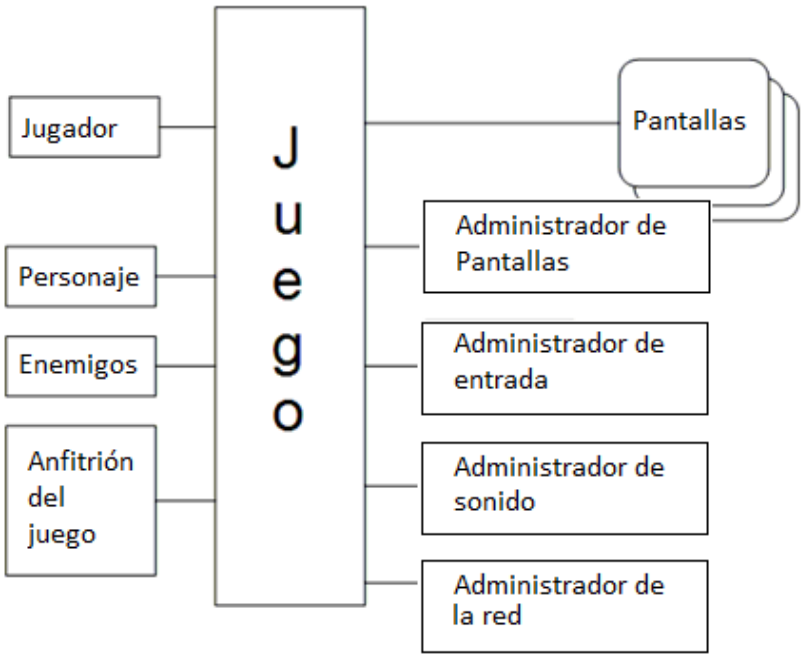
3.6 Alternativas y competencia

N/A

4 Vista General del Producto

Esta sección provee a alto nivel de las capacidades, interfaces con otras aplicaciones y configuraciones del sistema.

4.1 Perspectiva del producto.



4.2 Resumen de capacidades

Beneficios para el usuario	Características que lo soportan
Producto de entretenimiento con escenarios locales.	Ibarra será recorrido virtualmente, además de las características que los videojuegos ofrecen.
Cualquier cambio en los modelos del videojuego, pueden ser implementados.	Se conoce de todo el proceso de desarrollo, motivo por el cual cualquier cambio podrá ser implementado rápidamente.
Fácil instalación	El asistente de instalación liberara al usuario de tareas complejas.
Requerimientos de hardware	La aplicación podrá ser implementada solo sobre PC's con tarjetas de video para soporte de pixel y vertex shader 3.0, sin problemas para las PC's actuales.

4.3 Suposiciones y dependencias.

N/A

4.4 Costo y precio

La aplicación a desarrollarse no tendrá costo, pero será distribuida a través de CD el cual tendrá su respectivo costo.

4.5 Licenciamiento e instalación.

La instalación será a través de un instalador con un asistente que le guiara a través del proceso de instalación.

5 Características del producto.

5.1 Facilidad de uso

La aplicación será de escritorio y con compatibilidad probada para asegurar su correcto funcionamiento donde se pruebe la aplicación.

5.2 Uso Libre

El sistema desarrollado no tendrá claves ni algún otro tipo de restricción para su utilización.

5.3 Bajo costo

El costo esta dado solo por el medio de difusión de la aplicación en este caso a través de un CD

6 Restricciones

- El computador donde puede ser probado la aplicación debe tener por lo menos soporte de vertex y pixel shader 3.0 esto definido por la tarjeta de video instalada en el equipo.
- En la versión de red de la aplicación a desarrollarse solo se admite hasta 16 jugadores conectados simultáneamente.

7 Rangos de calidad

El desarrollo del presente sistema se elaborara siguiendo la Metodología de Desarrollo RUP, contemplando los parámetros de calidad que la metodología define.

8 Precedencia y prioridad

N/A

9 Otros requerimientos del producto.

Para la implementación del servidor de juegos y demás jugadores de la aplicación desarrollada, para la red se requiere de un switch que según los puertos del mismo los jugadores se limitaran a este.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de casos de uso de negocio:

Opciones de juego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
08/03/2011	1.0	Elaboración del documento de casos de uso de negocio	Wilmer Carrera

Tabla de Contenidos

Especificación de casos de uso de negocio:

Opciones de juego

1 Introducción

1.1 Propósito

El principal propósito de los casos de uso de negocio es proporcionar información sobre los procesos que se desarrollan en la selección de las opciones de juego propias de la aplicación.

1.2 Alcance

Este documento se aplica al proyecto “Ibarra Game v1.0 & Ibarra Game Network v1.0” que será desarrollado como tesis para la obtención del título de Ingeniero en Sistemas Computacionales de la Universidad Técnica del Norte.

1.3 Definiciones, Siglas y Abreviaciones

Ver glosario

1.4 Referencias

Las referencias aplicables son:

- Documento de visión v1.0
- Lista de riesgos v1.0
- Glosario 1.0
- Metodología RUP, 1999, Rational Software Corp.

1.5 Descripción

El documento de casos de uso de negocio contiene la siguiente información:

Descripción del caso de uso de negocio que provee una descripción del caso de uso de negocio, sus objetivos, metas y flujo de trabajo.

2 Opciones de juego

2.1 Descripción

El caso de uso de negocio: Opciones de juego corresponde a las diferentes opciones que pueden ser configuradas por el jugador antes de empezar con la lógica del videojuego. En la versión de un solo jugador estas opciones no influyen en la configuración de las características del videojuego, pero ayuda en el diseño de la aplicación, en la versión multijugador estas opciones son totalmente funcionales tanto para el servidor, como para un usuario de la aplicación.

Desde el inicio de la aplicación el administrador de pantallas empieza a renderizar las respectivas pantallas, según el flujo lógico del videojuego.

3 Metas

Proporcionar una interfaz agradable al usuario, tomando como molde para este objetivo, videojuegos comerciales, el usuario podrá seleccionar las opciones que más le convenga antes de iniciar el juego y utilizar el administrador de pantallas para su correcto funcionamiento.

4 Metas funcionales

Proporcionar opciones de configuración de la aplicación a través de interfaces gráficas, que sean de agrado del usuario.

5 Flujo

5.1 Flujo básico

5.1.1 Inicio del administrador de pantallas

Al inicio de la aplicación el primer componente en activarse es el administrador de pantallas, que según el proceso lógico de la ejecución de la aplicación será el responsable de mostrar la pantalla respectiva.

5.1.2 Presentación de las opciones de juego

Al iniciarse el administrador de pantallas y según lo programado, lo primero que se verá son las opciones de juego, y esta representa la primera pantalla donde el jugador interactúa por primera vez con la aplicación.

5.1.3 Configuración de la aplicación

En la pantalla de opciones el usuario podrá configurar varios aspectos del videojuego.

5.2 Flujo alternativo

5.2.1 Ingresar directamente al desarrollo del juego

Una vez iniciada la aplicación es posible ingresar directamente al videojuego, pero como diseño y estética de este tipo de productos, esto no es lo más recomendable.

6 Categoría

Este caso de uso de negocio forma parte del núcleo del proyecto

7 Riesgo

Ver lista de riesgos.

8 Propietario del proceso

El propietario es el Administrador de pantallas, componente que se inicia al ejecutar la aplicación y que forma parte del motor del videojuego desarrollado.

9 Precondiciones

Cumplir con los requerimientos de software y hardware para ingresar a la aplicación.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de casos de uso de negocio:

Para un solo jugador

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
09/03/2011	1.0	Elaboración del documento de casos de uso de negocio.	Wilmer Carrera

Tabla de Contenidos

Especificación de casos de uso:

Para un solo jugador

1 Introducción

1.1 Propósito

El principal propósito de los casos de uso de negocio es proporcionar información sobre los procesos que se desarrollan para iniciar la aplicación en la versión de un solo jugador. Esta información es utilizada para elaborar un plan de desarrollo para el proyecto.

1.2 Alcance

Este documento se aplica al proyecto “Ibarra Game v1.0 & Ibarra Game Network v1.0” que será desarrollado como tesis para la obtención del título de Ingeniero en Sistemas Computacionales de la Universidad Técnica del Norte.

1.3 Definiciones, Siglas y Abreviaciones

Ver glosario

1.4 Referencias

Las referencias aplicables son:

- Documento de visión v1.0
- Lista de riesgos v1.0
- Glosario 1.0
- Metodología RUP, 1999, Rational Software Corp.

1.5 Descripción

El documento de casos de uso de negocio contiene la siguiente información:

Descripción del caso de uso de negocio que provee una descripción del caso de uso de negocio, sus objetivos, metas y flujo de trabajo.

2 Para un solo jugador

2.1 Descripción

El caso de uso de negocio: Para un solo jugador corresponde a una de las versiones soportadas por la aplicación, aunque todas las versiones forman parte del proyecto no se puede ejecutar la aplicación más de una vez, se entiendo con esto que si la aplicación se la ejecuta para un solo jugador, este no podrá iniciar la misma aplicación en su versión multijugador y viceversa.

Antes de iniciar la aplicación y luego de ingresar a la pantalla inicial del videojuego, el jugador deberá seleccionar las opciones que más le convenga de la pantalla de opciones, aunque en la versión para un solo jugador estas opciones no influyen en la jugabilidad del videojuego o parámetros de configuración de la aplicación, cumplen un papel de estética para este tipo de sistemas

3 Metas

Inducir al jugador al interés por probar la aplicación, a través de una interfaz agradable.

4 Metas funcionales

Implementar casi el mismo flujo de productos comerciales de este tipo al proyecto desarrollado.

5 Flujo

5.1 Flujo básico

5.1.1 Selección de las opciones de juego

La primera pantalla de la aplicación es la selecciones de opciones que dependiendo de la versión de la aplicación el usuario podrá seleccionar las que más le convenga.

5.1.2 Interacción usuario – aplicación

En esta pantalla es donde por primera vez el usuario, tiene que ingresar datos a través del teclado en la aplicación, en este caso donde utilizara el teclado para la selección de las opciones que desea.

5.2 Flujo alternativo

5.2.1 Ingresar directamente al desarrollo del juego

Una vez iniciada la aplicación es posible ingresar directamente al videojuego, pero como diseño y estética de este tipo de productos, esto no es lo más recomendable.

6 Categoría

Este caso de uso de negocio forma parte del núcleo del proyecto, específicamente forma parte del administrador de pantallas, y como interactúa directamente con el usuario también integra el administrador de entrada de datos.

7 Riesgo

Ver lista de riesgos.

8 Propietario del proceso

El propietario es el administrador de pantallas y el administrador de entrada de datos, componentes que se inician al ejecutar la aplicación y que forma parte del motor del videojuego desarrollado.

9 Precondiciones

Cumplir con los requerimientos de software y hardware para ingresar a la aplicación.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de casos de uso de negocio:

Modo de ejecución en red

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
09/03/2011	1.0	Elaboración del documento de casos de uso de negocio	Wilmer Carrera

Tabla de Contenidos

Especificación de casos de uso de negocio:

Modo de ejecución en red

1 Introducción

1.1 Propósito

El principal propósito de los casos de uso de negocio es proporcionar información sobre los procesos que se desarrollan para iniciar la aplicación en la versión con soporte de red. Esta información es utilizada para elaborar un plan de desarrollo para el proyecto.

1.2 Alcance

Este documento se aplica al proyecto “Ibarra Game v1.0 & Ibarra Game Network v1.0” que será desarrollado como tesis para la obtención del título de Ingeniero en Sistemas Computacionales de la Universidad Técnica del Norte.

1.3 Definiciones, Siglas y Abreviaciones

Ver glosario

1.4 Referencias

Las referencias aplicables son:

- Documento de visión v1.0
- Lista de riesgos v1.0
- Glosario 1.0
- Metodología RUP, 1999, Rational Software Corp.

1.5 Descripción

El documento de casos de uso de negocio contiene la siguiente información:

Descripción del caso de uso de negocio que provee una descripción del caso de uso de negocio, sus objetivos, metas y flujo de trabajo.

2 Modo de ejecución en red

2.1 Descripción

El caso de uso de negocio: En red corresponde a una de las versiones soportadas por la aplicación, aunque todas las versiones forman parte del proyecto no se puede ejecutar la aplicación más de una vez, se entiendo con esto que si la aplicación se la ejecuta para un solo jugador, este no podrá iniciar la misma aplicación en su versión multijugador y viceversa.

Antes de iniciar la aplicación y luego de ingresar a la pantalla inicial del videojuego, el jugador deberá seleccionar las opciones que más le convenga de la pantalla de opciones, en esta versión de la aplicación obligatoriamente un usuario de la red, deberá ser el servidor del videojuego y los demás jugadores se unirán a la sesión previamente iniciada por el servidor.

El servidor será el responsable de configurar la sesión del videojuego, como por ejemplo: el límite de muertes, el escenario, límite de usuario que pueden estar conectados, modelo 3D del jugador, etc.

3 Metas

Proveer al usuario de una interfaz intuitiva y fácil de utilizar para el inicio de la aplicación sin inconvenientes.

4 Metas funcionales

Configuración de la aplicación en un entorno Cliente/Servidor.

5 Flujo

5.1 Flujo básico

5.1.1 Selección de las opciones del servidor

Un usuario designado será quien configure las opciones de la sesión de red, además será quien la inicie y dentro de la programación de la aplicación será el responsable de la red, desde el tráfico que puede existir hasta la terminación de la misma.

5.1.2 Selección de las opciones del cliente

Previamente debe haber una sesión de red a la que el usuario pueda unirse, el jugador podrá seleccionar el modelo que representara en el videojuego, para luego unirse a la sesión de red existente.

5.1.3 Inicio de la aplicación

Solo el anfitrión de la red será el responsable de iniciar la sesión para que los demás usuarios de la red se unan a ella, también tendrá la responsabilidad de terminar la sesión.

5.2 Flujo alternativo

5.2.1 Ingresar directamente al desarrollo del juego

Una vez iniciada la aplicación es posible ingresar directamente al videojuego, pero como diseño y estética de este tipo de productos, esto no es lo más recomendable.

6 Categoría

Este caso de uso de negocio forma parte del núcleo del proyecto, específicamente forma parte del administrador de pantallas, y como interactúa directamente con el usuario también integra el administrador de entrada de datos.

7 Riesgo

Ver lista de riesgos.

8 Propietario del proceso

El propietario es el Administrador de pantallas y el administrador de entrada de datos, componentes que se inician al ejecutar la aplicación y que forma parte del motor del videojuego desarrollado.

9 Precondiciones

Cumplir con los requerimientos de software y hardware para ingresar a la aplicación.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de casos de uso de negocio: Ejecución del videojuego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
09/03/2011	1.0	Elaboración del documento de casos de uso de negocio	Wilmer Carrera

Tabla de Contenidos

Especificación de casos de uso de negocio:

Ejecución del videojuego

1 Introducción

1.1 Propósito

El principal propósito de los casos de uso de negocio es proporcionar información sobre los procesos que se desarrollan en la ejecución del videojuego. Esta información es utilizada para elaborar un plan de desarrollo para el proyecto.

1.2 Alcance

Este documento se aplica al proyecto “Ibarra Game v1.0 & Ibarra Game Network v1.0” que será desarrollado como tesis para la obtención del título de Ingeniero en Sistemas Computacionales de la Universidad Técnica del Norte.

1.3 Definiciones, Siglas y Abreviaciones

Ver glosario

1.4 Referencias

Las referencias aplicables son:

- Documento de visión v1.0
- Lista de riesgos v1.0
- Glosario 1.0
- Metodología RUP, 1999, Rational Software Corp.

1.5 Descripción

El documento de casos de uso de negocio contiene la siguiente información:

Descripción del caso de uso de negocio que provee una descripción del caso de uso de negocio, sus objetivos, metas y flujo de trabajo.

2 Ejecución del videojuego

2.1 Descripción

El caso de uso de negocio: Ejecución del videojuego corresponde al desarrollo de la lógica de la aplicación dependiendo de qué versión se esté utilizando, es la parte más importante de todo el sistema desarrollado y será el responsable de la comunicación con de todos los módulos que componen el motor del videojuego como parte de un todo.

3 Metas

Dar al usuario entretenimiento a través de un videojuego.

4 Metas funcionales

Interacción de los módulos que componen el motor de videojuegos desarrollado.

5 Flujo

5.1 Flujo básico

5.1.1 Cumplir con los objetivos del videojuego.

En la versión para un solo jugador, el usuario deberá cumplir que las objetivos que le serán indicados durante la ejecución de la aplicación. En la versión en red, el servidor será quien especifique el criterio para ganar la partida.

5.2 Flujo alternativo

N/A

6 Categoría

Este caso de uso de negocio forma parte del núcleo del proyecto.

7 Riesgo

Ver lista de riesgos.

8 Propietario del proceso

N/A.

9 Precondiciones

Cumplir con los requerimientos de software y hardware para ingresar a la aplicación.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de casos de uso de negocio:

Criterio para ganar la partida

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
09/03/2011	1.0	Elaboración del documento de casos de uso de negocio	Wilmer Carrera

Tabla de Contenidos

Especificación de casos de uso de negocio:

Criterio para ganar la partida

1 Introducción

1.1 Propósito

El principal propósito de los casos de uso de negocio es proporcionar información sobre los procesos que se desarrollan para seleccionar el criterio para ganar la partida del videojuego. Esta información es utilizada para elaborar un plan de desarrollo para el proyecto.

1.2 Alcance

Este documento se aplica al proyecto “Ibarra Game v1.0 & Ibarra Game Network v1.0” que será desarrollado como tesis para la obtención del título de Ingeniero en Sistemas Computacionales de la Universidad Técnica del Norte.

1.3 Definiciones, Siglas y Abreviaciones

Ver glosario

1.4 Referencias

Las referencias aplicables son:

- Documento de visión v1.0
- Lista de riesgos v1.0
- Glosario 1.0
- Metodología RUP, 1999, Rational Software Corp.

1.5 Descripción

El documento de casos de uso de negocio contiene la siguiente información:

Descripción del caso de uso de negocio que provee una descripción del caso de uso de negocio, sus objetivos, metas y flujo de trabajo.

2 Criterio para ganar la partida

2.1 Descripción

El caso de uso de negocio: Criterio para ganar la partida corresponde al desarrollo de la lógica de la aplicación que durante se esté ejecutando y si y solo si el jugador está interesado en cumplir con los objetivos del videojuego independientemente de la versión de la aplicación, el usuario deberá de cumplir para culminar el videojuego.

En la versión para un solo jugador el criterio para ganar la partida será el cumplir con los objetivos del videojuego, en la versión multijugador, deberá cumplir con los parámetros impuestos por el servidor del videojuego.

3 Metas

Inducir al usuario a ganar la partida, porque un videojuego sin un objetivo no sería una aplicación de entretenimiento.

4 Metas funcionales

Durante la ejecución de la aplicación se calculará si el usuario cumplió con el criterio para ganar la partida.

5 Flujo

5.1 Flujo básico

5.1.1 Criterio para ganar la partida

En la versión para un solo jugador, el usuario deberá cumplir que las objetivos que le serán indicados durante la ejecución de la aplicación. En la versión en red, el servidor será quien especifique el criterio para ganar la partida.

5.2 Flujo alternativo

N/A

6 Categoría

Este caso de uso de negocio forma parte del núcleo del proyecto.

7 Riesgo

Ver lista de riesgos.

8 Propietario del proceso

N/A.

9 Precondiciones

Cumplir con los requerimientos de software y hardware para ingresar a la aplicación.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de casos de uso de negocio:

Pérdida del videojuego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
09/03/2011	1.0	Elaboración del documento de casos de uso de negocio	Wilmer Carrera

Tabla de Contenidos

Especificación de casos de uso de negocio:

Pérdida del videojuego

1 Introducción

1.1 Propósito

El principal propósito de los casos de uso de negocio es proporcionar información sobre los procesos que se desarrollan cuando el jugador perdió la partida del videojuego. Esta información es utilizada para elaborar un plan de desarrollo para el proyecto.

1.2 Alcance

Este documento se aplica al proyecto “Ibarra Game v1.0 & Ibarra Game Network v1.0” que será desarrollado como tesis para la obtención del título de Ingeniero en Sistemas Computacionales de la Universidad Técnica del Norte.

1.3 Definiciones, Siglas y Abreviaciones

Ver glosario

1.4 Referencias

Las referencias aplicables son:

- Documento de visión v1.0
- Lista de riesgos v1.0
- Glosario 1.0
- Metodología RUP, 1999, Rational Software Corp.

1.5 Descripción

El documento de casos de uso de negocio contiene la siguiente información:

Descripción del caso de uso de negocio que provee una descripción del caso de uso de negocio, sus objetivos, metas y flujo de trabajo.

2 Pérdida del videojuego

2.1 Descripción

El caso de uso de negocio: Pérdida del videojuego corresponde al desarrollo de la lógica de la aplicación que durante se esté ejecutando el usuario puede perder la partida.

A excepción de la versión en red, la aplicación en su versión para un solo jugador, el usuario solo puede perder si algún personaje le quita todos los puntos de vida. En la versión en red, los demás usuarios serán los responsables de quitar los puntos de vida a los demás jugadores, en esta versión si el usuario muere, luego de cierto tiempo será reintegrado al juago mientras el juego este activo, es decir, mientras no haya ganador.

En la versión para un solo jugador si el usuario pierde todos sus puntos de vida, será el usuario quien decida si desea continuar o no, en caso de que desee continuar, volverá a la posición inicial dependiendo de la misión que esté llevando a cabo.

3 Metas

Incluir la opción de muerte del usuario para mantener al usuario interesado en cumplir con los objetivos del videojuego.

4 Metas funcionales

Durante la ejecución de la aplicación se verificara los puntos de vida del jugador y de acuerdo con la lógica del videojuego se establecerá si ha perdido o no.

5 Flujo

5.1 Flujo básico

5.1.1 Reducción de los puntos de vida

El jugador reducirá sus puntos de vida si y solo si recibe daño por parte de objetos que pueden provocar daño.

5.1.2 Pérdida del videojuego

Dependiendo de la configuración de la aplicación, y durante la ejecución del videojuego se verificara si el usuario ha perdido.

5.2 Flujo alternativo

N/A

6 Categoría

Este caso de uso de negocio forma parte del núcleo del proyecto.

7 Riesgo

Ver lista de riesgos.

8 Propietario del proceso

N/A.

9 Precondiciones

Cumplir con los requerimientos de software y hardware para ingresar a la aplicación.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de casos de uso de negocio:

Salir del videojuego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
09/03/2011	1.0	Elaboración del documento de casos de uso de negocio	Wilmer Carrera

Tabla de Contenidos

Especificación de casos de uso de negocio:

Salir del videojuego

1 Introducción

1.1 Propósito

El principal propósito de los casos de uso de negocio es proporcionar información sobre los procesos que se desarrollan cuando el jugador decide salir del videojuego. Esta información es utilizada para elaborar un plan de desarrollo para el proyecto.

1.2 Alcance

Este documento se aplica al proyecto “Ibarra Game v1.0 & Ibarra Game Network v1.0” que será desarrollado como tesis para la obtención del título de Ingeniero en Sistemas Computacionales de la Universidad Técnica del Norte.

1.3 Definiciones, Siglas y Abreviaciones

Ver glosario

1.4 Referencias

Las referencias aplicables son:

- Documento de visión v1.0
- Lista de riesgos v1.0
- Glosario 1.0
- Metodología RUP, 1999, Rational Software Corp.

1.5 Descripción

El documento de casos de uso de negocio contiene la siguiente información:

Descripción del caso de uso de negocio que provee una descripción del caso de uso de negocio, sus objetivos, metas y flujo de trabajo.

2 Salir del videojuego

2.1 Descripción

El caso de uso de negocio: Salir del videojuego corresponde a una de las opciones que el usuario puede escoger mientras se está ejecutando el videojuego.

En la versión de red si un usuario sale del videojuego mientras se está ejecutando, este automáticamente perderá todos sus puntos acumulados y prácticamente perdería el videojuego. Con la opción de ingresar nuevamente pero con 0 puntos acumulados. En el caso de que sea el anfitrión de la red, si sale del videojuego ya iniciado, todos los clientes de la sesión serán expulsados automáticamente.

En la versión de un solo jugador puede salir cuando desee, pero cuando vuelva a ingresar al videojuego será situado al principio de la misión que estuvo llevando acabo anteriormente.

3 Metas

Dejar a libre criterio al usuario la decisión de salir de la aplicación.

4 Metas funcionales

Grabar los estados de la partida y proveer el mecanismo más adecuado para cuando el usuario deje el videojuego.

5 Flujo

5.1 Flujo básico

5.1.1 Menú emergente en la ejecución de la aplicación

El jugador mientras este ejecutando la aplicación podrá desplegar un menú emergente en el cual podrá selección si desee abandonar la partida o continuar.

5.1.2 Salir del videojuego

En caso de seleccionar la opción salir del videojuego, terminar la ejecución del mismo y será direcciona la pantalla inicial de la aplicación, en cualquier caso volverá al menú de inicio de la aplicación dependiendo de la versión que este ejecutando.

5.1.3 Guardar el estado de la partida

Solo en la versión para un solo jugador, según sea el progreso del usuario, la partida será guardada automáticamente.

5.2 Flujo alternativo

También puede salir presionado las teclas para salir de cualquier aplicación Alt + F4 en este caso, no se continuara con el flujo correcto para salir de la aplicación.

6 Categoría

Este caso de uso de negocio forma parte del núcleo del proyecto.

7 Riesgo

Ver lista de riesgos.

8 Propietario del proceso

N/A.

9 Precondiciones

Cumplir con los requerimientos de software y hardware para ingresar a la aplicación.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Caso de desarrollo

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
10/03/2011	1.0	Elaboración del documento caso de desarrollo.	Wilmer Carrera

Tabla de Contenidos

Caso de Desarrollo

1 Introducción

Para el desarrollo del presente proyecto se decidió adoptar RUP como metodología de desarrollo de software.

1.1 Propósito

El propósito del documento es describir el proceso de desarrollo adoptado por el Sistema.

1.2 Ámbito

El presente documento es aplicable a cualquier versión que se construyó a partir del motor de videojuegos desarrollado, como parte de un solo proyecto.

1.3 Definiciones, Siglas y Abreviaciones

Ver glosario.

1.4 Referencias

Las referencias aplicables son:

- Documento de visión v1.0
- Lista de riesgos v1.0
- Glosario 1.0
- Metodología RUP, 1999, Rational Software Corp.

1.5 Perspectiva

El presente documento explica en detalle el ciclo de vida de un proyecto siguiendo la metodología RUP. Además de requerimientos y especificaciones propias del sistema.

2 Clico de vida del Proyecto

El ciclo de vida del software en Rational Unified Process (RUP) está compuesto de 4 fases a través del tiempo, cada una concluye con hitos, cada fase es esencialmente un espacio de tiempo entre 2 hitos relevantes. Al final de cada fase una valoración es desarrollada para determinar los objetivos permitidos para pasar a la siguiente fase si la valoración es exitosa.

2.1 Incepción

Actividad	Flujo de trabajo	Artefactos
Gestión de Proyecto:		
Concepción y aprobación	Concepción del proyecto. Evaluación del alcance y riesgos del proyecto. Plan de desarrollo del proyecto.	Caso de negocio Visión (Preliminar) Plan de desarrollo de software. Lista de riesgos.
Revisión del plan de iteración inicial.	Plan para la siguiente iteración.	Plan de iteración
Gestión / Monitoreo	Gestión de la iteración Monitoreo y control del proyecto.	Registros de revisión Valoración de la iteración
Planificación de la siguiente iteración	Plan de la siguiente iteración Plan de desarrollo de software Definir la misión de evaluación.	Plan de iteración, actualización basada sobre la nueva funcionalidad que debe ser añadida en la nueva iteración. Plan de desarrollo de software actualizado de acuerdo a los cambios en el alcance y en el registro. La lista de riesgos debe ser revisada y analizada si existen riesgos restantes de importancia. Plan de pruebas actualizado para reflejar la misión para las pruebas de la próxima iteración. El resultado de la valoración de la iteración debe ser considerado para determinar si es necesario realizar cambios en el proceso.
Requerimientos		
Definición del alcance inicial	Análisis del problema Entendimiento de las necesidades del proyecto. Definición del sistema. Gestión del alcance del sistema. Gestión de cambios en los requerimientos.	El principal artefacto es la visión completa, incluyendo los casos de uso más importantes y por su prioridad la visión será más refinada cuanto más detallados sean los casos de uso.
Prototipo de	Prototipo de interfaz de	Prototipo de interfaz de usuario.

interfaz de usuario.	usuario.	
Refinar la definición del sistema	Refinar la definición del sistema (excepto las actividades relacionadas con las interfaces de usuario ya que están en tareas separadas.)	Modelo de casos de uso (con los casos de uso de alta prioridad detallados.) Visión actualizada. Especificaciones suplementarias.
Análisis y diseño		
Elaborar una síntesis de la arquitectura	Elaborar una síntesis de la arquitectura.	Prueba de concepto de la arquitectura.

2.2 Elaboración

Actividad	Flujo de trabajo	Artefactos
Gestión de Proyecto:		
Gestión / Monitoreo	Gestión de la iteración Monitoreo y control del proyecto. Alcanzar la misión aceptable (actividad: valorar y promover la calidad del sistema, valorar y mejorar el esfuerzo de las pruebas.)	Registros de revisión Valoración de la iteración.
Planificación de la siguiente iteración	Plan de la siguiente iteración Plan de desarrollo de software Definir la misión de evaluación.	Plan de iteración, actualización basada sobre la nueva funcionalidad que debe ser añadida en la nueva iteración. Plan de desarrollo de software actualizado de acuerdo a los cambios en el alcance y en el registro. La lista de riesgos debe ser revisada y analizada si existen riesgos restantes de importancia. Plan de pruebas actualizado para reflejar la misión para las pruebas de la próxima iteración. El resultado de la valoración de la iteración debe ser considerado para determinar si es necesario realizar cambios en el proceso.
Requerimientos		
Prototipo de interfaz de usuario	Prototipo de interfaz de usuario	Prototipo de interfaz de usuario.
Gestión de cambios en los requerimientos.	Gestión de cambios en los requerimientos.	Ordenar y priorizar los requerimientos
Refinar la definición del	Refinar la definición del sistema.	Modelos de casos de uso. Visión actualizada.

sistema.		Especificaciones suplementarias.
Definición de la arquitectura.		
Definir arquitectura candidata.	Definir arquitectura candidata.	Definir arquitectura candidata.
Estructurar el modelo de implementación	Estructurar el modelo de implementación.	
Refinar la arquitectura	Refinar la arquitectura.	
Soporte de desarrollo		
Desarrollo de componentes/características.	Animación de los modelos del videojuego. Implementar componentes de videojuegos. Integración de los componentes del videojuego como parte del motor del mismo. Pruebas y evaluación.	Modelo de desarrollo de software. Modelo de implementación. Plan de integración.

2.3 Construcción

Actividad	Flujo de trabajo	Artefactos
Gestión de Proyecto:		
Gestión / Monitoreo	Gestión de la iteración Monitoreo y control del proyecto. Alcanzar la misión aceptable.	Registros de revisión Valoración de la iteración
Planificación de la siguiente iteración	Plan de la siguiente iteración Plan de desarrollo de software Definir la misión de evaluación.	Plan de iteración, actualización basada sobre la nueva funcionalidad que debe ser añadida en la nueva iteración. Plan de desarrollo de software actualizado de acuerdo a los cambios en el alcance y en el registro. La lista de riesgos debe ser revisada y analizada si existen riesgos restantes de importancia. Plan de pruebas actualizado para reflejar la misión para las pruebas de la próxima iteración. El resultado de la valoración de la iteración debe ser considerado para determinar si es necesario realizar cambios en el proceso.
Requerimientos		
Prototipo de interfaz de	Prototipo de interfaz de usuario.	Prototipo de interfaz de usuario.

usuario.		
Gestión de cambios en los requerimientos.	Gestión de cambios en los requerimientos.	
Refinar la definición del sistema	Refinar la definición del sistema (excepto las actividades relacionadas con las interfaces de usuario ya que están en tareas separadas.)	Modelo de casos de uso (con los casos de uso de alta prioridad detallados.) Visión actualizada. Especificaciones suplementarias. El esfuerzo de definir los requerimientos se dividen en tareas más pequeñas con duración más corta.
Soporte de desarrollo		
Refinar arquitectura	Refinar la arquitectura	
Corrección de defectos.	Flujo de trabado idéntico a la actividad “Desarrollo de componentes / características” La corrección de defectos en el código fuente es una tarea importante.	
Desarrollo de componentes/características	Deseno del sistema Implementación de componentes. Integración del sistema Pruebas y evaluación	Modelo de desarrollo del software Modelo de implementación Plan de integración.

2.4 Transición

Actividad	Flujo de trabajo	Artefactos
Gestión de Proyecto:		
Gestión / Monitoreo	Gestión de la iteración Monitoreo y control del proyecto.	Registros de revisión Valoración de la iteración
Planificación de la siguiente iteración	Plan de la siguiente iteración Plan de desarrollo de software Definir la misión de evaluación.	Plan de iteración, actualización basada sobre la nueva funcionalidad que debe ser añadida en la nueva iteración. Plan de desarrollo de software actualizado de acuerdo a los cambios en el alcance y en el registro. La lista de riesgos debe ser revisada y analizada si existen riesgos restantes de importancia. Plan de pruebas actualizado para reflejar la misión para las pruebas de la próxima iteración.

		El resultado de la valoración de la iteración debe ser considerado para determinar si es necesario realizar cambios en el proceso.
Requerimientos		
Prototipo de interfaz de usuario.	Prototipo de interfaz de usuario.	Prototipo de interfaz de usuario.
Gestión de cambios de los requerimientos.	Gestión de cambios de los requerimientos.	
Refinar la definición del sistema	Refinar la definición del sistema (excepto las actividades relacionadas con las interfaces de usuario ya que están en tareas separadas.)	Modelo de casos de uso (con los casos de uso de alta prioridad detallados.) Visión actualizada. Especificaciones suplementarias.
Soporte de desarrollo		
Refinar arquitectura	Refinar la arquitectura.	Refinar la arquitectura.
Corrección de defectos.	Flujo de trabajo idéntico a la actividad.	
Desarrollo de componentes/características	Animación de los modelos de los videojuegos. Implementación de componentes. Integración del sistema. Pruebeas y evaluación.	Modelo de desarrollo del sistema. Modelo de implementación. Plan de integración.
Despliegue		
Plan de despliegue	Plan de despliegue	Plan de desarrollo de software
Desarrollo del material de soporte. **Gestión de aceptación de pruebas de ambiente de desarrollo**	Desarrollo del material de soporte. Gestión de aceptación de pruebas.	Material de soporte para el usuario final. Producto instalado y aceptado en el ambiente de desarrollo.
Unidad de despliegue del producto.	Unidad de despliegue del producto.	Artefactos de instalación Notas del release. Unidades de despliegue.
Gestión y aceptación de pruebas en ambiente de producción.	Gestión de aceptación de pruebas.	Producto instalado y aceptado en ambiente de producción

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Plan de Administración de Requerimientos

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
10/03/2011	1.0	Elaboración del documento plan de administración de requerimientos.	Wilmer Carrera

Tabla de Contenidos

Plan de Administración de Requerimientos

1 Introducción

1.1 Propósito

El propósito de este plan es establecer y documentar sistemáticamente la obtención, organización y documentación de los requisitos del sistema.

1.2 Alcance

Este plan provee información sobre el proyecto a desarrollado como tesis para la obtención del título de Ingeniero en Sistemas Computacionales.

1.3 Definiciones, Siglas y Abreviaciones

Ver glosario.

1.4 Referencias

- Documento de visión
- Plan de desarrollo de software.

1.5 Perspectiva

Dar a conocer los requerimientos del sistema para su desarrollo.

2 Administración de Requerimientos

2.1 Organización, Responsabilidades e Interfaces

2.1.1 Usuario

Persona quien utilizara el sistema que se está desarrollando.

2.1.2 Coordinar del proyecto

Responsable a nivel directivo del proyecto.

2.1.3 Desarrollador

Persona responsable del desarrollo de requerimientos funcionales de acuerdo con los procesos y estándares adoptados en el proyecto. Puede incluir actividades en cualquiera de las disciplinas de requerimientos: análisis y diseño, implementación y pruebas.

2.2 Herramientas, Entorno e Infraestructura

El entorno deberá constar de computadoras mínimo 3 máquinas, unidas en red y conectados a un switch con puertos suficientes. Además de todos los periféricos necesarios.

Las computadoras deberán estar provistas de Microsoft Windows en cualquier presentación de las versiones existentes del mercado, .net 3.5 o posterior, IDE Visual Studio 2008 o posterior, XNA 3.1 o posterior.

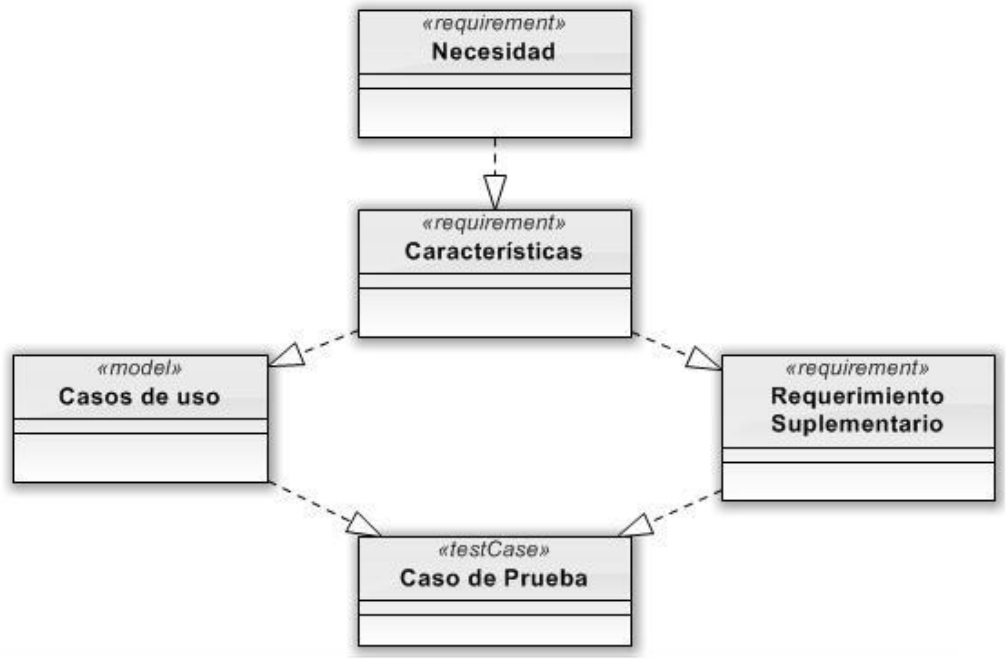
Para la correcta visualización de los gráficos todas las computadoras deben estar provistas de una tarjeta de video con soporte de pixel y vertex shader 3.0 o posterior.

3 Programa de Administración de Requerimientos

3.1 Identificación de Requerimientos

Artefacto (Tipo de Documento)	Item de Trazabilidad	Descripción
Requerimientos de Stakeholder (STR)	Requerimientos de Stakeholder (STRQ)	Requerimientos clave, incluyendo peticiones de cambio de los stakeholders.
Visión (VIS)	Necesidades de Stakeholder (NEED)	Necesidades clave de stakeholder o usuarios
Visión (VIS)	Característica (FEAT)	Condiciones o capacidades de esta versión del sistema
Modelo de casos de uso	Caso de uso (UC)	Casos de usi para esta versión documentados en MS Word.
Especificaciones suplementarias (SS)	Requerimientos suplementarios (SUPP)	Requerimientos no funcionales que no son capturados en el modelo de casos de uso

3.2 Trazabilidad



3.3 Atributos

3.3.1 Atributos de “Casos de uso”

Los casos de uso y especificaciones suplementarias serán administrados usando los atributos definidos en esta sección.

Estado

Propuesto	Bajo discusión.
Aprobado	Aprobados por el canal oficial para su diseño e implementación.
Rechazado	Rechazados por el canal oficial.
Incorporado	Casos de uso validado dentro del sistema de pruebas.

Prioridad

Alta	Casos de uso primordiales para el negocio.
Media	Prioridad media en relación a los casos de uso.
Baja	La implementación de los casos de uso pueden ser aplazados para siguientes iteraciones.

3.3.2 Atributos para casos de prueba.

Estado

Pendiente	Caso de negocio que no se ha desarrollado.
Error	Casos de pruebas erróneas.
Paso condicional	Pruebas completadas con pruebas ocasionadas por condiciones que no se han cumplido.
Exitosa	Pruebas realizadas exitosamente.

Las pruebas serán realizadas por los usuarios del sistema en el ambiente de desarrollo.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Plan de Desarrollo de Software

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
13/03/2011	1.0	Elaboración del documento Plan de Desarrollo de Software	Wilmer Carrera

Tabla de Contenidos

Plan de Desarrollo de Software

1 Introducción

Este plan de Desarrollo de Software es una versión preliminar para ser incluida como complemento a la propuesta presentada ante el Consejo Académico como plan de anteproyecto de tesis para la obtención del título como Ingeniero en Sistemas Computacionales.

El proyecto se ha basado en una metodología de Rational Unified Process con el fin de implantar un esquema inicial de esta metodología para futuros desarrollos.

La orientación del proyecto propuesto constituye una configuración del proceso RUP de acuerdo a las particularidades del proyecto, seleccionando roles, actividades y artefactos (documentos entregables)

Este documento a su vez es un entregable de los artefactos de RUP.

1.1 Propósito

El propósito del presente documento es proporcionar la información necesaria para el control del proyecto. En él se describe el enfoque de desarrollo de software.

Los usuarios del Plan de Desarrollo del Software son:

- El líder del proyecto lo utiliza para organizar la agenda, necesidades de recursos y para realizar el respectivo seguimiento.
- El desarrollador lo usa para entender lo que se debe hacer, cuando deben hacerlo y que otras actividades dependen de ello.

1.2 Alcance

El plan de desarrollo del software describe el plan completo usado para el desarrollo del sistema. El detalle de las iteraciones individuales se describe en los planes de cada iteración, documentos que se aportan de manera separada. Durante el proceso de desarrollo en el artefacto de Visión se definen las características del producto a desarrollarse, lo cual constituye la base para el desarrollo de las iteraciones.

Para la elaboración del presente documento se ha tomado como base el documento presentado en la defensa del anteproyecto ante el consejo académico y que fue aprobado para su desarrollo.

1.3 Resumen

El resto del documento está organizado en las siguientes secciones:

Vista general del proyecto: proporciona una descripción del propósito, alcance y objetivos del proyecto estableciendo los artefactos que serán producidos y utilizados durante el proyecto.

Organización del proyecto: describe la estructura organizacional del desarrollador.

Gestión del proceso explica los costos y planificación estimada, define las fases e hitos del proyecto y se describe como se realizara si seguimiento.

Planes y guías de aplicación: proporciona una vista global del proceso de desarrollo de software, incluyendo métodos, herramientas y técnicas que serán utilizadas.

2 Vista General del Proyecto

2.1 Propósito, Alcance y Objetivos

Definidos en el documento presentado ante el consejo académico.

2.2 Suposiciones y Restricciones

- La calidad del software de este tipo se limitara o lo considerado posible por parte del desarrollador ya que un sistema de este tipo es demasiado exigente por parte de los usuarios.
- El hardware para usar la aplicación deben cumplir con los requisitos para su correcto funcionamiento.

Debido a la complejidad del sistema la lista de suposiciones y restricciones se incrementaran durante el desarrollo del proyecto.

2.3 Entregables del Proyecto

El presente proyecto será desarrollado de acuerdo a la metodología RUP, la cual define la generación de artefactos que se constituyen como documentos entregables.

Todos los artefactos están sujetos a modificaciones a lo largo del proceso de desarrollo, por lo que una versión definitiva y completa de cada uno de ellos se obtendrá al finalizar el proyecto.

A continuación se presenta la lista de artefactos propuestos para el sistema.

1. **Plan de desarrollo del software.**

   El presente documento.

2. **Modelo de casos de uso del negocio.**

   Presentan los modelos de las funciones de negocio vistas desde la perspectiva de los actores externos (Agentes de registro, solicitantes finales, etc.) Permite situar al sistema en el contexto organizacional haciendo énfasis en los objetivos en este ámbito. En cada modelo se presenta un diagrama de caso de uso del negocio.

3. **Visión**

   Este documento define la visión del producto desde la perspectiva del cliente, especificando las necesidades y características del producto.

4. **Glosario**

   Documento que define los principales términos usados en el proyecto.

5. **Modelos de casos de uso.**

   Presenta las funciones del sistema y los actores que hacen uso de ellas. Se representa mediante modelos de casos de uso.

6. **Especificaciones de casos de uso**

   Para los casos de uso que lo requieran se realiza una descripción detallada utilizando una plantilla de documento, donde se incluyen precondiciones, post condiciones, flujo de eventos, requisitos no funcionales asociados. Para casos de uso cuyo flujo de eventos sea complejo se podrá adjuntarse una representación grafica mediante un diagrama de actividad.

7. **Especificaciones adicionales.**

   Este documento capturara todos los requisitos que no han sido incluidos como parte de los casos de uso y se refieren a requisitos no funcionales globales. Dichos requisitos incluyen: requisitos legales o normas, aplicación de estándares, requisitos de calidad del proyecto: tales como confiabilidad, desempeño, etc. u otros requisitos de ambiente, tales como sistema operativo, compatibilidad, etc.

8. **Prototipos de interfaces de usuario.**

   Se trata de prototipos que permiten al usuario hacerse una idea más o menos precisa de las interfaces que proveerá el sistema y así conseguir retroalimentación de su parte respecto a los requisitos del sistema. Estos prototipos serán: dibujos a mano en papel, dibujos con alguna herramienta grafica, siguiendo ese orden de acuerdo al avance del proyecto.

   Solo los de este último tipo serán entregables al final de la fase de elaboración. Así mismo este documento será desechado en la fase de Construcción en la medida que la resultado de las iteraciones vayan desarrollando el producto final.

9. **Modelo de análisis y diseño**

   Este modelo establece la realización de los casos de uso en clases y pasando desde una representación en términos de análisis (sin incluir aspectos de implementación) hacia una de diseño de acuerdo al avance del proyecto.

10. Modelo de implementación

Este modelo es una colección de componentes y los subsistemas que los contienen. Estos componentes incluyen ficheros ejecutables, ficheros de código fuente, y todo tipo de ficheros necesarios para la implantación y despliegue del sistema. (Este modelo es solo una versión preliminar al final de la fase de elaboración, posteriormente presenta refinamiento)

11. Modelo de despliegue

Este modelo muestra el despliegue, la configuración de tipos de nodos del sistema, en los cuales se hará el despliegue de los componentes.

12. Casos de Prueba.

Cada prueba es específica mediante un documento que establece las condiciones de ejecución, las entradas de la prueba, y los resultados esperados. Estos casos de prueba son aplicados como pruebas de regresión en cada iteración. Cada caso de prueba llevara asociado un procedimiento de prueba con las instrucciones para realizar la prueba y dependiendo del tipo de prueba de dicho procedimiento podrá ser automatizado mediante un archivo de pruebas.

13. Solicitud de cambio

Los cambios propuestos para los artefactos se formalizaran mediante este documento. Mediante este documento se hace seguimiento de los artefactos detectados, solicitud de mejoras o cambios en los requisitos del producto.

Así se provee un registro de decisiones de cambios, de su evaluación e impacto y se asegura que estos sean conocidos por el desarrollador. Los cambios se establecen respecto de la última línea base.

14. Plan de Iteración

Es un conjunto de actividades y tareas ordenadas temporalmente, con recursos asignados, dependencias entre ellas. Se realiza por cada iteración y para todas las fases.

15. Evaluación de Iteración

Este documento incluye la evaluación de los resultados de cada iteración, el grado en el cual se han conseguido los objetivos de la iteración, las lecciones aprendidas y los cambios a ser realizados.

16. Lista de Riesgos

Este documento incluye una lista de los riesgos conocidos y vigentes en el proyecto en orden decreciente de importancia y con acciones especificas de contingencia o para su mitigación.

17. Manual de Instalación.

Este documento incluye las instrucciones para realizar la instalación del producto.

18. Material de apoyo al usuario final

Corresponde a un conjunto de documentos y facilidades de uso del sistema, incluyendo guías del usuario, guías de operación, guías de mantenimiento.

19. Producto.

Los archivos del producto serán empaquetados en un instalador y almacenados en un CD con los mecanismos necesarios para facilitar su instalación. Se obtendrá una nueva versión al final de cada iteración.

2.4 Evolución del Plan de Desarrollo de Software

El plan de desarrollo de software se revisara semanalmente y se refinara antes del comienzo de cada iteración.

3 Organización del Proyecto

3.1 Estructura organizacional

Debido a que el presente proyecto forma parte del desarrollo de tesis presentada para la obtención del título de Ingeniería en Sistemas Computacionales, la persona quien desarrollara el sistema será el responsable del mismo desde la concepción hasta culminación del mismo, con el apoyo del director de tesis quien será el responsable de la revisión de la misma.

El personal asociado es el siguiente:

Desarrollador: responsable del proyecto desde las fases iniciales hasta la culminación del mismo.

Director de Tesis: revisor del avance y resultados de proyecto.

3.2 Interfaces Externas

N/A

3.3 Roles y Responsabilidades

A continuación se describen las principales responsabilidades de cada uno de los puesto del equipo de desarrollo de acuerdo con los roles que desempeñan RUP.

Persona	Rol Rational Unified Process
Desarrollador	El presente proyecto al ser parte del desarrollo de tesis, debe ser cumplida por la persona que la propuso, cabe mencionar que la aplicación no tiene vínculos con otras entidades, debido a estas razones todos las fases que define RUP serán afrontadas por esta persona entre las cuales constan: definición de requerimientos, diseño y

	análisis del sistema, pruebas, etc. para culminar con éxito el sistema propuesto.
Director de Tesis	Como reglamento para la presentación de Tesis, se establece que toda Tesis debe ser dirigida por un Director de la misma quien será responsable de la revisión y tareas de supervisión para alcanzar los objetivos propuestos.

4 Gestión del Proceso

4.1 Plan del Proyecto

En esta sección se presenta la organización en fases e iteraciones y el calendario del proyecto.

4.1.1 Plan de Fases

Fase	Nro. Iteraciones	Duración
Fase de Inicio	2	4 semanas
Fase de Elaboración	4	8 semanas
Fase de Construcción	15	8 semanas
Fase de Transición	3	4 semanas

Los hitos que marcan el final de cada fase se describen en la siguiente tabla:

Descripción	Hito
Fase de inicio	En esta fase desarrollara los requisitos de producto desde la perspectiva del usuario, los cuales serán establecidos en el artefacto de Visión. Los principales casos de uso serán identificados y se hará un refinamiento del plan de desarrollo del proyecto.
Fase de Elaboración	En esta fase se analizaran los requisitos y se desarrolla un prototipo de arquitectura (incluyendo las partes más relevantes y/o criticas de sistema). Al final de esta fase, todos los casos de uso correspondientes a requisitos que serán implementados en la primera versión de la fase de Construcción deben estar analizados y diseñados (en el modelo Análisis y Diseño). La revisión y aceptación del prototipo de la arquitectura del sistema marca el final de esta fase.
Fase de Construcción	Durante la fase de Construcción se termina de de analizar y diseñar todos los casos de uso, refinando el modelo Análisis / Diseño, dependiendo del número de iteraciones se construyen varias versiones a la cual se aplican pruebas.
Fase de Transición	En esta fase se preparan 3 versiones para su distribución, la implantación será decisión del usuario final.

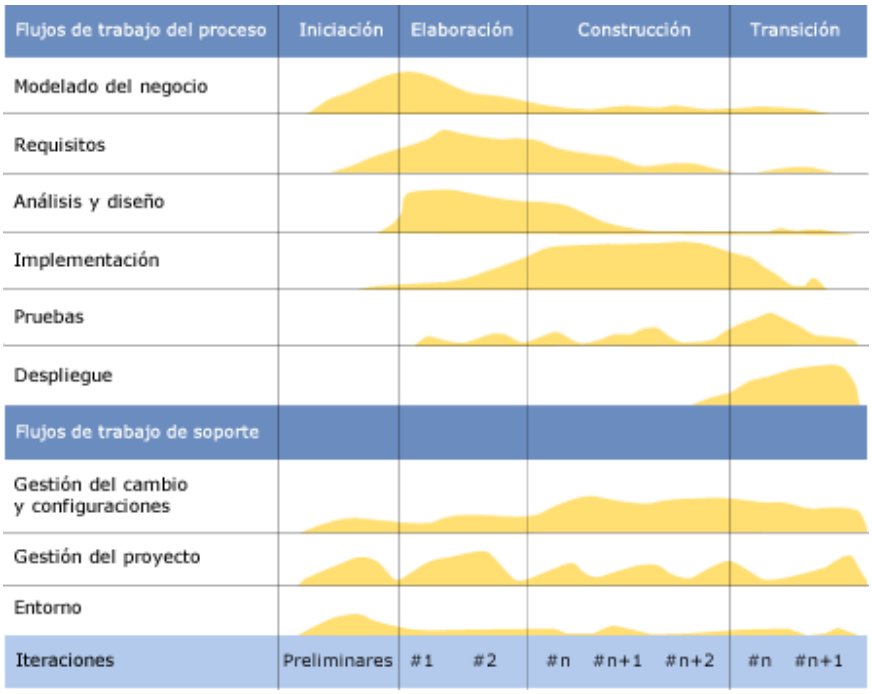
4.1.1.1Objetivos de las Iteraciones.

Fase	Iteración	Descripción	Hitos asociados	Riegos direccionados
Incepción	Iteración preliminar	Define el modelo de negocio, requerimientos del producto, plan del proyecto y casos de negocio.	Revisión de casos de uso.	Aclarar los requerimientos, desarrollar planes de proyecto realista y alcance. Determina la viabilidad de los proyectos desde el punto de vista empresarial.
Fase de Elaboración	Desarrollar una arquitectura prototipo.	Completar el análisis y desarrollo de todos los casos de usos. Desarrollar arquitectura prototipo.	Arquitectura prototipo.	Arquitectura del software clarificada. Técnicas de mitigación de riesgos. Prototipo para que el usuario revise.
Fase de Construcción	C1. Iteración desarrollo Beta.	Implementar y probar casos de uso para proveer la versión Beta.	Beta	Todas las características claves y arquitectura prospectiva usando en la versión Beta.
	C2. Iteración desarrollo versión inicial.	Implementar y probar casos de uso restantes, corregir defectos en versión Beta. Desarrollar el sistema inicial.	Software	Software revisado por el Director de Tesis. Producto de calidad considerable. Minimizar defectos. Costes de calidad reducidos.
	C3. Iteración desarrollar módulo completo.	Incorporar mejoras en los defectos de la versión inicial. Desarrollo del sistema completo.	Software	Liberación rápida asegurando el cumplimiento de los objetivos propuestos. Todas las funciones clave proporcionado el modulo completo.
Fase de Transición.	Sistema	Empaquetar y distribuir.	Sistema.	

4.1.2 Calendario del Proyecto

Se presenta un calendario con las principales tareas del proyecto incluyendo las fases de Inicio y Elaboración, como se ha comentado, la metodología RUP está caracterizado por la realización en paralelo de todas las disciplinas de desarrollo a lo largo del proyecto, con lo cual la mayoría de los artefactos son generados muy tempranamente en el proyecto pero van desarrollándose en mayor o menor grado de acuerdo a la fase e iteración del proyecto.

La siguiente figura ilustra el enfoque, en ella marca el énfasis de cada disciplina (workflow) en un momento determinado del desarrollo.



Ver documento plan de proyecto de titulación.

4.2 Monitoreo y Control del Proyecto.

Gestión de requisitos

Los requisitos del sistema son específicos en el artefacto de Visión. Cada requisito tendrá una serie de atributos tales como importancia, estado, iteración donde se implementa, etc. estos atributos permitirán realizar un seguimiento efectivo de cada requisito.

Control de plazos

El calendario del proyecto tendrá un seguimiento y evaluación semanal por el Director de Tesis para este proyecto.

Control de Calidad.

Los defectos detectados en las revisiones será responsabilidad del desarrollador corregirlas.

Gestión de Riesgos.

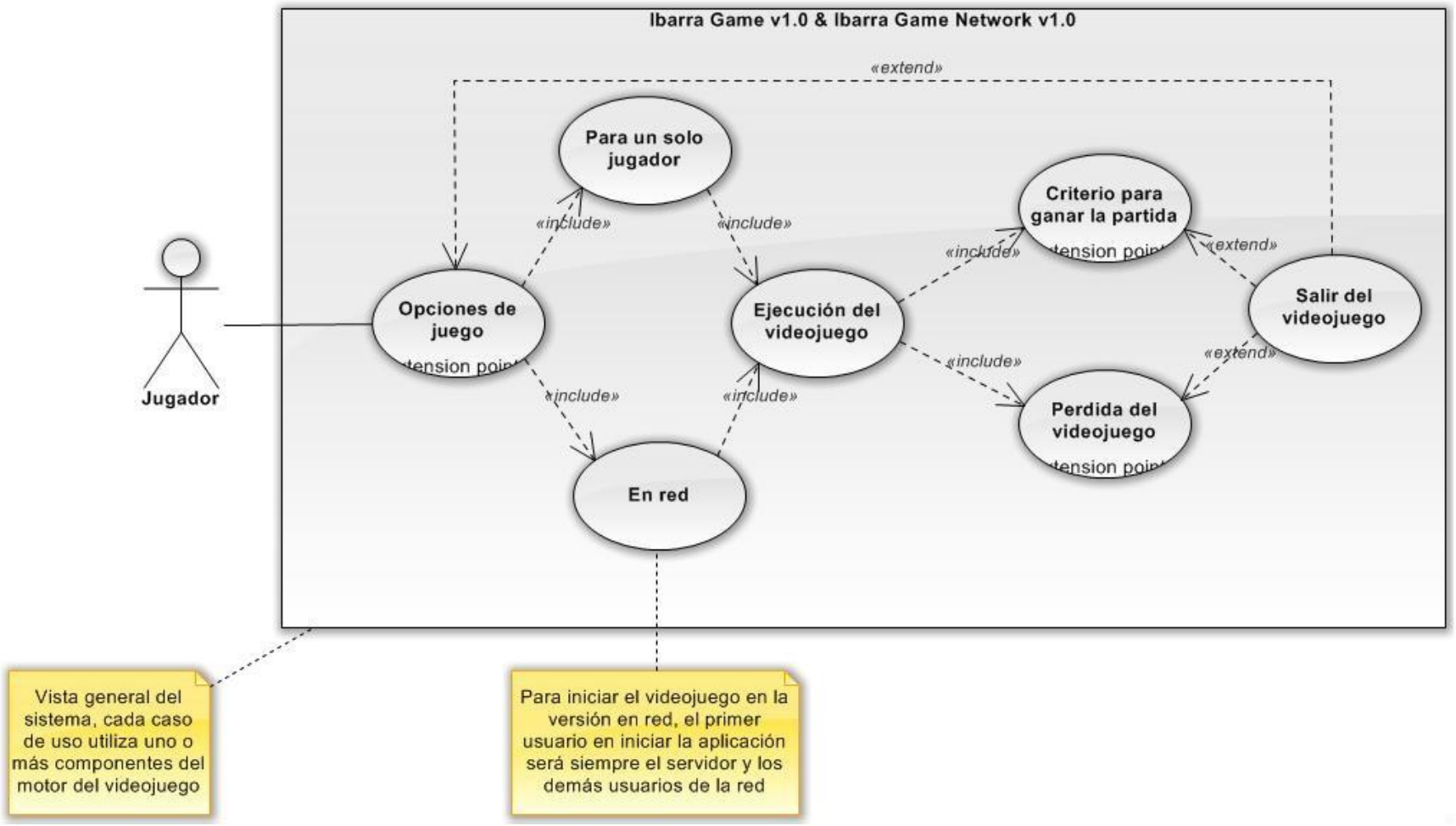
A partir de la fase de inicio se mantendrá una lista de riesgos asociados al proyecto y de las acciones establecidas como estrategia para mitigarlos acciones de contingencia. Esta lista será evaluada al menos una vez en cada iteración.

Gestión de Configuración.

Se realizará una gestión de configuración para llevar a un registro de los artefactos generados y sus versiones.

5 Referencias

Documento presentado ante el Consejo Académico como Plan de Proyecto de Titulación.



El motor de videojuegos desarrollado integra como parte de un todo, los componentes utilizados y establece la comunicación entre ellos, dependiendo de la funcionalidad de estos.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de Caso de Uso:

Opciones de juego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
14/03/2011	1.0	Elaboración del documento de especificación de caso de uso: Opciones de juego	Wilmer Carrera

Tabla de Contenidos

Especificación de Caso de Uso:

Opciones de juego

1 Opciones de juego

1.1 Descripción breve

El caso de uso describe el proceso para la presentación al usuario de las opciones de juego, que forma parte de la configuración del videojuego antes de iniciar el mismo.

Cabe mencionar que en la versión para un solo jugador estas opciones solo forman parte de la estética del mismo mas no representan parámetros configurables del videojuego.

2 Flujo de Eventos

2.1 Flujo básico

El motor de videojuegos incorpora el componente de administrador de pantallas, el cual será el responsable de decidir según el flujo lógico de la aplicación que pantalla será la que se muestre al usuario.

Además del componente de administrador de pantallas, también se inicia el componente de sonido de la aplicación.

Al iniciar la aplicación el usuario podrá visualizar una primera pantalla, con opciones que podrá seleccionar como bien le convenga.

Pantalla inicial de la versión para un solo jugador

Pantalla inicial de la versión multijugador.

Para le versión en red del videojuego las opciones se presentan para quien será el servidor que puede ser cualquier computador y para los clientes del mismo.

La interfaz inicial para el usuario es prioritaria ya que por facilidad para el jugador podrá navegar sin contratiempos dentro de las opciones del mismo, al ser este un producto de entretenimiento, se debe cuidar al máximo los detalles en cuanto a la perspectiva que tendrá el usuario desde la primera vez que se utilice el producto.

2.2 Flujo alternativo

2.2.1 Ingreso directo a al videojuego

En todos los videojuegos vistos del mercado, al probarlos inician con una pantalla de introducción y a continuación muestra una pantalla de opciones que en algunos casos puede configurar opciones de la aplicación y opciones de jugabilidad, por cuestiones de estética este flujo alternativo no es lo más recomendable.

3 Requerimientos especiales

El computador debe tener instalado una tarjeta de video con soporte de pixel y vertex shader 3.0 como mínimo.

Microsoft Windows XP o posterior.

XNA 3.1 en su versión distribuible.

.Net Framework 3.5

4 Precondiciones

Iniciar el motor de videojuegos.

Iniciar el administrador de pantallas

Iniciar el administrador de sonido

Iniciar el administrador de entrada de datos ya sea por teclado o con el ratón.

4.1 Iniciar el motor de videojuegos

Como núcleo del sistema, es obligatorio iniciar el motor de videojuegos que será responsable de comunicar los diferentes componentes entre sí, según sea necesario.

4.2 Iniciar el administrador de pantallas

Lo primero que observara el jugador al iniciar la aplicación será la pantalla de opciones de juego, y será la primera vez donde el jugador interactuara directamente con la aplicación.

4.3 Iniciar el administrador de sonido

El sonido como parte del ambiente del videojuego también será iniciado al principio de la aplicación.

4.4 Iniciar el administrador de entrada de datos ya sea por teclado o con el ratón.

Para la comunicación con el usuario se inicia el administrador de entrada de datos que será el responsable de capturar los eventos generados por el usuario al utilizar el teclado y ratón.

5 Postcondiciones

Inicio del videojuego.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de Caso de Uso:

Para un solo jugador

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
15/03/2011	1.0	Elaboración del documento de especificación de caso de uso: Para un solo jugador.	Wilmer Carrera

Tabla de Contenidos

Especificación de Caso de Uso:

Para un solo jugador

1 Para un solo jugador

1.1 Descripción breve

El proyecto desarrollado comprende 2 versiones del mismo, que parten del mismo motor de videojuegos desarrollado, para un solo jugador y multijugador.

Se puede observar que se ha divido el proyecto en 2 versiones independientes, es decir 2 aplicaciones, la razón por la cual no se unió las versiones es por el tamaño de la aplicaciones y jugabilidad de cada uno de ellos. La primera versión como este caso de uso lo especifica es para un solo jugador, en la cual las opciones para esta versión no cumplen ninguna funcionalidad pero forman parte de la estética de la aplicación.

2 Flujo de Eventos

2.1 Flujo básico

Al iniciar la aplicación y luego de iniciar los componentes requeridos el usuario será capaza de seleccionar las opciones que más le convenga.

Opciones de la versión para un solo jugador

2.2 Flujo alternativo

2.2.1 Ingreso directo a al videojuego

En todos los videojuegos vistos del mercado, al probarlos, inician con una pantalla de introducción y a continuación muestra una pantalla de opciones que en algunos casos se puede configurar opciones de la aplicación y opciones de jugabilidad, por cuestiones de estética este flujo alternativo no es lo más recomendable.

3 Requerimientos especiales

El computador debe tener instalado una tarjeta de video con soporte de pixel y vertex shader 3.0 como mínimo.

Microsoft Windows XP o posterior.

XNA 3.1 en su versión distribuible.

.Net Framework 3.5

4 Precondiciones

Iniciar el motor de videojuegos.

Iniciar el administrador de pantallas

Iniciar el administrador de sonido

Iniciar el administrador de entrada de datos ya sea por teclado o con el ratón.

4.1 Iniciar el motor de videojuegos

Como núcleo del sistema, es obligatorio iniciar el motor de videojuegos que será responsable de comunicar los diferentes componentes entre sí, según sea necesario.

4.2 Iniciar el administrador de pantallas

Lo primero que observara el jugador al iniciar la aplicación será la pantalla de opciones de juego, y será la primera vez donde el jugador interactuara directamente con la aplicación.

4.3 Iniciar el administrador de sonido

El sonido como parte del ambiente del videojuego también será iniciado al principio de la aplicación.

4.4 Iniciar el administrador de entrada de datos ya sea por teclado o con el ratón.

Para la comunicación con el usuario se inicia el administrador de entrada de datos que será el responsable de capturar los eventos generados por el usuario al utilizar el teclado y ratón.

5 Postcondiciones

Inicio del videojuego.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de Caso de Uso:

Modo de ejecución en red

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
15/03/2011	1.0	Elaboración del documento de especificación de caso de uso: En red.	Wilmer Carrera

Tabla de Contenidos

Especificación de Caso de Uso:

Modo de ejecución en red

1 Modo de ejecución en red

1.1 Descripción breve

El proyecto desarrollado comprende 2 versiones del mismo, que parten del mismo motor de videojuegos desarrollado, para un solo jugador y multijugador.

Se puede observar que se ha divido el proyecto en 2 versiones independientes, es decir 2 aplicaciones, la razón por la cual no se unió las versiones es por el tamaño de la aplicaciones y jugabilidad de cada uno de ellos.

En esta versión de la aplicación, el usuario tiene 2 posibilidades escoger la aplicación para ser el servidor de videojuegos, y la segunda como usuario de una sesión de red ya existente.

2 Flujo de Eventos

2.1 Flujo básico

Al iniciar la aplicación y luego de iniciar los componentes requeridos el usuario será capaza de seleccionar las opciones que más le convenga.

Opciones de servidor

Opción buscar sesión

En la opción de servidor de videojuegos, este usuario será quien establezca la sesión de red para que los usuarios de la red se unan a la sesión previamente creada por el anfitrión de la red.

Los jugadores que serán usuario de la red deben seleccionar la sesión y posteriormente unirse a ella.

2.2 Flujo alternativo

2.2.1 Ingreso directo a al videojuego

En todos los videojuegos vistos del mercado, al probarlos, inician con una pantalla de introducción y a continuación muestra una pantalla de opciones que en algunos casos se puede configurar opciones de la aplicación y opciones de jugabilidad, por cuestiones de estética este flujo alternativo no es lo más recomendable.

3 Requerimientos especiales

El computador debe tener instalado una tarjeta de video con soporte de pixel y vertex shader 3.0 como mínimo.

Microsoft Windows XP o posterior.

XNA 3.1 en su versión distribuible.

.Net Framework 3.5

4 Precondiciones

Iniciar el motor de videojuegos.

Iniciar el administrador de pantallas

Iniciar el administrador de sonido

Iniciar el administrador de entrada de datos ya sea por teclado o con el ratón.

4.1 Iniciar el motor de videojuegos

Como núcleo del sistema, es obligatorio iniciar el motor de videojuegos que será responsable de comunicar los diferentes componentes entre sí, según sea necesario.

4.2 Iniciar el administrador de pantallas

Lo primero que observara el jugador al iniciar la aplicación será la pantalla de opciones de juego, y será la primera vez donde el jugador interactuara directamente con la aplicación.

4.3 Iniciar el administrador de sonido

El sonido como parte del ambiente del videojuego también será iniciado al principio de la aplicación.

4.4 Iniciar el administrador de entrada de datos ya sea por teclado o con el ratón.

Para la comunicación con el usuario se inicia el administrador de entrada de datos que será el responsable de capturar los eventos generados por el usuario al utilizar el teclado y ratón.

5 Postcondiciones

Inicio del videojuego.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de Caso de Uso:

Ejecución del videojuego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
15/03/2011	1.0	Elaboración del documento de especificación de caso de uso: Ejecución del videojuego.	Wilmer Carrera

Tabla de Contenidos

Especificación de Caso de Uso:

Ejecución del videojuego

1 Ejecución del videojuego

1.1 Descripción breve

Luego de configurar las opciones del juego, el usuario de la aplicación podrá comenzar el videojuego, donde se desarrolla toda la parte lógica de la jugabilidad del mismo.

En un marco global, en este caso de uso, se desarrolla toda la lógica de videojuego y donde el motor desarrollado utiliza la mayor parte de los componentes gráficos y de comportamiento lógico que lo componen.

2 Flujo de Eventos

2.1 Flujo básico

Según la jugabilidad de cada versión el usuario podrá utilizar la aplicación.

Pantalla de ejecución del videojuego, versión para un solo jugador.

Pantalla de ejecución del videojuego, versión en red.

De acuerdo a las versiones desarrolladas, la ejecución del videojuego es diferente en cada una de estas, a lo largo de la descripción del desarrollo del sistema se explicará más a fondo de la funcionalidad de cada una de estas.

La primera versión, para un solo jugador, la principal diferencia es la aplicación de inteligencia artificial en objetos que el jugador no puede controlar, esto en términos de jugabilidad, más no de funcionalidad.

La segunda versión, con soporte multijugador presenta soporte para red de computadores, pero difiere de la primera, en exactamente la característica de la misma, la aplicación de inteligencia artificial.

Ya que las dos versiones nacen del mismo motor en un futuro se planea combinar las características de uno en el otro, para su complementación, y mejorar así su calidad, tanto en los gráficos como también en otras características propias de productos de software de este tipo.

2.2 Flujo alternativo

N/A

3 Requerimientos especiales

El computador debe tener instalado una tarjeta de video con soporte de pixel y vertex shader 3.0 como mínimo.

Microsoft Windows XP o posterior.

XNA 3.1 en su versión distribuible.

.Net Framework 3.5

Puerto FastEthernet o GigabitEthernet

Switch con puertos suficientes para armar una red de computadores local.

4 Precondiciones

En la versión de un solo jugador:

Iniciar los objetivos de la partida.

Iniciar los gráficos del ambiente según la partida en desarrollo.

Crear los objetos de juego.

Cargar el motor de física.

Cargar el sistema de partículas.

Cargar el motor de inteligencia artificial.

Iniciar el sonido.

Iniciar el administrador de entrada de datos.

Iniciar el sistema de animación de modelos 3D.

En la versión en red:

Crear la sesión de red.

Iniciar los gráficos del ambiente según la partida en desarrollo.

Crear los objetos de juego.

Cargar el motor de física.

Cargar el sistema de partículas.

Iniciar el sonido.

Iniciar el administrador de entrada de datos.

Iniciar el sistema de animación de modelos 3D.

Iniciar el administrador de la red.

4.1 Iniciar los objetivos de la partida

Cada vez que se inicia el videojuego se indica el criterio para ganar la partida.

4.2 Iniciar los gráficos del ambiente según la partida en desarrollo

La aplicación debe esperar a que los gráficos sean iniciados, dependiendo de la cantidad de estos, dependerá el tiempo de espera, además de la velocidad de procesamiento de computador donde se instalará la aplicación.

4.3 Crear los objetos de juego.

Los objetos dependiendo de cómo interactuara con el ambiente de juego, serán cargados y se posicionaran en el escenario del videojuego.

4.4 Cargar el motor de física.

El motor de física se aplicara como mallas de colisión en los objetos seleccionados para el mismo, el jugador también será un objeto físico.

4.5 Cargar el sistema de partículas

Para los efectos especiales se lo hará a través de un sistema de partículas con plantillas previamente configuradas, y que serán renderizadas según sea necesario.

4.6 Cargar el motor de inteligencia artificial.

Solo la versión de un solo jugador estará habilitado, los objetos que no sean manipulados por el jugador tendrán esta funcionalidad, más adelante se explicara cómo se afrontó este complejo tema.

4.7 Iniciar el sonido.

Para mejor presentación para el jugador cada escenario estará ambientado por sonido, para dar mayor realce a la aplicación.

4.8 Iniciar el administrador de entrada de datos.

El jugador podrá manipular su personaje en el videojuego a través del teclado, el administrador de entrada de datos será el responsable de traducir esas acciones a acciones de juego que serán representadas en el videojuego.

4.9 Iniciar el sistema de animación de modelos 3D.

Los objetos que presentan animación serán administrados por este componente. La reproducción en bucle, terminación, pausa e inicio de las animaciones se desarrollan en este módulo.

4.10 Crear la sesión de red.

Solo el usuario seleccionado podrá ser el servidor del videojuego, quien será el responsable de configurar las variables del mismo e iniciar la sesión.

5 Postcondiciones

Inicio del videojuego

Unión a la sesión de red existente.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de Caso de Uso:

Criterio para ganar la partida

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
17/03/2011	1.0	Elaboración del documento de especificación de caso de uso: Criterio para ganar la partida.	Wilmer Carrera

Tabla de Contenidos

Especificación de Caso de Uso:

Criterio para ganar la partida

1 Criterio para ganar la partida

1.1 Descripción breve

Los videojuegos tienen como objetivo, brindar entretenimiento al jugador, dependiendo del tipo de videojuego, el criterio para ganar el videojuego es lo que mantiene al jugador interesado en probar la aplicación, el cual el usuario deberá cumplir.

En las versiones desarrolladas, el criterio para ganar es diferente en cada uno de ellos, en el primero, con opciones para un solo jugador, para ganar la partida se debe cumplir con los objetivos específicos de cada partida, en la segunda versión, los parámetros para ganar serán configurados por el servidor de juegos, todos los demás usuarios de la red deben tratar de cumplir ese objetivo para ganar.

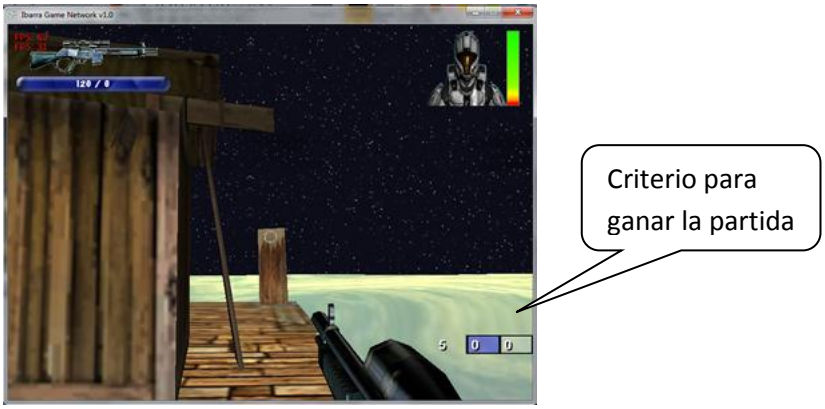
Todos los videojuegos comerciales tienen este marco de lógica.

2 Flujo de Eventos

2.1 Flujo básico

En la primera versión, cuando el jugador cumple con el objetivo a cumplir, ganar la partida y de acuerdo al nivel o escenario en el que esté ganara el videojuego o será promovido al siguiente nivel.

La segunda versión, con soporte multijugador, el servidor de juegos tendrá la responsabilidad de señalar al ganador de la partida e informar a los demás jugadores quien será el vencedor



2.2 Flujo alternativo

N/A

3 Requerimientos especiales

El computador debe tener instalado una tarjeta de video con soporte de pixel y vertex shader 3.0 como mínimo.

Microsoft Windows XP o posterior.

XNA 3.1 en su versión distribuible.

.Net Framework 3.5

Puerto FastEthernet o GigabitEthernet

Switch con puertos suficientes para armar una red de computadores local.

4 Precondiciones

Iniciar el escenario.

Indicar los objetivos de la partida.

4.1 Iniciar el escenario

Cargar los objetos del escenario donde se desenvolverá la lógica del videojuego.

4.2 Iniciar los objetivos de la partida

Presentar de manera gráfica cuales son los objetivos de la partida, para que el usuario los pueda cumplir, en la versión multijugador, para ganar la partida, el jugador local debe competir con los demás usuarios de la red por la victoria.

5 Postcondiciones

Inicio del videojuego

Unión a la sesión de red existente.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de Caso de Uso:

Pérdida del videojuego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
17/03/2011	1.0	Elaboración del documento de especificación de caso de uso: Pérdida del videojuego.	Wilmer Carrera

Tabla de Contenidos

Especificación de Caso de Uso:

Pérdida del videojuego

1 Pérdida del videojuego

1.1 Descripción breve

En un videojuego como se puede ganar el mismo, también el jugador puede perder, en la mayoría de productos comerciales de este tipo presentan niveles de dificultad, ya sea en la manera de implementar inteligencia artificial en los objetos del videojuego, incrementando la velocidad de ejecución del mismo.

En la versión para un solo jugador, para perder, el usuario debe perder todos los puntos de vida, que inicialmente tiene, para perder puntos de vida el jugador debe recibir daño de los objetos que son los “enemigos” del jugador. El presente proyecto no tiene niveles de dificultad que el usuario pueda seleccionar, la dificultad radica en los niveles de juego que el usuario debe atravesar antes de ganar el videojuego.

Para la versión multijugador, el jugador local debe recibir daño de los demás jugadores que en este caso serían, los demás usuario de la red, hasta que los puntos de vida lleguen a cero.

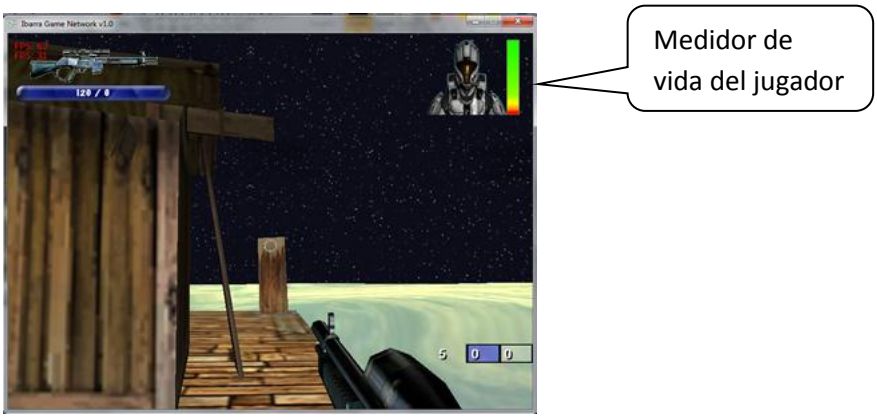
Todos los videojuegos comerciales tienen este marco de lógica.

2 Flujo de Eventos

2.1 Flujo básico

En la primera versión, cuando el jugador cumple con el objetivo a cumplir, ganar la partida y de acuerdo al nivel o escenario en el que esté ganara el videojuego o será promovido al siguiente nivel.

La segunda versión, con soporte multijugador, el servidor de juegos tendrá la responsabilidad de señalar al ganador de la partida e informar a los demás jugadores quien será el vencedor.



2.2 Flujo alternativo

N/A

3 Requerimientos especiales

El computador debe tener instalado una tarjeta de video con soporte de pixel y vertex shader 3.0 como mínimo.

Microsoft Windows XP o posterior.

XNA 3.1 en su versión distribuible.

.Net Framework 3.5

Puerto FastEthernet o GigabitEthernet

Switch con puertos suficientes para armar una red de computadores local.

4 Precondiciones

Iniciar el jugador.

4.1 Iniciar el jugador

Al iniciar el jugador los puntos de vida están completos.

5 Postcondiciones

Inicio del videojuego

Unión a la sesión de red existente.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificación de Caso de Uso:

Salir del videojuego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
17/03/2011	1.0	Elaboración del documento de especificación de caso de uso: Salir del videojuego.	Wilmer Carrera

Tabla de Contenidos

Especificación de Caso de Uso:

Salir del videojuego

1 Salir del videojuego

1.1 Descripción breve

Como opción en el desarrollo del videojuego, al presionar la tecla de salida de la aplicación, se mostrara un menú en el cual, se podrá seleccionar la opción de salir de la aplicación

En la versión para un solo jugador, se perderán los datos que no hayan sido guardados en el transcurso de la partida.

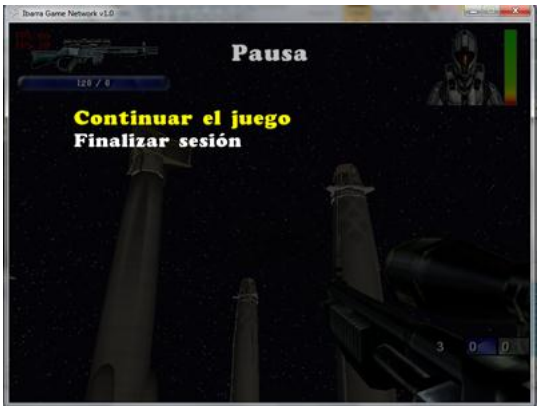
Para la versión multijugador, dependiendo de quien salga de la sesión afectara el videojuego, es decir si un usuario de la red sale, no afectara la el desarrollo normal del videojuego, si encaso el jugador que sale es el primero en las estadísticas de mejor puntaje del videojuego, este puntaje se eliminara de las entradas de las estadísticas y el segundo mejor tomara su lugar. Pero si quien decide abandonar la partida es el servidor del juego, la sesión se perderá y los demás usuarios serán expulsados automáticamente de la sesión.

2 Flujo de Eventos

2.1 Flujo básico

En la primera versión, al presionar la tecla de salida muestra un menú de selección, con opción para salir del videojuego.

La segunda versión, dependiendo de si es no el servidor se mostrará un menú con la opción se salir del videojuego.



2.2 Flujo alternativo

N/A

3 Requerimientos especiales

El computador debe tener instalado una tarjeta de video con soporte de pixel y vertex shader 3.0 como mínimo.

Microsoft Windows XP o posterior.

XNA 3.1 en su versión distribuible.

.Net Framework 3.5

Puerto FastEthernet o GigabitEthernet

Switch con puertos suficientes para armar una red de computadores local.

4 Precondiciones

Iniciar el juego.

4.1 Iniciar el juego

Una vez iniciado el videojuego, el jugador será quien decida si desea salir o reanudar el juego.

5 Postcondiciones

Inicio del videojuego

Unión a la sesión de red existente.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Especificaciones adicionales

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
10/03/2011	1.0	Elaboración del documento de especificaciones adicionales	Wilmer Carrera

Tabla de Contenidos

Especificaciones adicionales

1 Introducción

1.1 Propósito

Especificar las los casos de uso de los requerimientos no funcionales para el proyecto a desarrollarse.

1.2 Alcance

El presente documento será aplicable al proyecto a desarrollarse como tesis previa la obtención del título de Ingeniero en Sistema Computacionales.

1.3 Descripción

Este documento explica la funcionalidad, usabilidad, fiabilidad, compatibilidad del sistema a desarrollarse, como se puede observar, son especificaciones no funcionales, que el sistema debe cumplir.

2 Funcionalidad.

2.1 Entretenimiento

La principal funcionalidad de sistemas de este tipo es proporcionar entretenimiento a los usuarios que prueban la aplicación, dependiendo de esto, el usuario decidirá si desea o no seguir utilizando la aplicación.

3 Usabilidad

3.1 Dirigido a cualquier persona

Cualquier persona, con interés de probar la aplicación podrá hacerlo, independientemente de la edad, género, etc. Lo que hace de este producto no tenga restricciones de uso por parte de los usuarios.

3.2 Intuitivo y agradable

El sistema a desarrollarse debe estar enfocado a usuarios inexpertos en el uso de software de este tipo. Las opciones de juego deben ser claras y no deben confundir al usuario.

4 Fiabilidad

4.1 Cantidad mínima de errores durante la ejecución de la aplicación

Los errores durante el desarrollo deben ser corregidos, y los reportados en la fase de pruebas de la aplicación, para luego lanzar la versión final que sería la versión de producción.

5 Rendimiento

5.1 Frames de actualización por segundo

El número de frames que deben ser actualizados por segundo no debería ser menor los 20fps para su normal funcionamiento, esto depende de los recursos de disco y memoria que utiliza la aplicación, si este número considerado normal baja, de lo especificado puede provocar que la aplicación presente parpadeos en la pantalla, o cualquier otro.

5.2 Conexiones al servidor de juegos

El número máximo de conexiones soportadas por el servidor y como restricción por parte de la aplicación es de 16 conexiones simultaneas, aunque este número se puede extender, pero por rendimiento se limitara a este número, para que pueda soportar más conexiones se debe pensar en servidores dedicados para este propósito.

6 Compatibilidad

6.1 Descripción de compatibilidad

El sistema desarrollado solo podrá ser probado sobre sistemas Operativos Windows a partir de la versión XP, pero lo más importante es cumplir con los requisitos para el soporte de gráficos tridimensionales y efectos que se aplican ya que son los que más recursos consumen.

El PC donde se probara la aplicación debe tener como mínimo una tarjeta de video con soporte de pixel y vertex shader 3.0 para su correcta ejecución, restricción que en la actualidad no presenta mayor complicación.

7 Ayuda de los requisitos del sistema

Puede ser visto en la página de información de los requisitos para juegos hechos con XNA.

8 Interfaces

8.1 Interfaces de usuario

Pantallas de acceso a la aplicación

Opciones de configuración fáciles de utilizar.

8.2 Interfaces de hardware

Conexión a la red a través cables UTP hacia un switch.

8.3 Interfaces de software

El motor del videojuego es el responsable de la comunicación entre los módulos y componentes que la aplicación necesita.

8.4 Interfaces de comunicación

El motor de videojuegos será el responsable de la administración de información que fluye a través de la red.

9 Requisitos de licenciamiento

N/A

10 Legalidad, Copyright, y Otros avisos

El presente proyecto es desarrollado enteramente por el autor del proyecto, y que será entregada posteriormente a la Universidad Técnica del Norte, para la Facultad de Ingeniería en Ciencias Aplicadas, de la Carrera de Ingeniería en Sistemas Computacionales.

Los modelos 3D utilizados fueron descargados de sitios web que los distribuyen de manera libre, cabe señalar que la animación y edición de estos modelos también fueron realizados por el autor del proyecto.

El framework utilizado y demás software tienen sus propias disposiciones de legalidad y derechos de autor.

11 Estándares aplicables.

N/A.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Realización de Caso de Uso:

Opciones de juego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
19/03/2011	1.0	Elaboración del documento de la realización del caso de uso: Opciones de juego.	Wilmer Carrera

Tabla de Contenidos

Realización de Caso de Uso

Opciones de juego

1 Introducción

1.1 Propósito

Definir como se realizó la implementación de las opciones de juego en el presente proyecto.

1.2 Alcance

El presente documento, específica que componentes del motor de videojuegos desarrollado utiliza, así como también la funcionalidad de los mismos.

1.3 Definiciones, Siglas, y Abreviaciones

Ver glosario.

1.4 Referencias

Especificación de casos de uso: Opciones de juego.

1.5 Descripción

Mostrar el flujo de componentes utilizados a raíz de la inicialización del motor de videojuegos construido y la integración del mismo en las diferentes versiones que fueron desarrolladas.

2 Flujo de eventos - Diseño

Para mostrar las opciones de juego el primer componente que se inicia del motor de videojuegos es el administrador de pantallas, que es el responsable de renderizar la pantalla según el flujo lógico de la aplicación y presentarla al usuario.

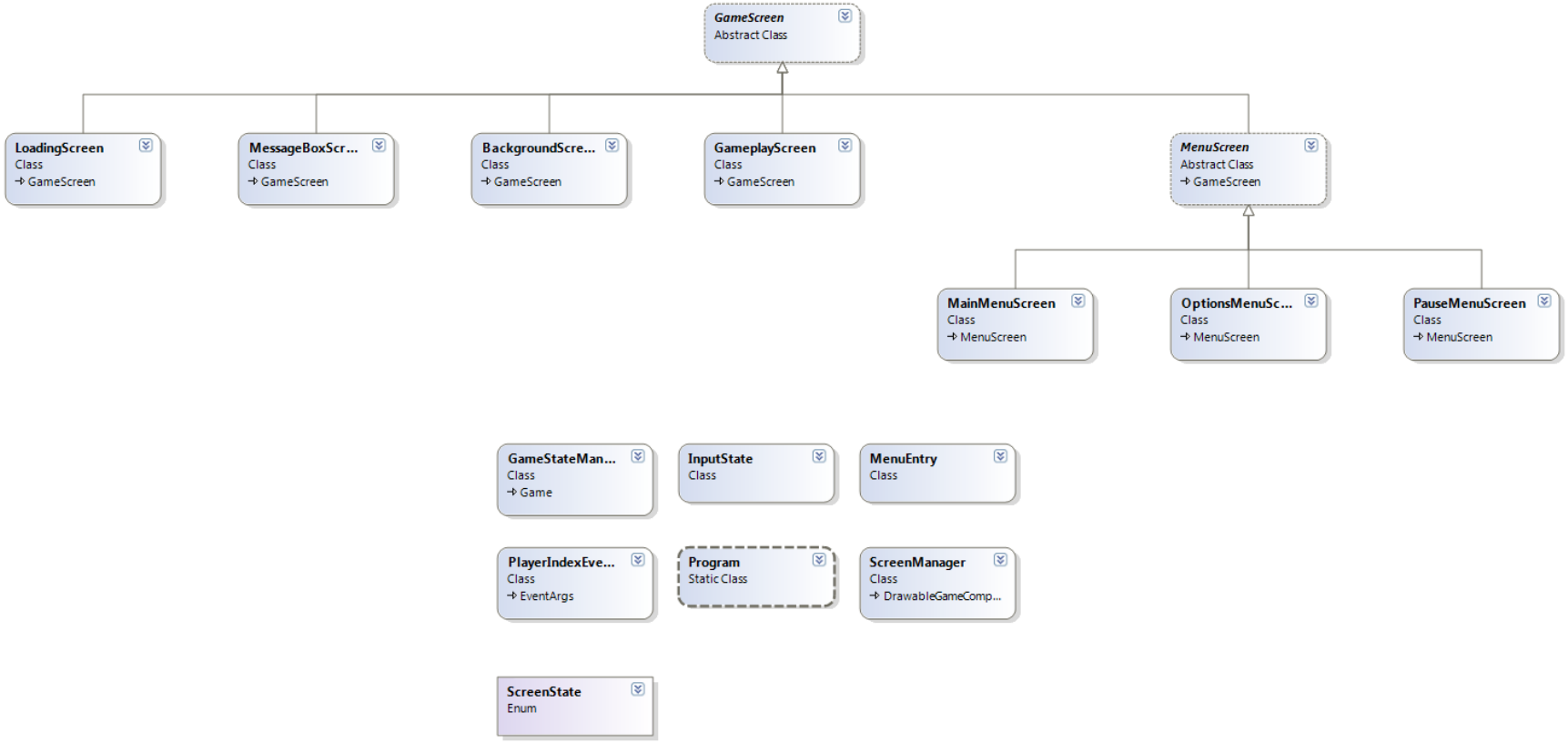


Diagrama de clases del componente de administración de pantallas.

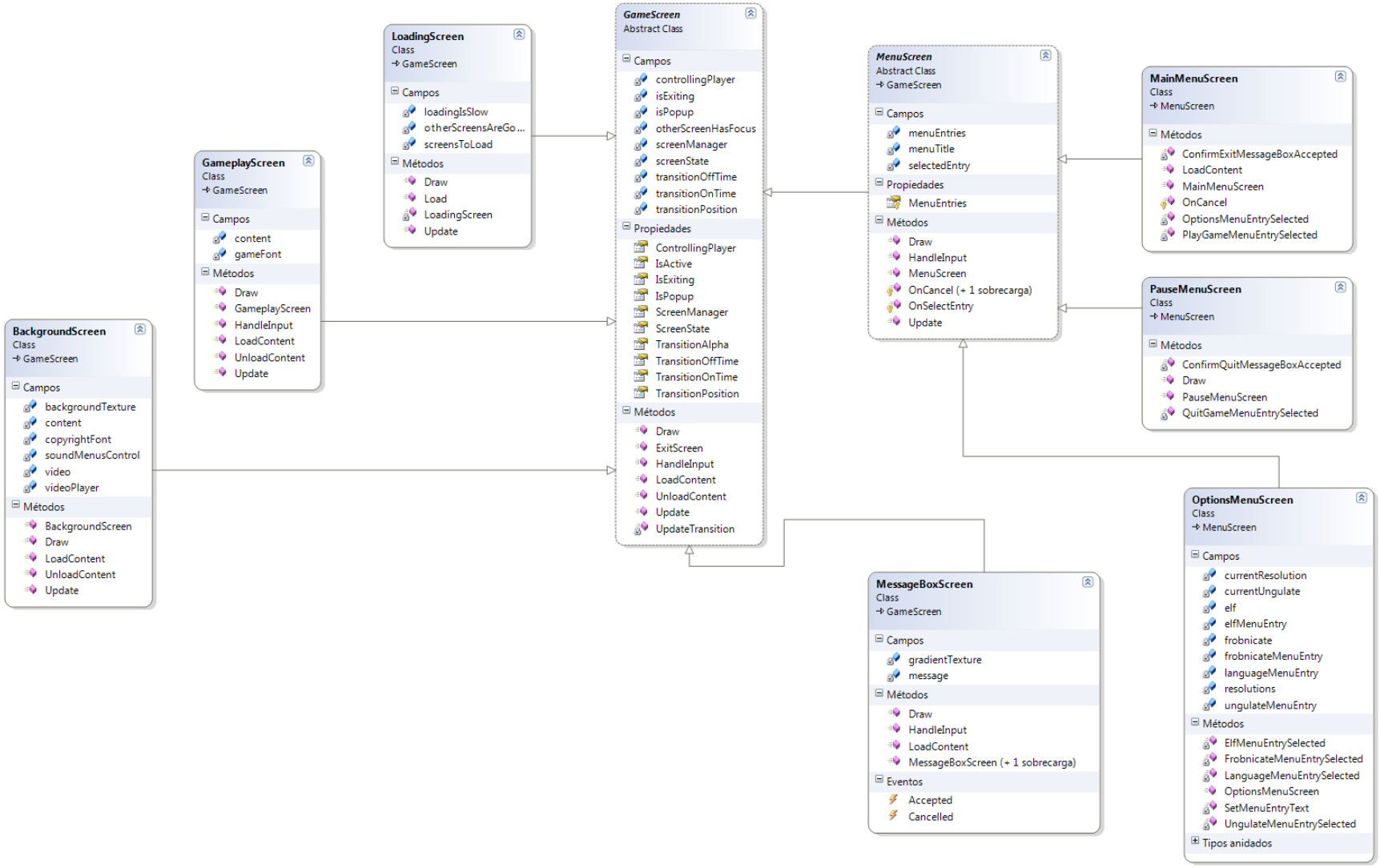
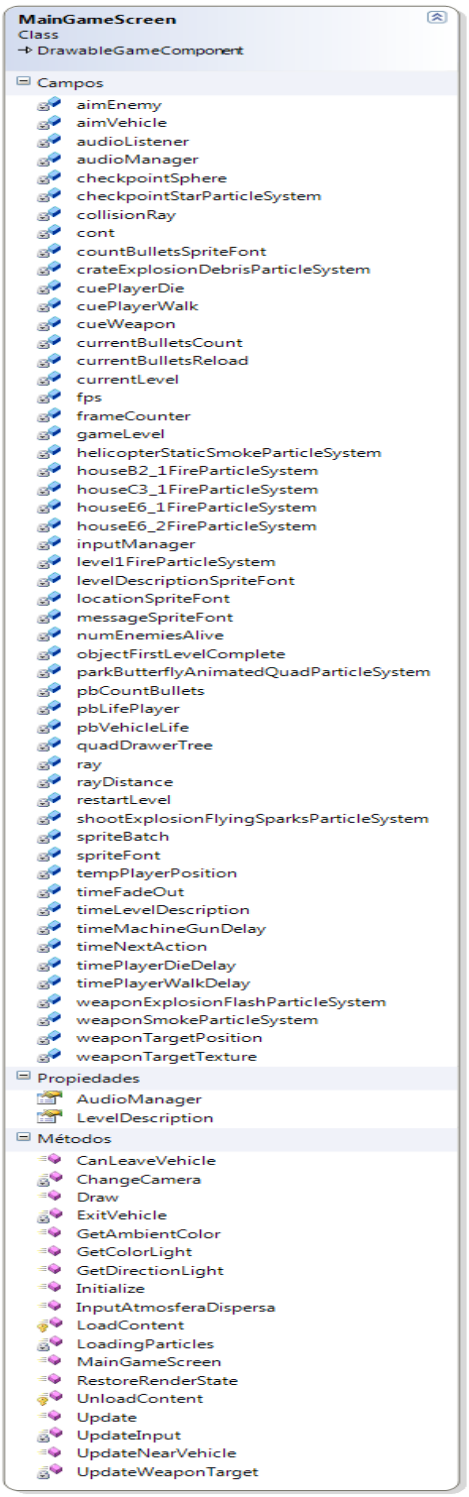


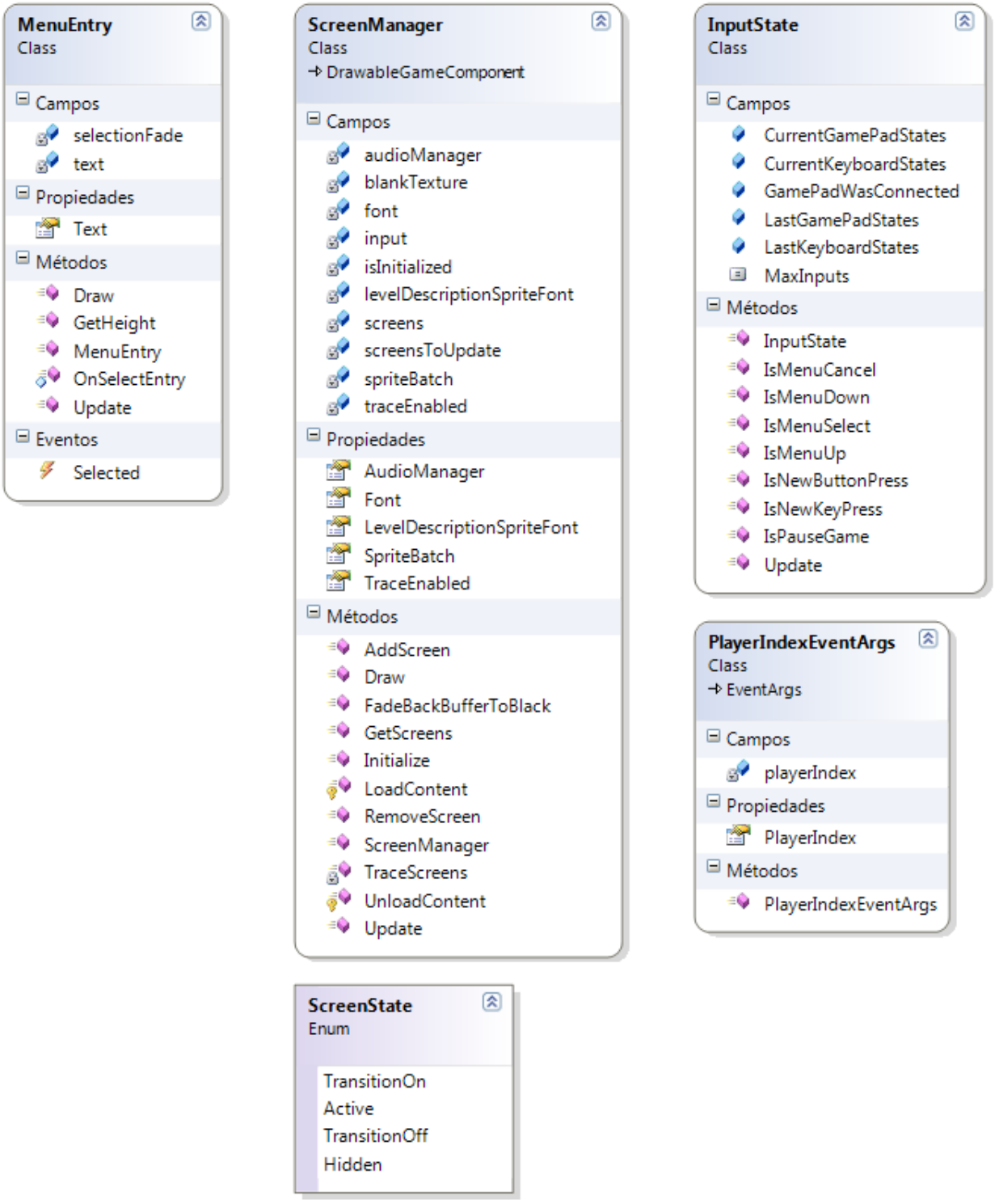
Diagrama de diseño

Diagrama de clase MainGameScreen



Representa la clase principal del videojuego, aunque no tiene nada que ver con las opciones de juego, pero según el flujo de eventos, esta pantalla es inicializada, luego de seleccionar jugar en el menú de opciones.

Diagramas de clase



La clase ScreenManager, es la encargada de administrar las pantallas.

Nótese que todas las clases, variables, métodos están en el idioma inglés, esto es por convención por parte del desarrollador, ya que durante el desarrollo del mismo, el poner

nombre de variables en español resultaba nombres demasiado largos, por ejemplo, con la variable en español string tiempodeactualizaciondevehiculo en contraste con su equivalente en inglés string vehicleupdatetime como se puede ver los artículos son eliminados y como resultado variables más cortas, y como última razón los métodos de actualización y dibujo, los proporciona el framework DRAW y UPDATE, siguiendo este esquema, en el proyecto también se decidió utilizar nombres de métodos en inglés.

3 Requerimientos derivados

Interfaces intuitivas para el usuario.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Realización Caso de Uso:

Para un solo jugador

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
19/03/2011	1.0	Elaboración del documento de la realización del caso de uso: Para un solo jugador.	Wilmer Carrera

Tabla de Contenidos

Realización Caso de Uso:

Para un solo jugador

1 Introducción

1.1 Propósito

Definir como se realizó la implementación de las opciones de juego para un solo jugador en el presente proyecto.

1.2 Alcance

El presente documento, específica que componentes del motor de videojuegos desarrollado utiliza, así como también la funcionalidad de los mismos.

1.3 Definiciones, Siglas, y Abreviaciones

Ver glosario.

1.4 Referencias

Especificación de casos de uso: Para un solo jugador.

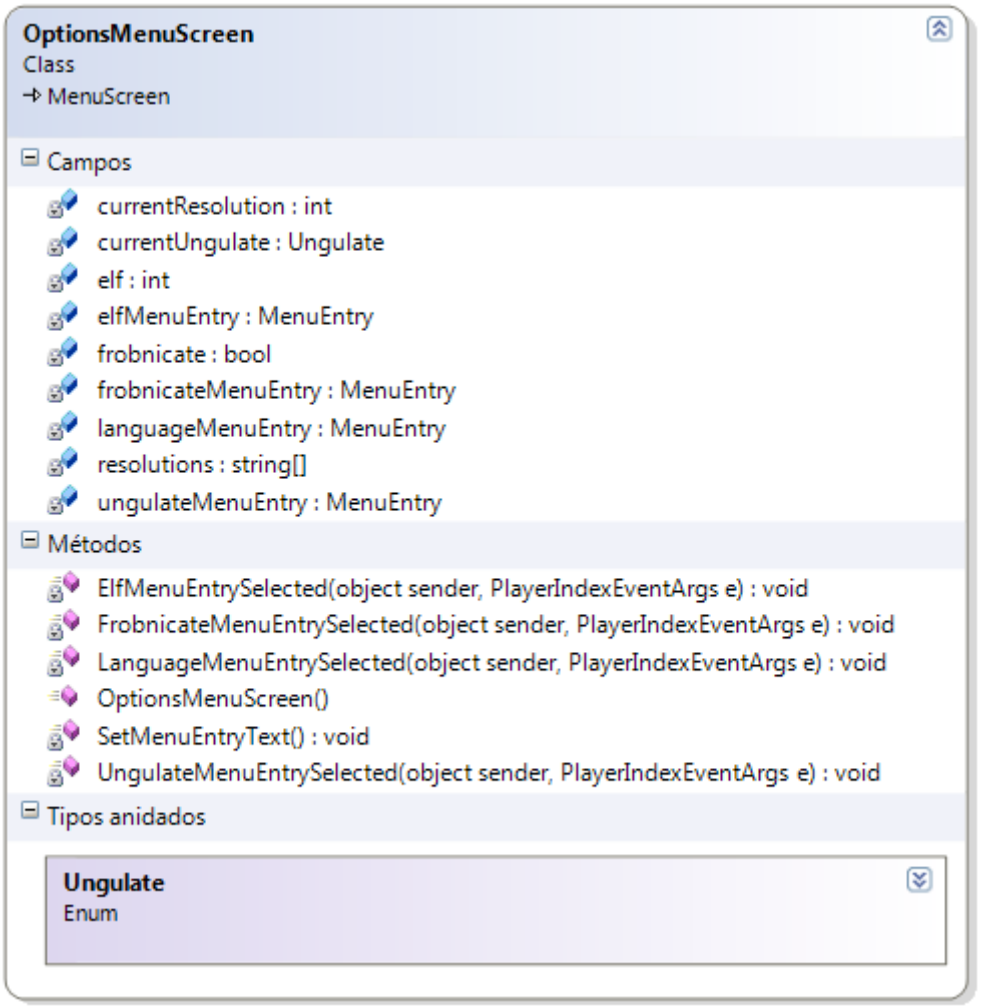
1.5 Descripción

Mostrar el flujo de componentes utilizados a raíz de la inicialización del motor de videojuegos construido y la integración del mismo en las diferentes versiones que fueron desarrolladas.

2 Flujo de eventos - Diseño

Para mostrar las opciones de juego para un solo jugador el primer componente que se inicia del motor de videojuegos es el administrador de pantallas, que es el responsable de renderizar la pantalla según el flujo lógico de la aplicación y presentarla al usuario.

Diagrama de clase: OptionMenuScreen



No representa ninguna funcionalidad pero se lo implemento por estética en aplicaciones de este tipo.

3 Requerimientos derivados

Interfaces intuitivas para el usuario.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Realización Caso de Uso:

Modo de ejecución en red

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
19/03/2011	1.0	Elaboración del documento de la realización del caso de uso: En red.	Wilmer Carrera

Tabla de Contenidos

Realización Caso de Uso:

Modo de ejecución en red

1 Introducción

1.1 Propósito

Definir como se realizó la implementación de las opciones de juego para la versión en red en el presente proyecto.

1.2 Alcance

El presente documento, específica que componentes del motor de videojuegos desarrollado utiliza, así como también la funcionalidad de los mismos.

1.3 Definiciones, Siglas, y Abreviaciones

Ver glosario.

1.4 Referencias

Especificación de casos de uso: Modo de ejecución en red.

1.5 Descripción

Mostrar el flujo de componentes utilizados a raíz de la inicialización del motor de videojuegos construido y la integración del mismo en las diferentes versiones que fueron desarrolladas.

2 Flujo de eventos - Diseño

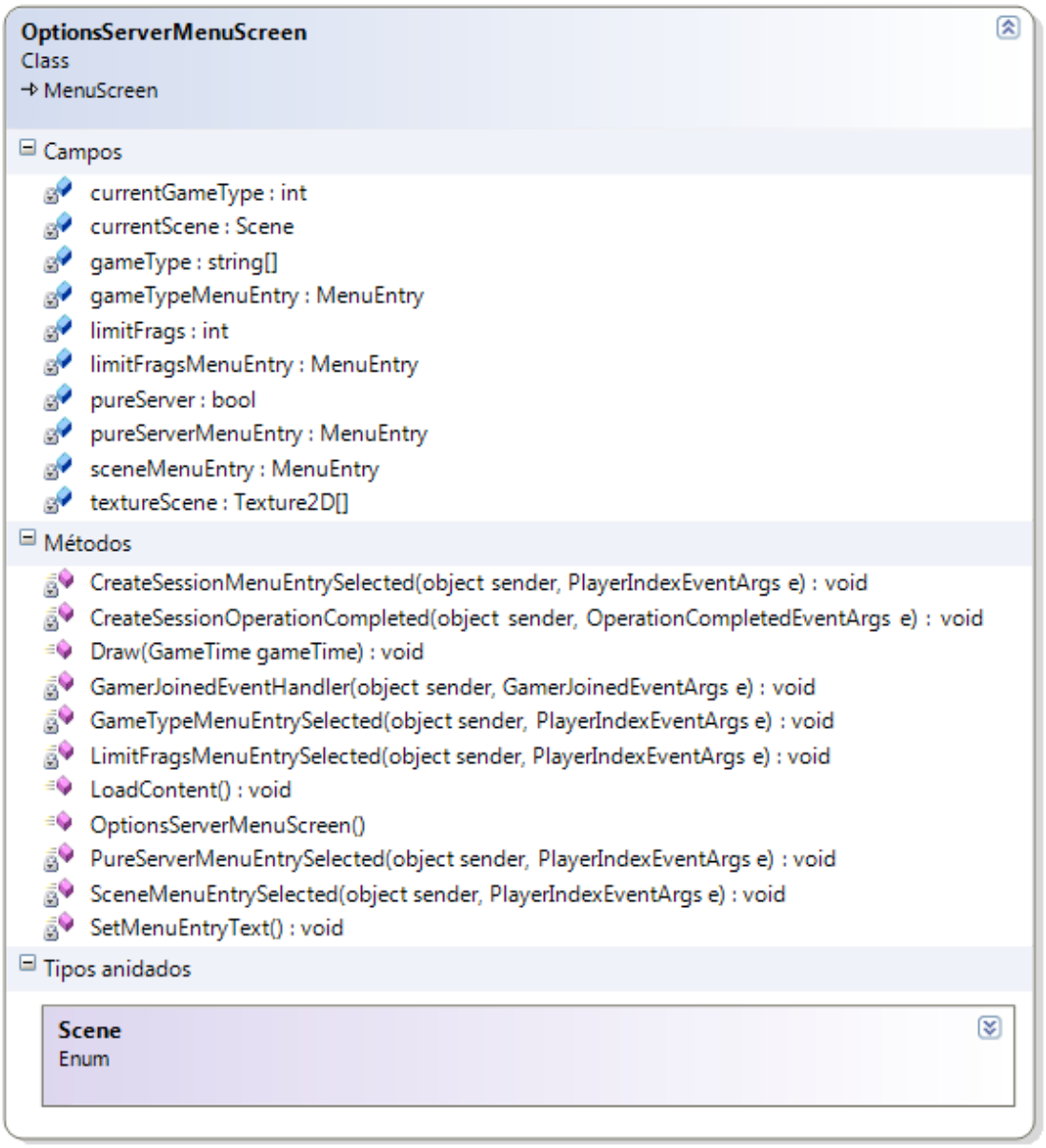
Para mostrar las opciones de juego en red el primer componente que se inicia del motor de videojuegos es el administrador de pantallas, que es el responsable de renderizar la pantalla según el flujo lógico de la aplicación y presentarla al usuario.

Las opciones de juego para la versión en red, difieren para quien será el servidor y los clientes de la aplicación.

El servidor será quien configure las opciones del videojuego, y los clientes a través de las opciones presentadas anteriormente buscaran las sesiones a las que pueden unirse.

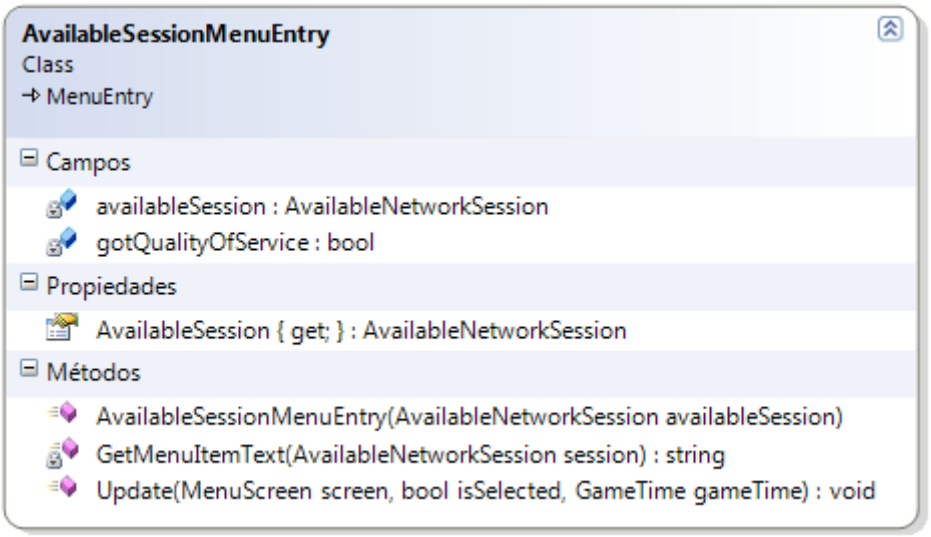
La pantalla común para ambos casos, y que forma parte de la configuración del videojuego es donde el usuario puede seleccionar su personaje que lo representará en la ejecución del videojuego.

Diagrama de clase OptionsServerMenuScreen, opciones de configuración del servidor de videojuegos.



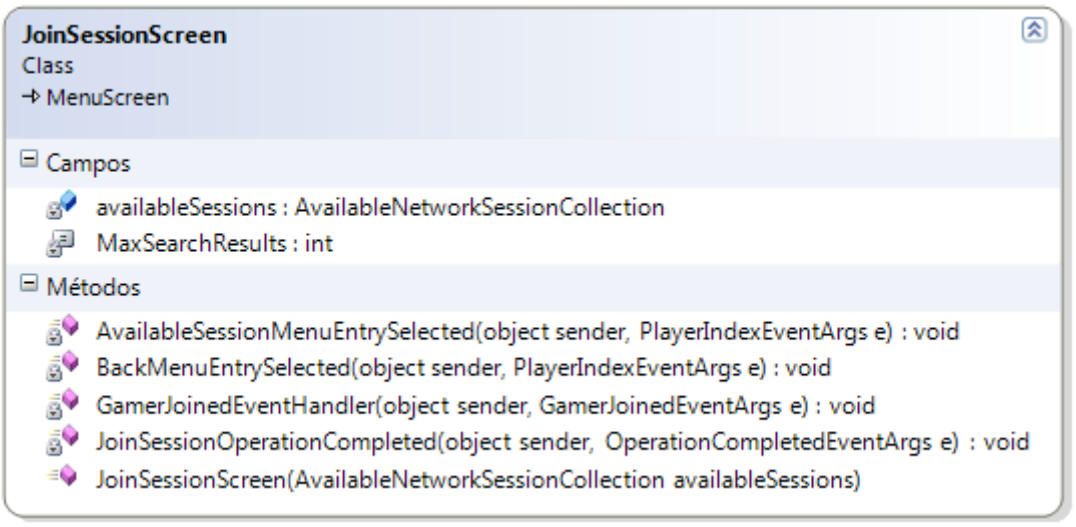
Las opciones para los clientes de la red está dado por el administrador de red, parte del motor de videojuegos extendido de la versión preliminar para un solo jugador.

Diagrama de clase AvailableSessionMenuEntry



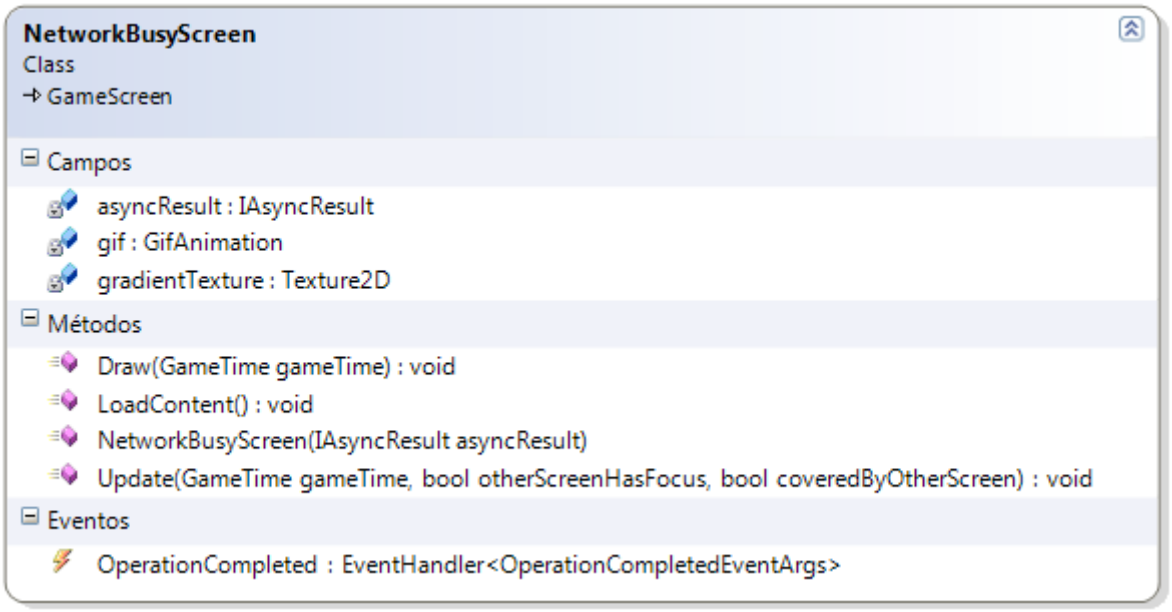
Muestra las sesiones validas a las que el usuario puede unirse y empezar a competir con otros usuarios para ganar el videojuego.

Diagrama de clase JoinSessionScreen



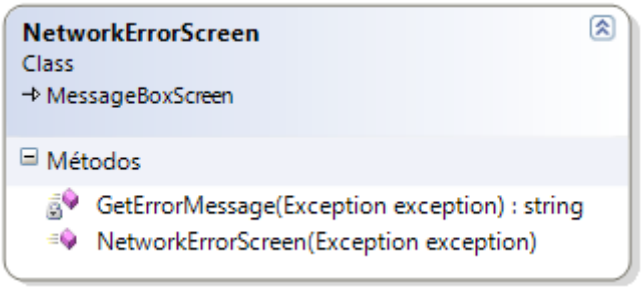
Captura los eventos de la red para operaciones necesarias antes de la ejecución del videojuego.

Diagrama de clase NetworkBusyScreen



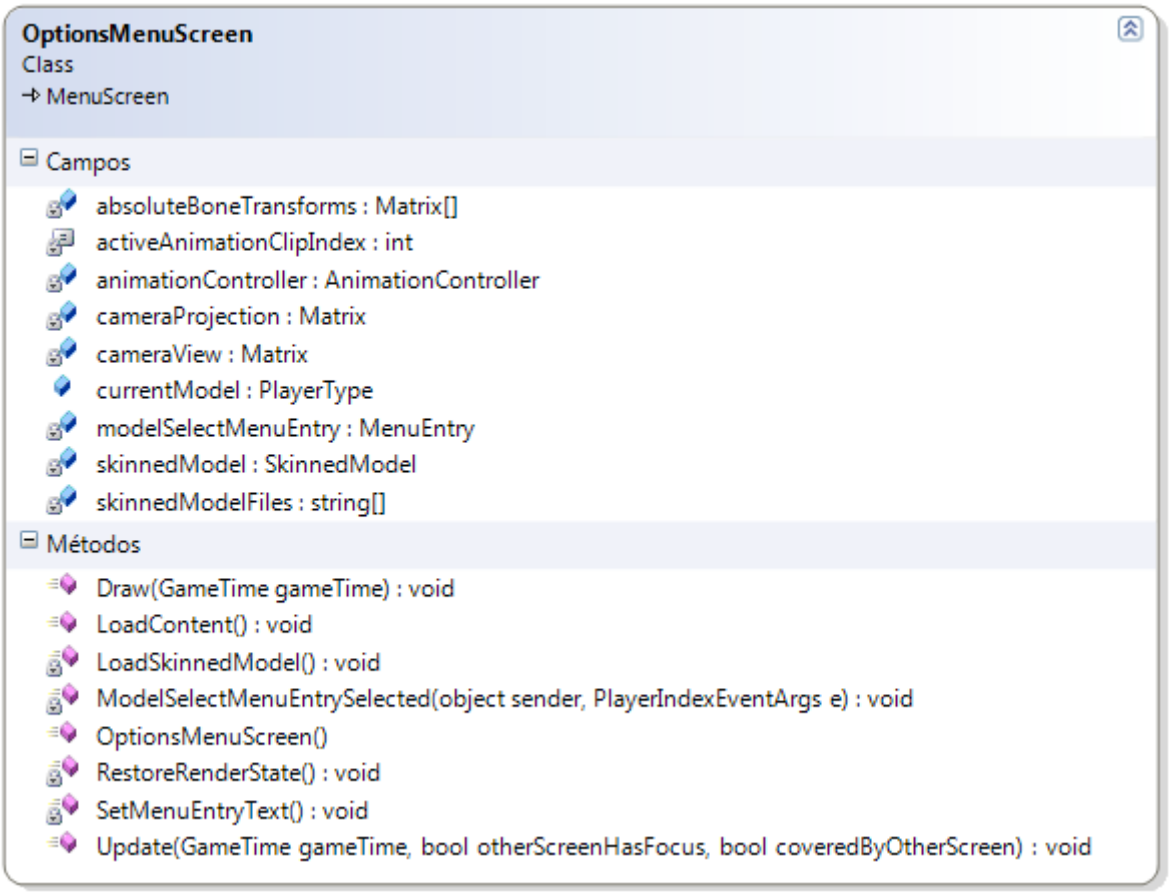
Pantalla de notificación para el usuario de las operaciones de red por ejemplo: esta pantalla se activará para mostrar el progreso cuando el usuario se una a alguna sesión de red válida.

Diagrama de clase NetworkErrorScrren



Muestra una pantalla de error en el caso de falla en la red producida por la misma.

Diagrama de clase OptionsMenuSrceen



Pantalla de opciones común para el servidor y clientes de la aplicación.

3 Requerimientos derivados

Interfaces intuitivas para el usuario.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Realización Caso de Uso:

Ejecución del videojuego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
19/03/2011	1.0	Elaboración del documento de la realización del caso de uso: Ejecución del videojuego.	Wilmer Carrera

Tabla de Contenidos

Realización Caso de Uso:

Ejecución del videojuego

1 Introducción

Definir como se realizó la implementación de la ejecución del videojuego en el presente proyecto, representa el caso de uso funcional más importante de todo el proyecto, ya que es aquí donde se desarrolla la razón del proyecto y la ejecución de la mayoría de los componentes del motor de videojuegos.

1.1 Alcance

El presente documento, específica que componentes del motor de videojuegos utiliza, así como también la funcionalidad de los mismos.

1.2 Definiciones, Siglas, y Abreviaciones

Ver glosario.

1.3 Referencias

Especificación de casos de uso: Ejecución del videojuego.

1.4 Descripción

Mostrar el flujo de componentes utilizados a raíz de la inicialización del motor de videojuegos construido y la integración del mismo en las diferentes versiones que fueron desarrolladas.

2 Flujo de eventos - Diseño

Todos los componentes que forman parte del motor de videojuegos, se inician para la ejecución del mismo, y de acuerdo a la lógica de la aplicación, estos componentes serán activados y desactivados.

En esta parte del desarrollo de la aplicación es donde se debe poner mayor énfasis con respecto al rendimiento de la aplicación, ya que con la mayoría de los componentes

ejecutándose, la aplicación podría verse afectada independientemente de las características de hardware de la maquina donde será probada la aplicación.

Según la versión de las aplicaciones desarrolladas, el flujo de eventos puede diferir una de la otra, pero cabe resaltar que las dos versiones desarrolladas nacen del mismo motor de videojuegos desarrollado en un principio.

La mayoría de componentes como se explicó anteriormente en la explicación teórica del desarrollo del sistema, fueron obtenidos de internet y gracias a la comunidad XNA que suben componentes para videojuegos y que cualquier desarrollador los puede implementar libremente, sin restricciones de uso.

Debido a la complejidad de este tipo de sistemas se decidió *"No reinventar la rueda"* si ya hay algo hecho y que funciona es mejor utilizarlo, con esto se debe especificar que no solo por el hecho de tener acceso a componentes que pueden ser utilizados, no se los revisó, al contrario la mayoría de componentes fueron revisado e igualmente fueron modificados para que se comporten como se esperaba. Cambios que serán especificados en el desarrollo de este documento.

Este documento será dividido en secciones los cuales representaran los componentes del motor de videojuegos.

Motor de Física

El motor de física seleccionado de entre los tantos disponibles en internet fue JigLibX, potente y rápido, más las características propias que supero a sus similares, características necesarias para las dos versiones que se estaban creando, entre ellas, simulación de física de vehículos, varias primitivas, posibilidad de detección de colisiones a través de máscaras de colisión con otros objetos físicos, primitivas configurables, máscaras de colisión para personajes, física sobre terrenos creados con mapas de alturas, entre otras.

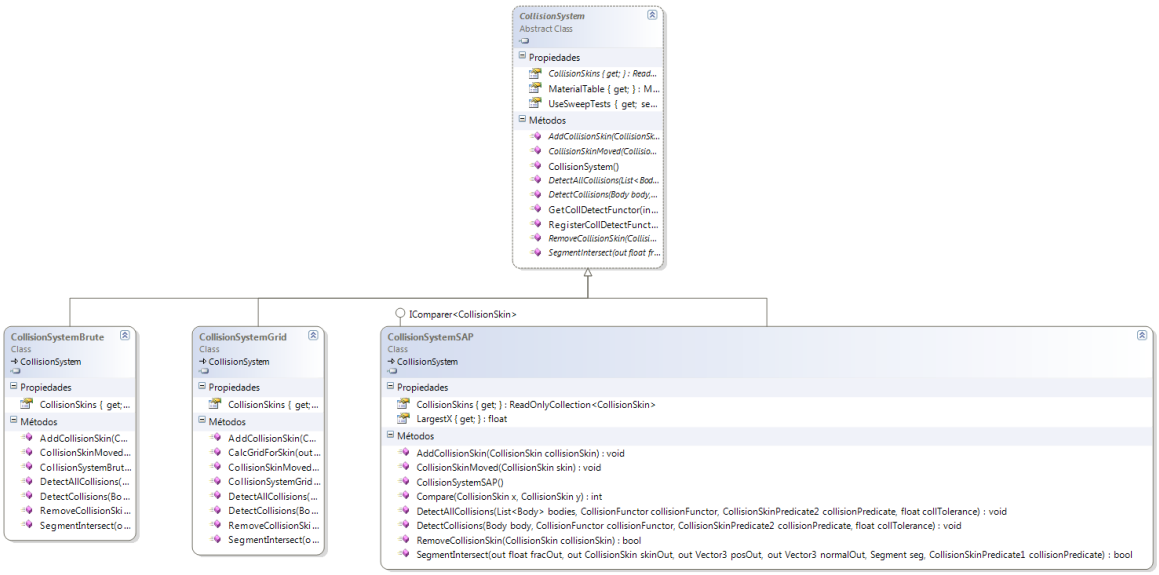
Crear un simulador de física propio para computadoras puede ser de lo más complicado, esto debido principalmente a los cálculos matemáticos que se requiere programar, si bien la matemática propia en algunos casos es difícil, esto se complica más cuando se requiera pasar esos cálculos a algún lenguaje de programación.

Jiglib es de código abierto, es decir se dispone de los códigos fuentes, lo que fue de gran ayuda para su modificación para asegurar su funcionamiento dependiendo de lo que se requería hacer.

Cualquier modelo 3D puede tener comportamiento físico, a excepción de los modelos estáticos que no necesitan simulación física, las máscaras físicas pueden envolver estos modelos y también sirven para detección de colisiones con otros objetos o con rayos de colisión.

Los rayos de colisión son segmentos trazados desde un punto de inicio hasta el objeto con el cual hace colisión, con el cual se determina la distancia y tipo de objeto con el cual choca. En un espacio tridimensional los rayos de colisión tienen un componente adicional que es la profundidad representada por el eje z.

Diagrama de clases JigLibX.Collision



JigLibX está diseñado para soportar tres sistemas de colisión:

CollisionSystemSAP: Implementa un sistema de colisión (prueba de fase general) basado sobre el algoritmo barrer-y-pasar.

CollisionSystemGrid: Implementa un sistema de colisión por división del mundo envolviéndolo en una rejilla con una cierto tamaño de configuración. Si los objetos son eventualmente distribuidos esto reduciría el número de comprobaciones que necesita hacer.

CollisionSystemBrute: Este sistema de colisión quien comprueba cada mascara una contra otra. Para escenas pequeñas esta es más rápida o incluso más rápido que el sistema de colisión Grid.

Diagrama de clases de las funciones de colisión

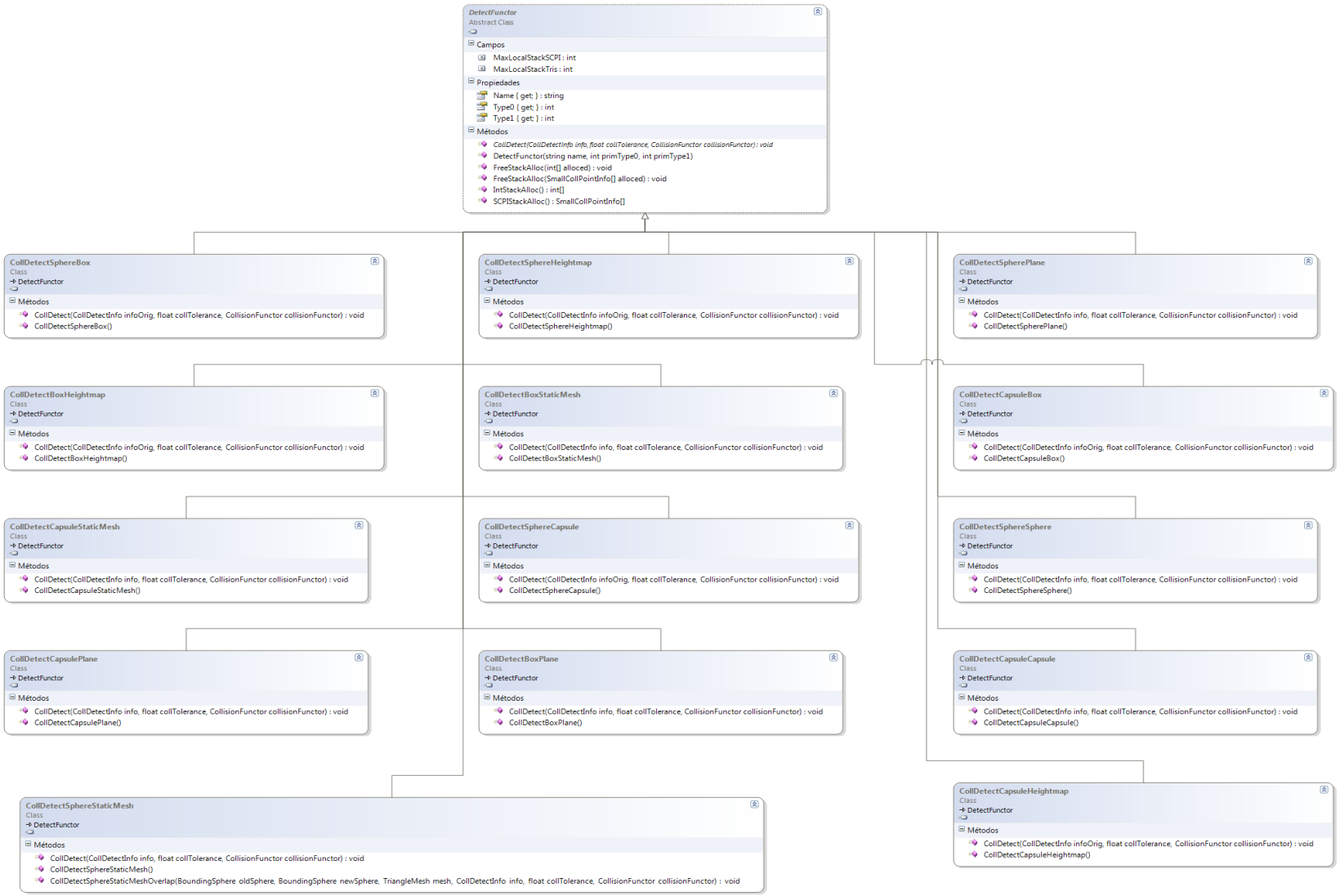


Diagrama de clases de las primitivas de colisión

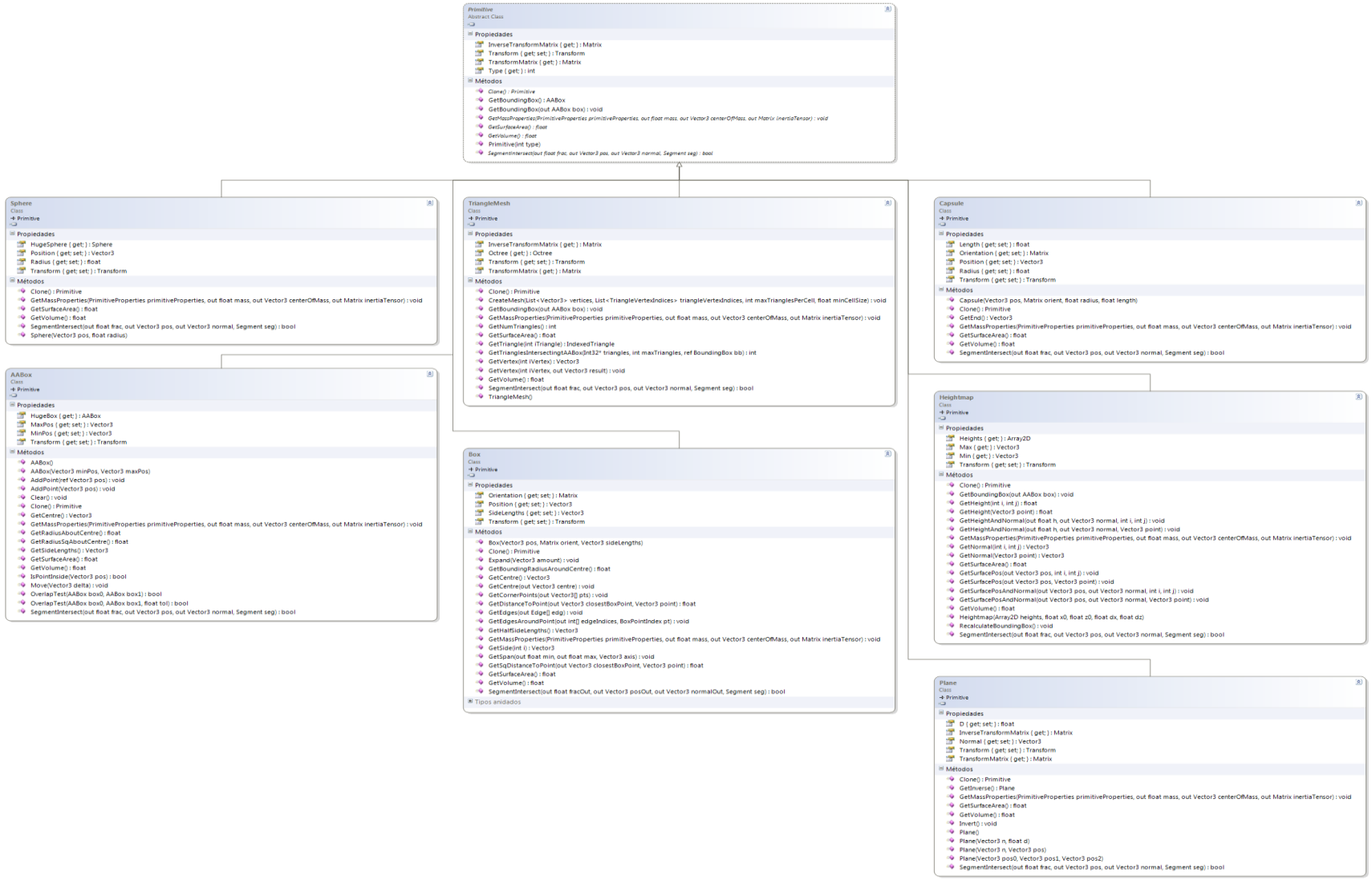


Diagrama de clases de las restricciones de los objetos físicos.

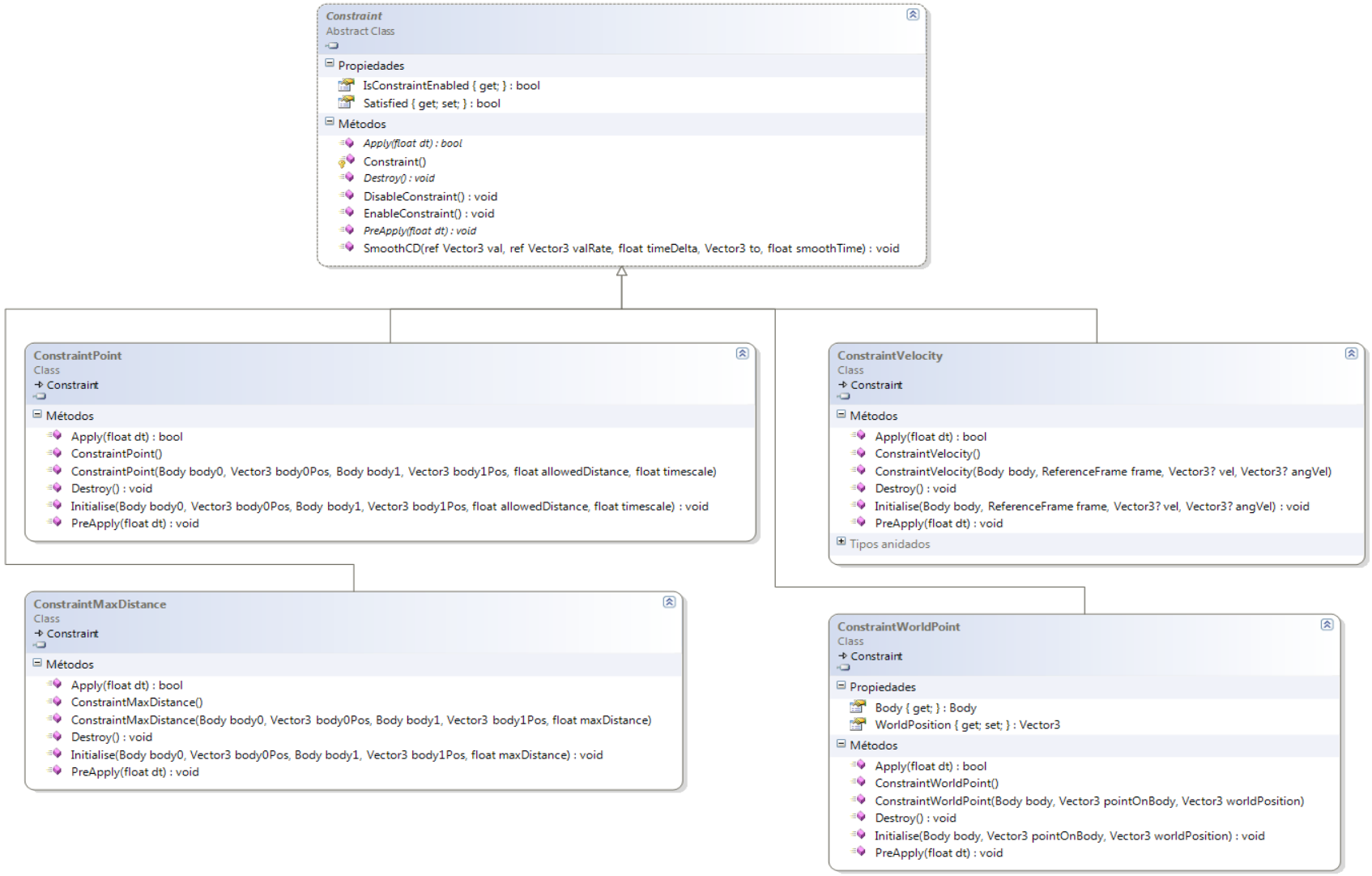
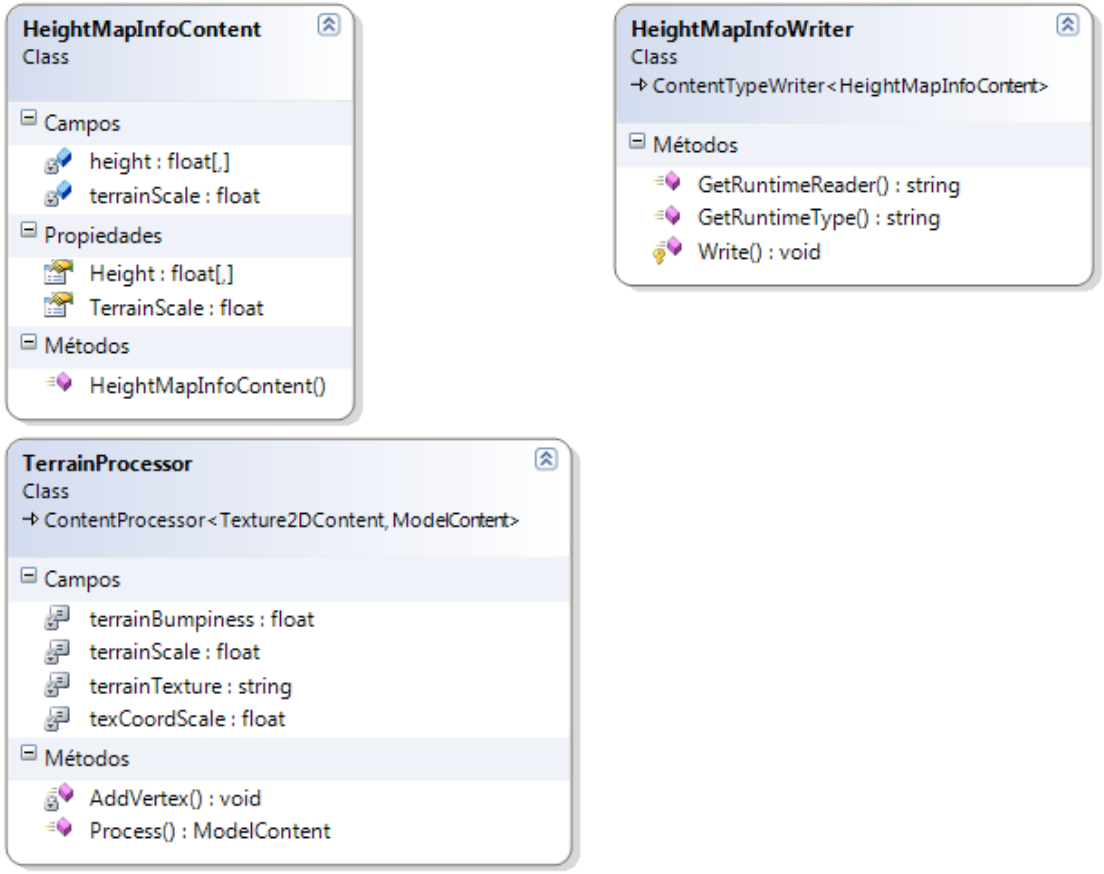


Diagrama de clases del procesador de mapa de alturas

El procesador de mapa de alturas pasa una imagen en escala de grises a un modelo 3D, las partes más oscuras representa menor realce en el modelo 3D que será obtenido a partir de este, las partes más claras tendrán más relieve. Todos los valores para obtener el modelo 3D generado son configurables.

Este procesado de mapa de alturas acompaña la descarga del motor de física JibLibX.



Sistema de animación 3D

La animación de modelos de los personajes 3D de los proyectos desarrollados se lo realizo a través de una librería de animación para XNA llamada XNAnimation, la cual reproduce clips de animación esqueletal, un tipo de animación de modelos 3D previamente explicado, librería de la cual también se tuvo acceso a los códigos fuentes.

Diagrama de clases del procesador de contenido XNAnimation Pipeline.

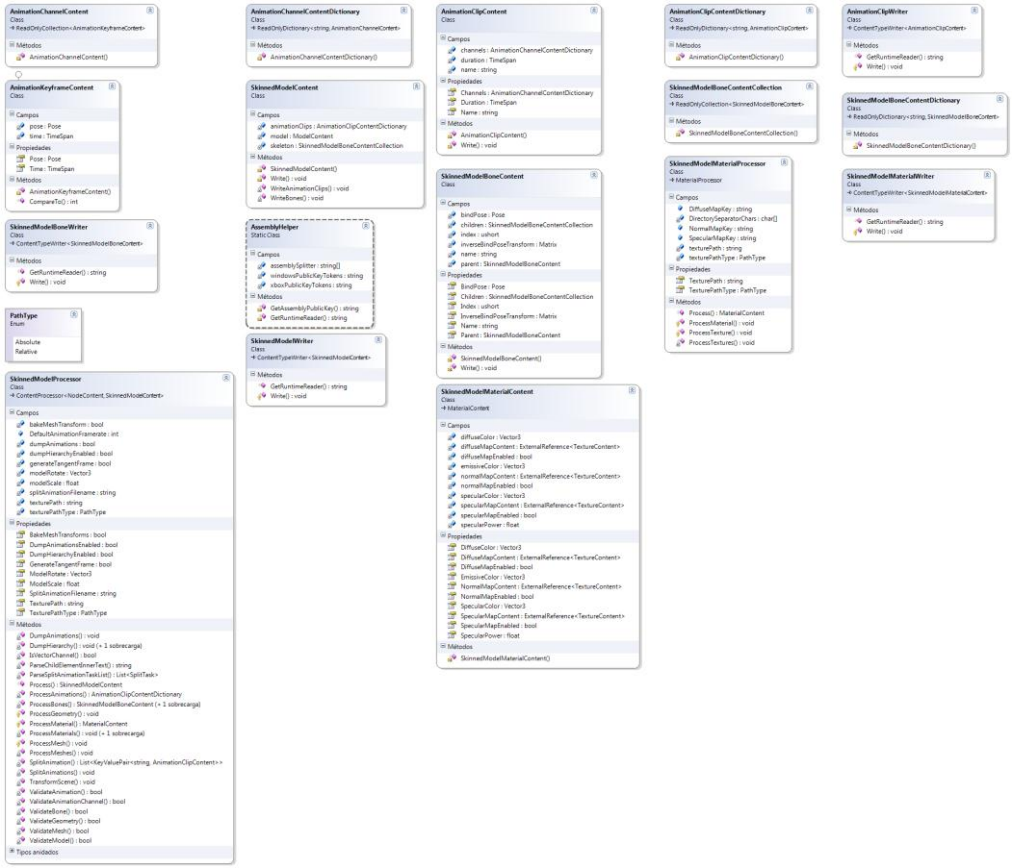
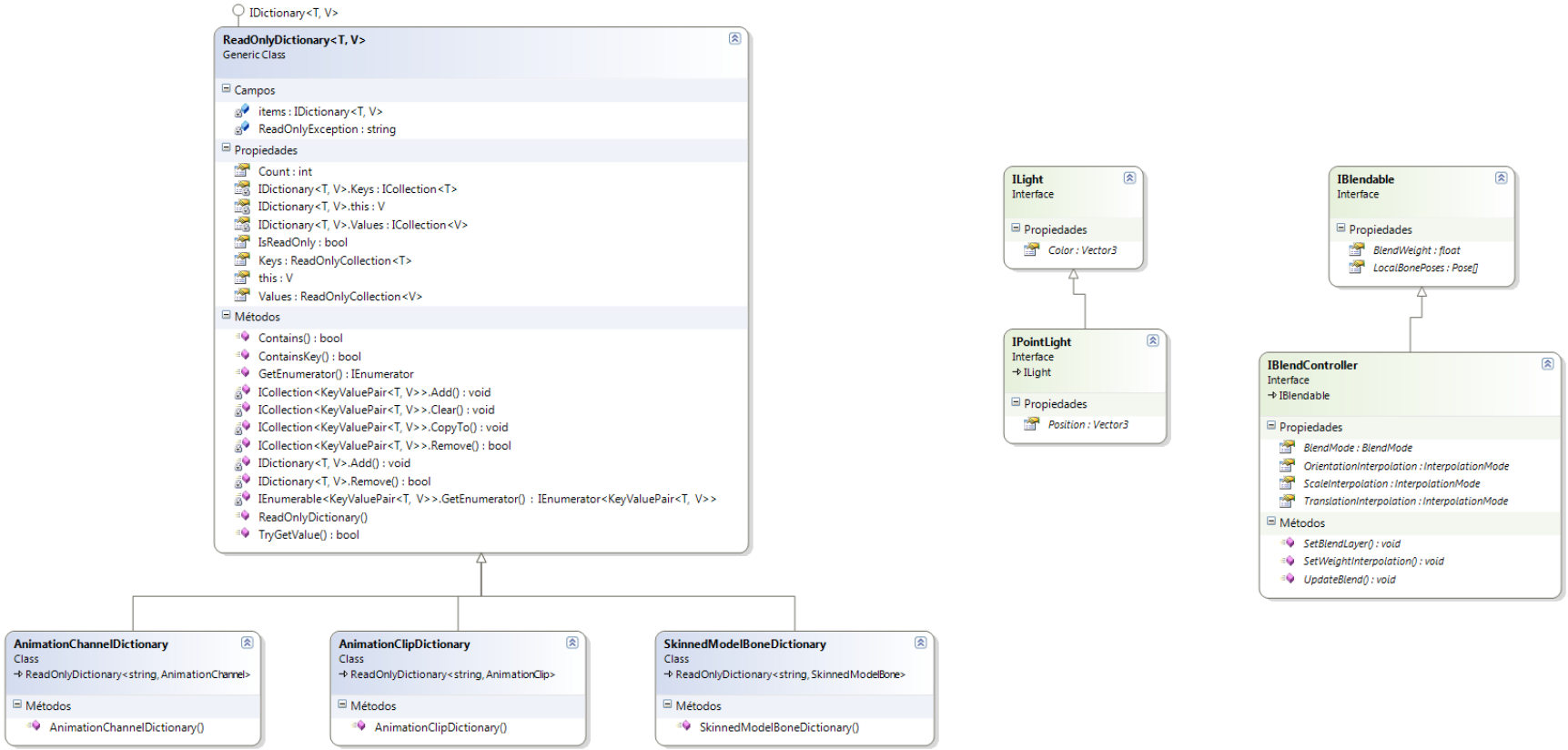
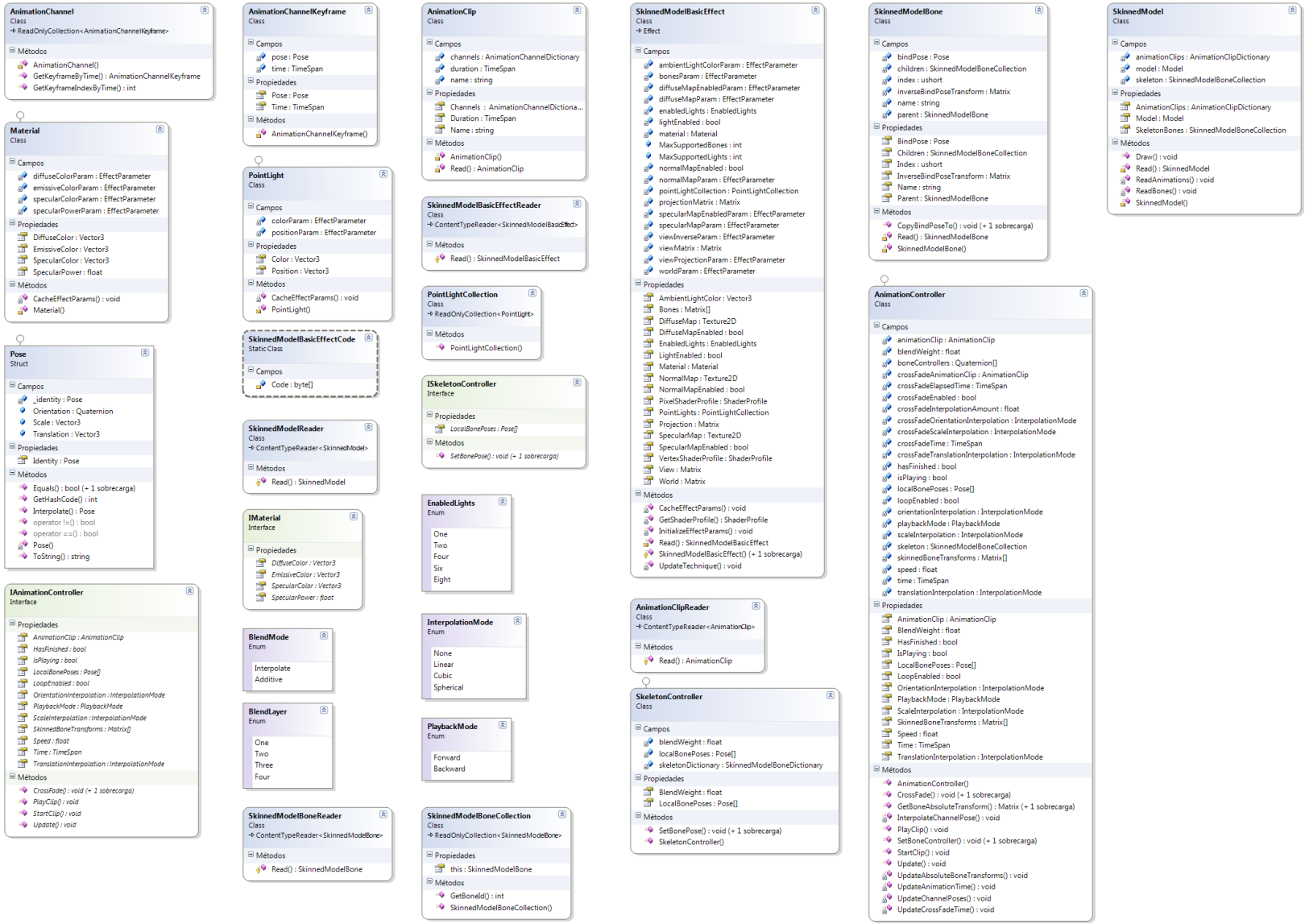


Diagrama de clases XNAnimation.





En ambas versiones de los videojuegos desarrollados se utilizan estos dos componentes esenciales.

Sistema de partículas DPSF

El sistema de partículas utilizado se lo obtuvo desde internet, pero solo a los compilados, es decir archivos dll, pero que fue adaptado sin problemas, ya que la versatilidad de este sistema de partículas radica en que se maneja a través de plantillas de partículas, fácilmente configurables y adaptables para cualquier efecto especifico que se deseaba implementar, la gran variedad de ejemplos que acompaña la descarga de este componente, facilito enormemente su entendimiento y utilización.

Componentes varios

Los demás componentes, algunos de los cuales fueron desarrollados por el autor, otros mediante ejemplos disponibles en internet se los pudo acoplar e implementar.

En el resto del documento se expondrá dos documentos, en los cuales se mostraran todos los diagramas de clases utilizadas por los proyectos desarrollados respectivamente, los anteriores diagramas de clase no están incluidos en los documentos siguientes ya que no forman parte del proyecto, pero sí de la solución de Visual Studio:

Diagrama de clases Ibarra Game v1.0 (versión para un solo jugador.)

Por el tamaño del diagrama de clases este viene en el DVD que contiene la documentación de la tesis.

Diagrama de clases Ibarra Game Network v1.0 (versión multijugador.)

Por el tamaño del diagrama de clases este viene en el DVD que contiene la documentación de la tesis.

3 Requerimientos derivados

Interfaces intuitivas para el usuario.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Realización Caso de Uso:

Criterio para ganar la partida

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
19/03/2011	1.0	Elaboración del documento de la realización del caso de uso: Criterio para ganar la partida.	Wilmer Carrera

Tabla de Contenidos

Realización Caso de Uso:

Criterio para ganar la partida

1 Introducción

1.1 Propósito

Definir como se realizó la implementación funcional del criterio para ganar la partida en las versiones de los proyectos desarrollados

1.2 Alcance

El presente documento específica como se realizó la búsqueda del ganador del videojuego cuando este está en ejecución.

1.3 Definiciones, Siglas, y Abreviaciones

Ver glosario.

1.4 Referencias

Especificación de casos de uso: Criterio para ganar la partida.

1.5 Descripción

Mostrar el flujo lógico para la búsqueda del ganador del videojuego en cualquiera de las versiones que se está ejecutando.

2 Flujo de eventos - Diseño

El criterio para ganar la partida es diferente en las versiones de los videojuegos construidos, en la versión para un solo jugador, el jugador debe completar una serie de misiones que le serán informadas al principio de cada misión, en la versión multijugador, quien cumpla los parámetros de configuración del videojuego, será quien gane la partida.

La búsqueda del ganador del videojuego se lleva a cabo en la clase principal de los videojuegos desarrollados.

Diagrama de clases MainGameScreen.cs (Clase principal de la versión para un solo jugador)

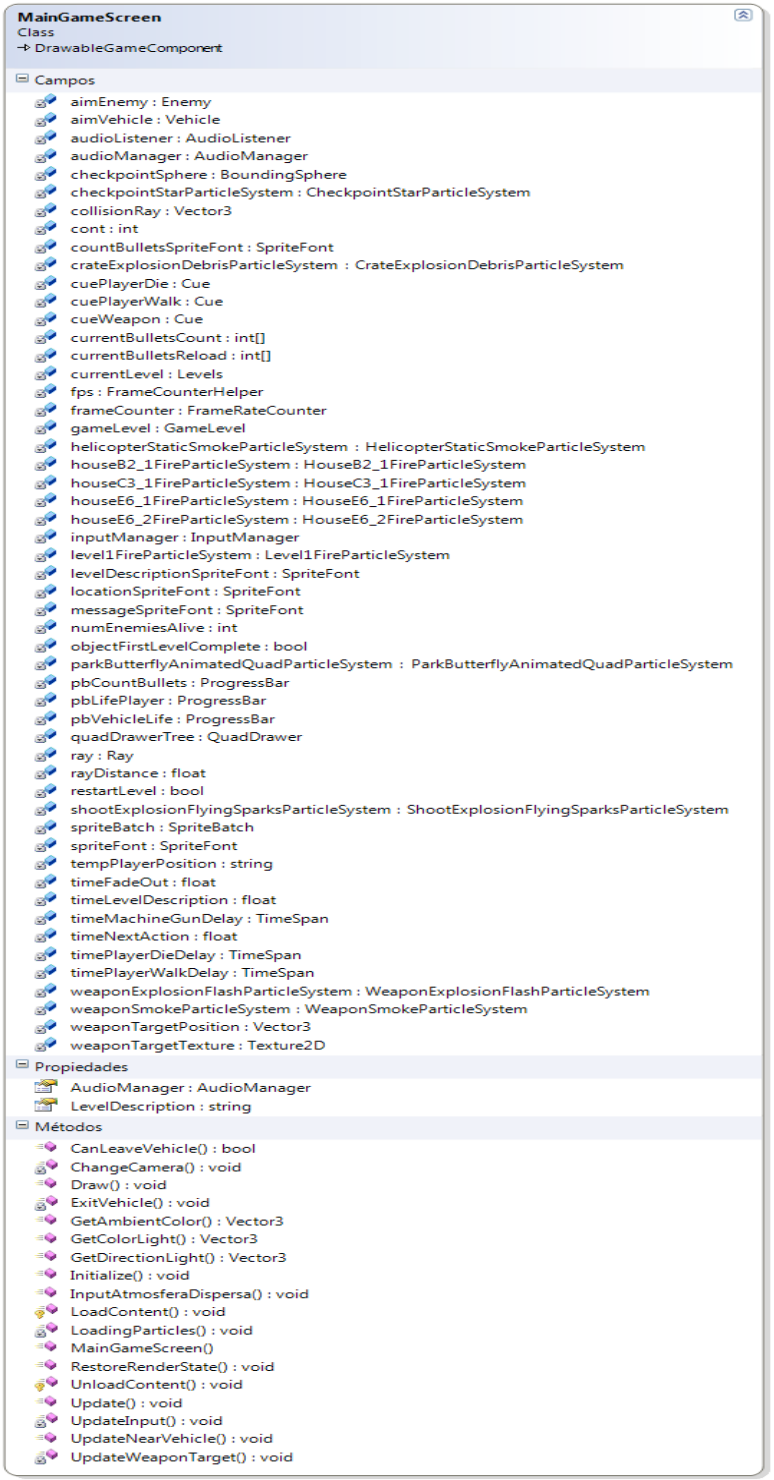
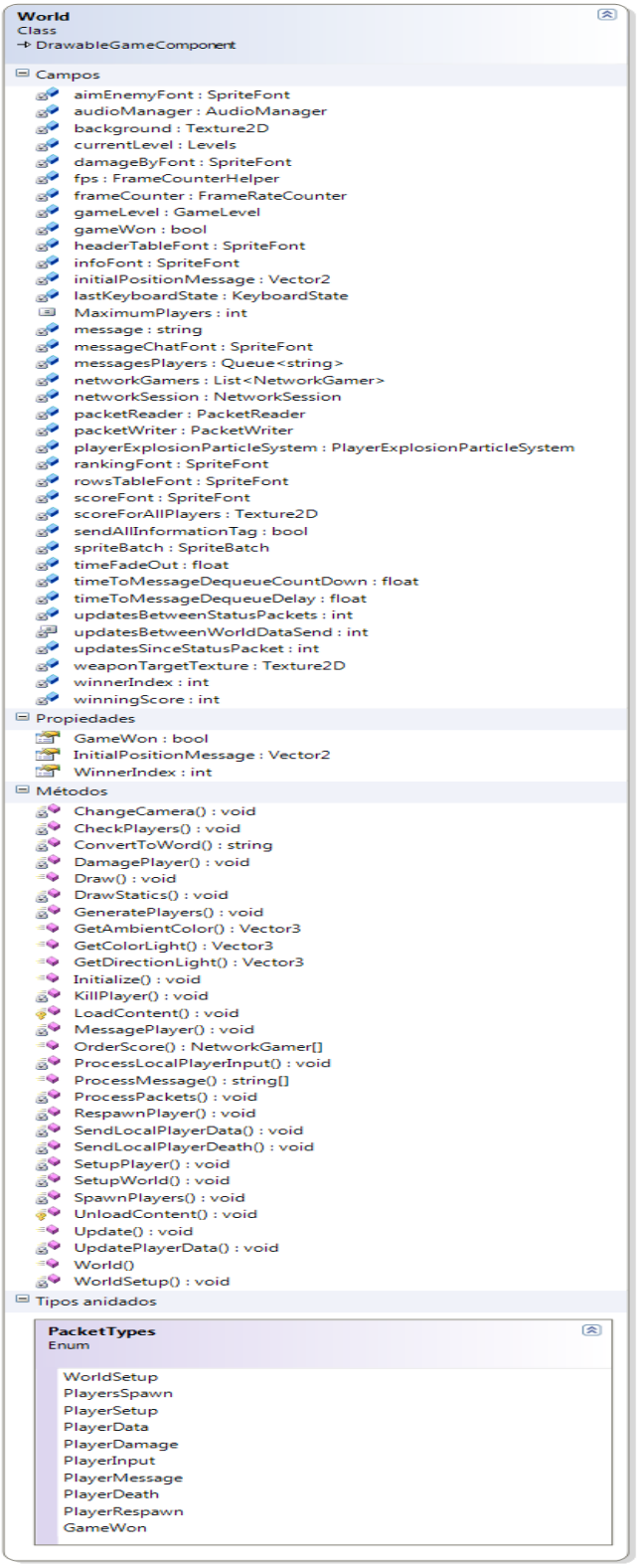


Diagrama de clase World.cs (Clase principal de la versión multijugador)



Como se explicó en la documentación técnica del proyecto, el desarrollo de la lógica del videojuego se desarrolla en la clase UPDATE de las clases, nótese que estas clases

derivan de la clase DrawableGameComponent, es decir estas clases son componentes con comportamiento gráfico y que puede ser utilizado en proyectos futuros.

3 Requerimientos derivados

Interfaces intuitivas para el usuario.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Realización Caso de Uso:

Pérdida del videojuego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
11/04/2011	1.0	Elaboración del documento de la realización del caso de uso: Pérdida del videojuego.	Wilmer Carrera

Tabla de Contenidos

Realización Caso de Uso:

Pérdida del videojuego

1 Introducción

1.1 Propósito

Definir como se realizó la implementación funcional del criterio para ganar la partida en las versiones de los proyectos desarrollados.

1.2 Alcance

El presente documento específica la realización de la lógica de pérdida del videojuego en ejecución, en cualquiera de las versiones desarrolladas.

1.3 Definiciones, Siglas, y Abreviaciones

Ver glosario.

1.4 Referencias

Especificación de casos de uso: Pérdida del videojuego.

1.5 Descripción

Mostrar el flujo lógico para que el usuario debido a ciertas circunstancias pierda el videojuego.

2 Flujo de eventos - Diseño

Debe entenderse que perder el videojuego como concepto de videojuegos, es decir, perder el videojuego equivale a perder la partida o sesión de juego que el usuario está ejecutando, en términos técnicos, cuando los puntos de vida del jugador llegan a ser menor que uno.

El flujo de eventos es el mismo que el caso de uso “Criterio para ganar la partida” debido a que estos casos de uso son parte funcional del videojuego, el flujo de estos eventos se

desencadenan en los métodos de la clase principal del videojuego, según la versión que se esté ejecutando.

En la versión desarrollada para un solo jugador para perder la partida además de que los puntos de vida lleguen a ser menor que uno, también de acuerdo a los objetivos de las niveles de juego se establece condiciones que también afectan a la pérdida de puntos de vida del jugador.

La versión multijugador presenta una condición adicional para perder la partida, solo cuando se cumpla, será que el jugador salga del escenario y caiga hacia el vacio.

Los diagramas de clase para este caso de uso son los mismos que los presentados en “Criterio para perder la partida”.

3 Requerimientos derivados

Interfaces intuitivas para el usuario.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Realización Caso de Uso:

Salir del videojuego

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
11/04/2011	1.0	Elaboración del documento de la realización del caso de uso: Salir del videojuego.	Wilmer Carrera

Tabla de Contenidos

Realización Caso de Uso:

Salir del videojuego

1 Introducción

1.1 Propósito

Definir como se realizó la implementación funcional para salir del videojuego cuando este está en ejecución.

1.2 Alcance

El presente documento específica la opción que el jugador tiene para salir del videojuego en ejecución.

1.3 Definiciones, Siglas, y Abreviaciones

Ver glosario.

1.4 Referencias

Especificación de casos de uso: Salir del videojuego.

1.5 Descripción

Mostrar el flujo lógico para que el usuario pueda salir del videojuego cuando este está en ejecución o en una partida está en progreso.

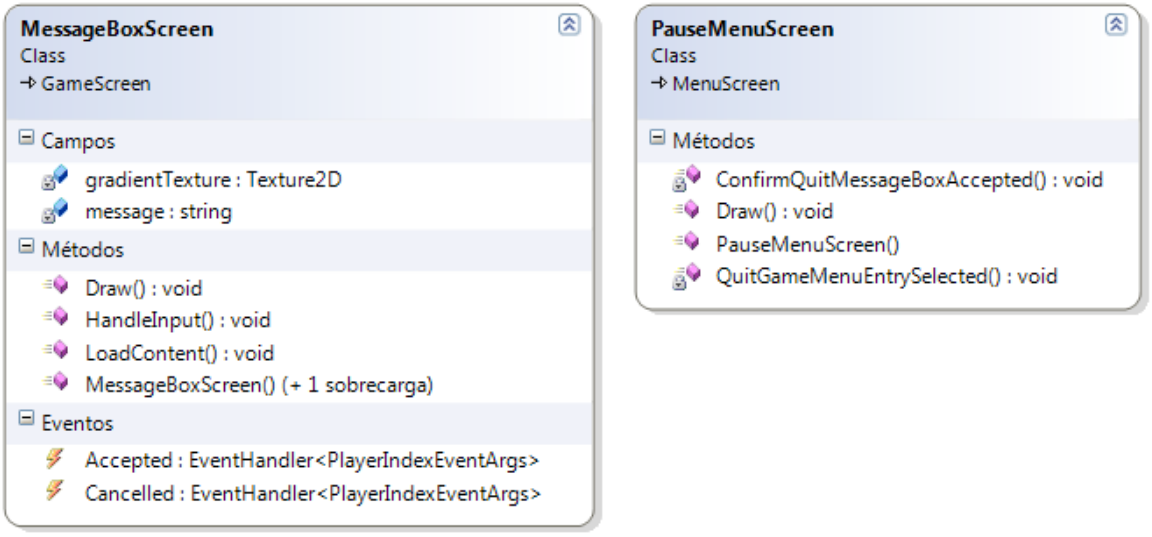
2 Flujo de eventos - Diseño

El jugador tiene la posibilidad de abandonar la partida en el momento en que o decida, con las consecuencias que esto conlleva, las más inmediata y que afectaría directamente al jugador, es la perdida de datos del progreso de la sesión o partida que se esté desarrollando.

Si el jugador decide abandonar la partida, este debe presionar la tecla de escape, posteriormente se le presentara un menú emergente con opciones, una de ellas para salir de la partida, si en caso se seleccionara esta opción, el juego en ejecución terminaría y el

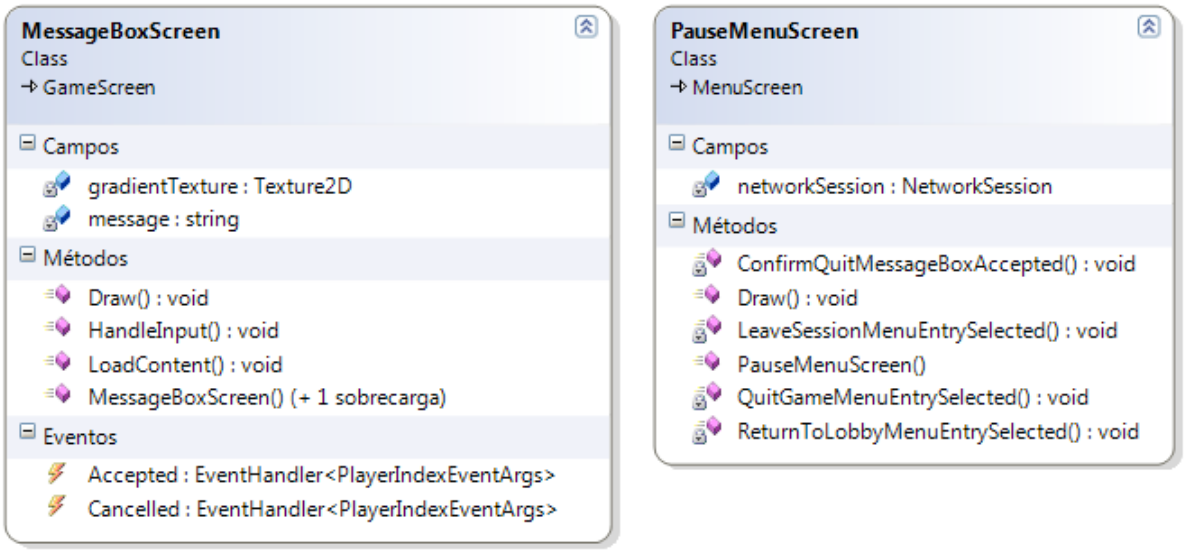
jugador será direccionado a la primera pantalla de inicio de la aplicación, donde podría recomenzar el videojuego o salir completamente de la aplicación.

Diagrama de clases de las pantallas que son mostradas cuando el jugador decide abandonar la partida (versión para un solo jugador).



Dependiendo de la versión que se esté ejecutando las opciones pueden diferir.

Diagrama de clases de las pantallas que se muestran cuando el jugador decide salir del videojuego (versión multijugador del videojuego).



En la versión multijugador antes de salir del videojuego, se desencadenan eventos de salida de sesión de red y los demás jugadores de la red, que estén dentro de la sesión del videojuego, deben conocer que un jugador ha salido del videojuego.

3 Requerimientos derivados

Interfaces intuitivas para el usuario.

Universidad Técnica del Norte

Ibarra Game v1.0 & Ibarra Game Network v1.0

Plan de Construcción de Integración

Versión 1.0

Historial de Revisión

Fecha	Versión	Descripción	Autor
04/04/2011	1.0	Elaboración del documento	Wilmer Carrera

Tabla de Contenidos

Plan de Construcción de Integración

1 Introducción

1.1 Propósito

Este documento describe el plan para la integración de los componentes del software Ibarra Game v1.0 & Ibarra Game Network v1.0 sistemas que serán propuestos como proyecto de tesis, dentro de un ejecutable y prototipo demostrable.

1.2 Alcance

El plan de construcción de integración aplica al sistema desarrollado como tesis para la obtención del título en Ingeniería en Sistemas Computaciones.

1.3 Definiciones, Siglas, and Abreviaciones

Ver glosario.

1.4 Referencias

Las referencias aplicables son:

- Documento de visión v1.0
- Lista de riesgos v1.0
- Glosario 1.0
- Metodología RUP, 1999, Rational Software Corp.

2 Subsistemas

La fase de elaboración se describió el prototipo de arquitectura de desarrollo para verificar la rapidez y rendimiento de la arquitectura para la primera versión. Esto incluyo la implementación de interfaces hacia subsistemas externos.

El proceso principal es la ejecución del videojuego, donde todos los componentes que han sido incorporados dentro del motor desarrollado y que han sido adquiridos de fuentes externas, no implementa características para la seguridad de la aplicación ya que cualquier usuario puede acceder a la aplicación.

EL motor de videojuegos desarrollado es quien implementa las interfaces de comunicación entre los componentes del mismo.

La siguiente tabla ilustra los subsistemas y procesos a implementar para el prototipo de arquitectura.

Subsistema	Procesos	Componentes
Motor de física Procesador de mapa de alturas Sistema de partículas Librería de animación 3D Librería de animación 2D	Ejecución de la aplicación	DLL

3 Construcciones

La integración (en la iteración) está dividida en un numero de incrementos, cada resultado es una construcción de la pruebas de integración. La integración del prototipo está organizado en 2 construcciones de integración como se describe a continuación:

La construcción de la integración incluye los siguientes pasos:

- Ensamblado de componentes específicos dentro de los directorios de compilación.
- Creación y compilación de enlaces hacia archivos de comandos.
- Compilación y enlace a componentes dentro del ejecutable.
- Ejecución de pruebas de integración.

3.1 Construcción de integración 1

La primera construcción de integración sigue la siguiente funcionalidad básica:

- **Opciones de juego:** configuración de la aplicación.
- **Ejecución del videojuego:** proceso principal del sistema.

La primera construcción de integración incluye los siguientes subsistemas y compontes.

Subsistemas	Componentes
Opciones de juego	Aplicacion.exe
Ejecución del videojuego	XnAnimation.dll DPSF.dll JiglibX.dll

3.2 Construcción de integración 2

La segunda construcción de integración sigue la siguiente funcionalidad básica:

- **Cerrar la aplicación:** guardar la partida y cerrar la aplicación.

La segunda construcción de integración incluye los siguientes subsistemas y componentes.

Subsistemas	Componentes
Cerrar la aplicación	Aplicación principal
Guardar datos de la partida.	Archivo de texto con datos de la partida.

CAPITULO V.

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

Se consiguió completar las versiones de los videojuegos propuestos en su totalidad y superando las expectativas iniciales del proyecto ya que en un principio para la versión para un solo jugador no se planteó la posibilidad de utilizar vehículos con comportamiento físico, efectos de sistemas de partículas, luces ni sombras, y para la versión multijugador nunca se pensó en la posibilidad de incorporar un chat entre los jugadores y por el desconocimiento inicial de framework no se conocía que se podía transmitir streaming de voz a través de la red, simulación de trafico de red, únicamente instanciado un componente, estas y otras características que se pueden ver y probar en los videojuegos desarrollados que dan como resultado un producto con mayores prestaciones de calidad y presentación.

Los estándares de calidad para este tipo de software no están definidos por las compañías o personas que se dediquen a la creación de videojuegos, si no más bien quien decide si un producto de estos es de calidad, es el usuario que utilice el producto, debido a esto, uno de los objetivos planteados no puede ser completado o definido por lo antes expuesto, además al ser novatos en este campo no se puede definir reglas que se deben seguir para entregar un producto final de calidad, si bien hasta ahora con las compañías existentes no hay normas publicadas para el desarrollo de videojuegos, con este antecedente no se tiene la autoridad suficiente como para dar reglas o como fue expuesto estándares de calidad sobre un producto de entretenimiento, por la razón antes citada sobre el ser novatos, fue también un motivo por lo que se estableció como objetivo el entregar como resultado estándares de calidad que como se manifestó no pudo ser completado con las aclaraciones pertinentes.

Dada la complejidad del desarrollo de un videojuego, en algunos casos fue necesaria la adaptación de elementos ya existentes y de acuerdo a lo que el proyecto requería estos fueron modificados.

Con el esfuerzo necesario y aprendizaje metodológico del framework XNA, se puede lograr desarrollar un videojuego completo y avanzado.

Se superó con éxito la dificultad que presentaba este tipo de sistemas, al principio un tanto frustrante, pero que continuamente durante el desarrollo del sistema esa sensación se iba convirtiendo en interesante y emocionante.

Se creó una historia y un guion completo tomando como referencia la geografía de Ibarra, resaltando que el presente proyecto se basó en el videojuego Resident Evil de la compañía CAPCOM. En un principio no se lo concibió así, pero durante el desarrollo del sistema adoptó esta similitud con el videojuego mencionado.

No se consiguió producir los sonidos propios para el videojuego, por la falta de tiempo, y desconocimiento de programas que ayuden con este propósito, motivo por el cual, todos los sonido implementados por el sistema fueron recolectados desde fuentes externas y adaptadas al mismo.

Se logró utilizar una herramienta de modelo 3D, reconocido en el mercado a nivel mundial, Autodesk 3ds Max 2010, aunque no se lograron modelar objetos 3D complejos, se consiguieron animar modelos 3D de acuerdo a las necesidades del proyecto. Algunos

de los modelos 3D fueron obtenidos desde internet, los mismos que tienen una licencia de uso libre para propósitos no comerciales.

EL motor de videojuegos es extensible y personalizable para videojuegos 3D, pero solo del tipo: disparos en primera persona y en tercera persona.

Se consiguió crear un nuevo mecanismo de unión de jugadores a un videojuego en progreso, para la versión multijugador de los aplicativos desarrollados, opción que este tipo de videojuego requería, y que puede ser utilizado para otros videojuegos dependiendo de la necesidad del mismo.

XNA y C# productos de Microsoft Corporation, derivados de la familia del framework .NET, a lo largo del desarrollo del sistema se notó de la potencia de estos productos, que cada año va mejorando, lo más actual, el cambio de .NET 3.5 a .NET 4.0, no todos los productos de dicha empresa son restringidos ni costosos, existen proyectos de código abierto que fácilmente se los puede adquirir a través de internet y en cuanto a costos el mejor ejemplo, XNA es de descarga y uso libre para quien desee utilizar este excelente framework.

Como última conclusión con certeza se puede manifestar que desarrollar este tipo de sistemas es emocionante por ser una tarea ardua, gratificante por el conocimiento adquirido, divertido por el tipo de sistema que se estaba desarrollando, y según la evolución del desarrollo de sistemas de entretenimiento generará expectativa por conocer la que vendrá.

"Hacer software es más que conectarse a una base de datos".

5.2 Recomendaciones

XNA no solo se lo puede utilizar en el desarrollo de videojuegos, aunque sea un framework para este fin, a partir de este se puede derivar muchos otros productos como: sistemas multimedia, aplicaciones para dispositivos móviles, etc. Un mercado que en el país no es muy explotado.

Hasta la elaboración del presente documento, está en auge los dispositivos con soporte para Windows Phone 7, implementaciones de aplicaciones XNA para silverlight, dispositivos de hardware como Tablets Pc, aplicaciones Touch Screen, entre otras (inicios del año 2012), con los conocimientos necesarios sobre este framework no solo se desarrollaría para PC sino también para las plataformas antes mencionadas con lo cual se incursionaría sobre un mercado nuevo con grandes beneficios económicos.

Desarrollar aplicaciones con XNA requiere cierto grado de habilidad al programar y un conocimiento medianamente avanzado del lenguaje de programación, en este caso C#, habilidades que se logró adquirir durante el desarrollo del sistema, resaltando que al inicio del proyecto el conocimiento era básico, es decir, se conocía el lenguaje pero no todo su potencial además del desconocimiento del desarrollo de sistemas de entretenimiento, con lo que resulta recomendable probar nuevas tecnologías de desarrollo para aumentar y mejorar los conocimientos.

Por todos los motivos anteriormente citados es altamente recomendable, ubicar dentro de la malla curricular de las instituciones educativas superiores (Universidades, Institutos, etc.), alguna materia en la cual aplique la enseñanza de este framework o cualquier otro con similares características. Programación Orientada a Videojuegos, posible opción del nombre de una nueva materia para la especialidad de Ingeniería en Sistemas, los beneficios serian múltiples ya que como se puede observar en los anteriores capítulos, se pondrá en práctica todos los conocimientos adquiridos en otras materias como: Matemáticas, Inteligencia Artificial, Simulación de Computadores, Redes de Computadores, Estadística, Diseño Gráfico, Animación y Modelado 3D, programación Orientada a Objetos, etc. Conocimiento aplicado en una sola, con un solo fin.

Actualmente ninguna de las instituciones educativas del país tiene dentro de su esquema de enseñanza, el desarrollo de sistemas de entretenimiento, aunque a nivel mundial es una de las ramas que más beneficios económicos brinda, como la industria del cine.

Como se puede contemplar actualmente en el país son pocas las empresas que se dedican al desarrollo de sistemas de entretenimiento, dentro de estos los videojuegos, una buena opción con poca competencia donde la calidad del producto es la que se impone ante los demás, por esto, con un buen equipo de desarrollo, poco capital, diseñadores y sobre todo entusiasmo y dedicación se podría conseguir buenos resultados.

CAPITULO VI.

TRABAJOS FUTUROS

En esta sección se propone lo que se podría realizar, no obstante solo son propuestas quizás nunca se las realice (“espero realizarlas”), pero vale la pena mencionarlas, quien revise este documento las puede proponer como su proyecto de tesis o con cualquier otro fin.

Quien sea que lea este documento o esta sección, o tal vez solo esta hoja, sea quien piense que en desarrollo de software hay más por hacer que sacar un reporte de una base de datos, con esto no se quiere menospreciar otros proyectos, sino más bien abrir otras opciones que no han sido explotadas, ya que para las opciones existentes cada vez más la competencia crece y este campo al ser uno de los menos explotados en nuestro país, sería una buena opción incursionar en este ámbito.

6.1 Videojuegos profesionales

La industria de los videojuegos nunca terminará, y con el avance del tiempo, las nuevas tecnologías en hardware y software serán mejores y con mayores prestaciones, por este motivo y muchos otros, se debería incursionar en esta industria, un buen equipo de desarrolladores, diseñadores, creativos y demás personal necesario, se podría crear videojuegos mucho más profesionales y de mejor calidad, los videojuegos presentados en esta tesis fueron desarrollados por una sola persona, y cabe mencionar que la calidad de estos no es muy buena, debido a la complejidad del mismo y por los muchos detalles artísticos que se debe emplear en proyectos de este tipo.

Los videojuegos que se pueden desarrollar son infinitos, solo se necesita un buen equipo, tiempo, y trabajo desinteresado, ya que al principio dedicarse a esta industria tal vez no deje grandes ingresos económicos, pero la originalidad de una idea plasmada en un videojuego es donde se establece las opciones de acogida del producto por parte del usuario o jugador.

Las principales empresas desarrolladoras de videojuegos son Capcom, Konami, Electronics Arts, etc., nótese que son compañías extranjeras, y ninguna de las comercialmente conocidas son de América Latina.

Figura 78. Videojuego desarrollado por You yun tech con Microsoft XNA

Fuente: El autor

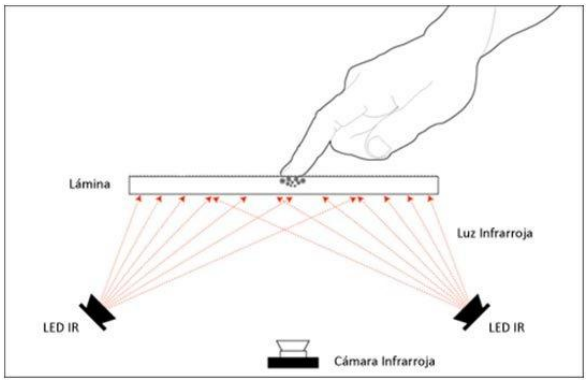
6.2 Aplicaciones multitouch

Uno de los proyectos más ambiciosos, y que se lo podría desarrollar con esta tecnología, son las aplicaciones multitouch obviamente para dispositivos que soporten esta tecnología.

El siguiente contenido fue extraído del blog de Javier Canton en que comparte información de varios temas de esta tecnología XNA.

Basado en la tecnología llamada "Diffused Illumnation" en la cual la idea principal es iluminar una superficie traslucida desde abajo con luz IR y usar una cámara IR para detectar solo la luz con dicha longitud de onda, al acercar por ejemplo una mano desde arriba a dicha superficie todo aquello que se situé más cerca reflejará dicha luz IR con mayor intensidad que lo que se encuentre a mayor distancia, es decir la yema de nuestros dedos será diferenciada del resto de la mano gracias a los niveles de intensidad reflejada.

Figura 79. Elementos de hardware



Fuente: http://geeks.ms/blogs/jcanton/archive/2010/01/02/multitouch.aspx

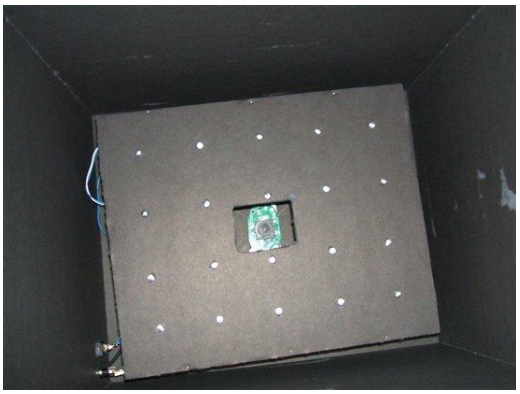
Para la construcción se puede utilizar una caja, también se necesita cámara IR, cualquier webcam es capaz de detectar el IR solo que viene de fábrica con un filtro de IR para que la luz de dicha longitud de onda no llegue al sensor, tan simple como quitarle dicho filtro y ya llegaba la información IR al sensor, pero también la visible, lo ideal es que solo llegara la IR.

Figura 80. Elementos de hardware – webcam

Fuente: http://geeks.ms/blogs/jcanton/archive/2010/01/02/multitouch.aspx

Luego de modificar la webcam, ya solo faltaba iluminar la superficie con led infrarroja para lo que se puede usar un pequeño circuito de 20 leds alimentados por un transformador a 12 V:

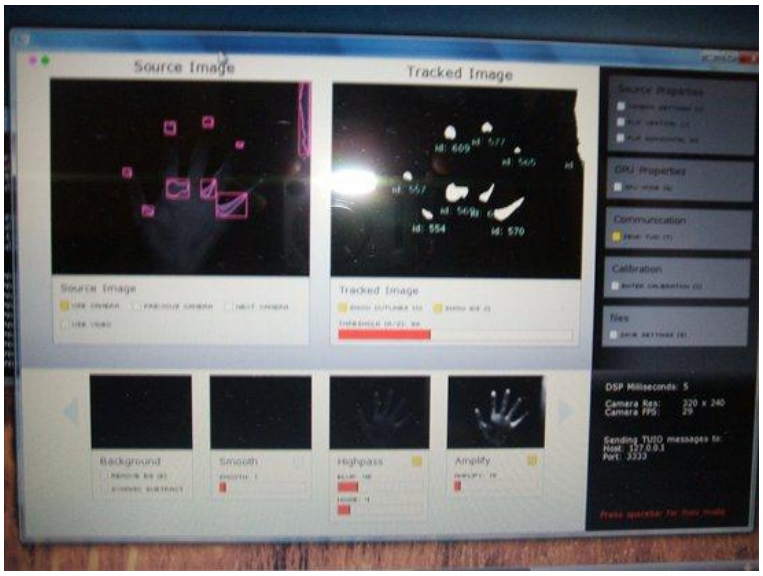
Figura 81. Dispositivo multitouch



Fuente: http://geeks.ms/blogs/jcanton/archive/2010/01/02/multitouch.aspx

Para el software se debe crear el driver para poder usar dicho dispositivo desde código, este driver consistía en un sistema que fuese capaz de ir capturando las imágenes de la cámara, aplicarle algunos filtros y realizar tracking sobre los puntos detectados, esa lista de puntos sería usada por las aplicaciones a desarrollarse. Para este sistema puede basarse en una solución open source ya existente llamada TBeta y desarrollada por NUI Group, esta aplicación ya realiza dicha tarea y actúa como servidor, enviando por un socket UDP dicha lista de puntos.

Figura 82. Software multitouch



Fuente: El autor

6.3 Aplicaciones móviles

Figura 83: Dispositivo móvil

Windows Phone 7 es un sistema operativo móvil desarrollado por Microsoft, como sucesor de la plataforma Windows Mobile. Está pensado para el mercado de consumo generalista en lugar del mercado empresarial por lo que carece de muchas funcionalidades que proporciona la versión anterior. Microsoft ha decidido no hacer compatible Windows Phone 7 con Windows Mobile 6 por lo que las aplicaciones existentes no funcionan en Windows Phone 7 haciendo necesario desarrollar nuevas aplicaciones. Con WP7 Microsoft ofrece una nueva interfaz de usuario, integra varios servicios en el sistema operativo y planea un estricto control del hardware que implementará el sistema operativo, evitando la fragmentación con la evolución del sistema. Microsoft planea una importante actualización para finales de 2011 que incluirá Internet Explorer 9 y algunas mejoras que según Microsoft lo harán competitivo con sistemas operativos de móviles actuales como iOS de Apple o Android de Google.

En la versión actual de Visual Studio ofrece la posibilidad de desarrollar aplicaciones para dispositivos móviles que tengan el sistema operativo Windows Phone, ya sea con Silverlight o con XNA, no solo se podrá desarrollar juegos para dichos dispositivos con las bondades que ofrece el framework de XNA, sino también aprovechar toda la potencia y características del framework para desarrollar aplicaciones móviles a través Silverlight como: eventos touchscreen, reconocimiento de gestos, captura de eventos de hardware etc.

El desarrollo de aplicaciones para estos dispositivos es escaso en nuestro medio, por lo cual sería una buena alternativa para los desarrolladores de software para incursionar en un mercado nuevo.

"Como puede notarse todos estos proyectos propuestos son nuevos en términos tecnológicos para el mercado nacional. Las posibilidades de desarrollo de aplicaciones, específicamente con el framework XNA son limitadas solo por la creatividad del desarrollador o del equipo de desarrollo, el alcance y los resultados que se podrían conseguir serian impresionantes."

BIBLIOGRAFÍA

Libros consultados

- Curtis Bennett, **A Simple Introduction to Game Programming With C# and XNA 3.1**, First Edition: 2009, Estados Unidos de America.
- Jerry Lee Ford, Jr. **XNA 3.1 Game Development for Teens**, Course Technology PTR A part of engage Learning, Estados Unidos de America.
- David. M Bourg, **Physics for Game Developers**, Primera edicion, O'Reilly, Estados Unidos de America, Enero 2002.
- Aaron Reed**, Learning XNA 3.0, Primera edición**, O'Reilly, Estados Unidos de América, Noviembre 2008.
- Aaron Reed**, XNA 3.0 Unleashed**, Primera edición, O'Reilly, Estados Unidos de América, Enero 2009.
- Rob Miles, **Introduction to Programming Through Game Development Using Microsoft XNA Game**, Primera edición 2010, Microsoft Press, Estados Unidos de América.
- Rob Miles, **C# Development**, Edición 1.1, University of Hull, Estados Unidos de América, Octubre 2009.
- Riemer Grootjans, **XNA 3.0 Game Programming Recipes**, Primera edición 2009, Apress, Estados Unidos de América.
- Alexandre Santos Lobão, Bruno Evangelista, José Antonio Leal de Farias, and Riemer Grootjans, **Beginning XNA 3.0 Game Programming From Novice to Professional**, Primera edición 2009, Apress, Estados Unidos de América.

Direcciones electrónicas

- http://msdn.microsoft.com/en-us/xna/default.aspx
- http://forums.xna.com/forums/
- http://www.codeplex.com/site/search?projectSearchText=xna
- http://www.ziggyware.com/
- http://xnacommunity.codeplex.com/
- http://www.tombraiderforums.com/showpost.php?p=3473801&postcount=1
- http://code-spot.co.za/2009/04/08/how-to-turn-xsi-mod-tool-into-a-level-editor-for-your-xna-games-updated-for-xna-30/
- http://mpe.codeplex.com/
- http://jiglibx.codeplex.com/
- http://www.hilva.com/

Todos los recursos que pudieron ser consultados están en el idioma inglés, a excepción de la información extraída de la comunidad española alojada en codeplex.

ANEXOS

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

ESCUELA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES

PLAN DEL PROYECTO DE TITULACIÓN

Propuesto por:	**Áreas Técnicas del Tema:**
Wilmer Manuel Carrera Pérez	Diseño Gráfico Multimedia Redes de Computadores Desarrollo de Software Inteligencia Artificial Simulación Computacional
Director del Proyecto: Ing. Mauricio Rea	**Fecha:** 17 de mayo de 2010

1. Tema del proyecto

Desarrollo de un videojuego con texturas y modelos 3D, multiusuario, en un entorno cliente/servidor (internet) utilizando el framework de Microsoft XNA.

2. Problema

2.1. Antecedentes

A nivel mundial, grandes compañías se han dedicado al desarrollo de videojuegos para computador y otras consolas como XBOX, PS2 entre otras, estableciendo un mercado de consumidores considerablemente alto, generando así ganancias significativas a los fabricantes de estos.

Los videojuegos no son un mal hábito para las personas, sino, son una fuente de entretenimiento, además de conseguir el máximo rendimiento de los componentes del computador donde se implantará el software, y según la calidad del mismo, el usuario estará interesado y con interés en probar la aplicación.

Algunas simulaciones computacionales realizadas sobre sucesos o eventos reales que ocurren, han terminado por ser videojuegos como los simuladores de vuelo, pistas de autos, etc. Representando así una realidad virtual en la cual se simula eventos que pueden ocurrir en realidad.

La producción de videojuegos casi no difiere de la producción de software en general, pero un videojuego de computador tiene más componentes que un software que cumpla alguna actividad, como audio, el sonido propio para cada etapa o escena del videojuego, si el videojuego tiene historia, este será en base a esta, el diseño de personajes, todo videojuego tiene personajes propios nacidos de la invención de los desarrolladores del mismo en caso no tenga historia, niveles del videojuego donde se establece la dificultad o diferentes escenarios donde se desenvuelve la aplicación, y demás componentes, que hacen del desarrollo de videojuegos de computador un conjunto de habilidades que debe poseer el desarrollador de software de entretenimiento.

El desarrollo también depende de la plataforma para la cual se quiera implementar el videojuego, PC, móviles, consolas, ahora con la tecnología a utilizar se puede conseguir la funcionalidad tanto para plataformas PC (S.O. Windows de Microsoft), XBOX 360(Consola desarrollada por Microsoft) y Zune (Dispositivo de multimedia desarrollado por Microsoft).

2.2. Situación actual

Nuestro país es netamente consumidor de videojuegos de computador y otras plataformas, la producción de este tipo de software es escasa en el mercado nacional, sin embargo, a nivel nacional está en auge el desarrollo de software corporativo y a la medida para empresas que necesiten automatizar algún proceso, pero con este crecimiento de ofertas de software, también aumenta la competitividad entre profesionales en desarrollo de sistemas.

A nivel nacional, en casi todas las Universidades o Institutos donde se puede obtener un título como profesional en el desarrollo de sistemas o afines, se capacita a las personas en generar software para satisfacer las necesidades de las empresas u otras problemas que pueden ser resueltos con la asistencia del computador, sin embargo, con el antecedente citado anteriormente: el mercado de consumidores de videojuegos para computador, es uno de las más grandes a escala mundial; establecer otra opción para profesionales en desarrollo de software sería una buena alternativa.

2.3. Prospectiva

Abriendo nuevos caminos para el desarrollo de software, que no solo sea para empresas, el cual actualmente en nuestro país es muy competitivo, establecería a nuestra provincia y eventualmente a nuestro país no solo en consumidor sino en un productor de software de videojuegos. Grandes empresas del mundo se dedican al desarrollo de videojuegos, si esta es una buena opción, se debería incursionar en ella, como ya se mencionó, en el desarrollo de videojuegos no solo interviene profesionales en desarrollo de software,

dentro de este proceso se deben incluir personas con conocimientos en producción de audio, video, imágenes, etc. que también sería una buena opción para dichas personas.

Eventualmente de no abrir nuevas opciones para un profesional en desarrollo de sistemas habrá más competitividad entre las empresas que desarrollen software, sin menospreciar esta actividad que en realidad es muy útil para las empresas y la sociedad que la necesite. Existen muchas oportunidades en el campo laboral para desarrolladores de software, si el desarrollo de videojuegos para múltiples plataformas es una de ellas, y además de tener un gran mercado de consumidores, esta sería una buena opción para abrir nuevos mercados locales para deponer en gran parte nuestra posición de ser consumidores de software de entretenimiento y posteriormente ser un referente como provincia y nación en desarrollo de software de videojuegos.

2.4. Planteamiento del problema

Nuestro país y provincia, el desarrollo de videojuegos presenta algunos problemas:

- El desarrollo de videojuegos es casi nula en nuestro país.
- La escasa capacitación en el uso de tecnologías para el desarrollo de videojuegos.
- Complejidad en el desarrollo de videojuegos.
- El desarrollar un videojuego incluye muchos componentes que deben ser propios de cada videojuego como sonido, historia, texturas, etc.
- Falta de interés por parte de los desarrolladores de software para abrir nuevas opciones que se puede explotar y generar una fuente de ingresos.
- Creencia de que los videojuegos son malos para la sociedad y que no representan utilidad para la misma.

3. Objetivos

3.1. Objetivo General

Desarrollar un videojuego con texturas y modelos 3D, multiusuario, en un entorno cliente/servidor (internet) utilizando el framework de Microsoft XNA.

3.2. Objetivos Específicos

- Presentar una investigación sobre la tecnología de Microsoft para desarrollar videojuegos XNA.
- Desarrollar un videojuego en que se utilice esta nueva tecnología.
- Describir estándares de calidad para sistemas de entretenimiento.
- Aprovechar al máximo las funcionalidades que ofrece el API de XNA en la creación

de videojuegos.

- Establecer una nueva visión en el campo laboral para los desarrolladores de software.
- Definir las ventajas y desventajas de utilizar esta nueva tecnología.
- Con la investigación a realizarse presentar una guía sobre la creación de videojuegos utilizando esta tecnología.

4. Alcance

- El software a desarrollarse podrá ser implementado sobre consolas PC, Xbox 360, Zune, propiedad de Microsoft.
- El videojuego será probado solo sobre PC debido al costo de adquirir las consolas que lo soportan.
- El sistema podrá ser implementado sobre una red de computadores, internet y como última opción para un solo jugador.
- El modelado de los objetos será en 3D.
- El ambiente del videojuego tendrá animaciones y sonido.
- Los objetos que no podrán ser controlados por el usuario tendrán inteligencia artificial.
- El videojuego utilizará lugares de la ciudad de Ibarra como escenarios.
- Los objetos del escenario tendrán efectos como luces, movimiento, etc.
- El control de mando del usuario será a través del teclado y ratón.
- El diseño de los objetos del videojuego serán realizados en una PC normal para mayor compatibilidad donde se instale la aplicación.
- Interfaz agradable e intuitiva para el usuario.

4.1. Descripción del videojuego.

La PC donde se realizará el diseño tendrá las siguientes características:

- Procesador Intel Core(TM)2 Duo CPU 2.0 GHz
- 4 GB RAM
- Windows 7 Ultimate 64 bits.

Propuesta para un solo jugador:

El video juego para este modo será al estilo de la saga **Grand Theft Auto**

Descripción:

- Género: Acción.
- Modos de juego: Un jugador.
- Numero de misiones: 10.
- Niveles de dificultad: Cada escenario o misión del videojuego tendrá su propio nivel de dificultad, especificado por cada misión.
- Historia: La trama empieza con un grupo de tres ladrones atacando una sucursal de un Banco (ficticio). Un hombre y una mujer van en cabeza mientras que otro les cubre las espaldas. Sin embargo, al doblar una esquina, el tercer ladrón es disparado por la mujer, quien huye con el otro hombre y el dinero. El traicionado llamado Miguel (ficticio) ha de hacérselo pagar a los traidores: María (ficticio) ex novia de Miguel, y a Luis, que son cabecillas de un cartel colombiano afincado en Ibarra. Las misiones a realizar serán para atrapar a los traidores. (Trama de GTAIII adaptada).

Propuesta para multijuador

El video juego para este modo será al estilo de la saga **Quake.**

Descripción:

- Género: Acción en primera persona.
- Modos de juego: Multijugador.
- Número de conexiones (jugadores) máximo: 4.
- Objetivo: El juego permitirá a los jugadores, cuyas computadoras están conectadas a una red o a Internet, disputar partidas entre sí, en tiempo real. Usa una estructura de arquitectura cliente-servidor que requiere que los clientes de todas las computadoras de los jugadores se conecten a un solo servidor. El objetivo en *Quake* es moverse a través de todo el campo de batalla eliminando (*fragueando*, del inglés *frag*) a los jugadores enemigos y anotándose puntos basándose en los objetivos del tipo de juego. Cuando los puntos de vida de un jugador llegan a cero, el avatar del jugador es eliminado (*fragueado*); luego el jugador reaparece en otro punto del mapa y sigue jugando con sus puntos de vida restaurados, pero sin las armas ni items que recolectó anteriormente. El juego termina cuando un jugador o equipo alcanza un puntaje específico, o cuando se termina el tiempo. (Objetivo de Quake III Arena).

Mapa del Videojuego.

El límite del videojuego será 5 bloques desde el parque Pedro Moncayo hasta la avenida Sánchez y Cifuentes (Norte). Avenida Obispo Mosquera (Este), Avenida José Mejía (Oeste), Avenida Juan de Salinas (Sur).

5. Justificación del Proyecto

La presente tesis será una investigación sobre la tecnología para el desarrollo de videojuegos utilizando la plataforma de XNA desarrollada por Microsoft que dará como resultado, una metodología para el desarrollo de videojuegos, y el proceso a seguir para conseguir el producto terminado.

Anteriormente, los videojuegos eran desarrollador en C y C++ cuyos lenguajes ofrecían la manipulación directa de los componentes del computador a bajo nivel. Con la actual tecnología proporcionada por Microsoft manipular los controladores de los componentes del computador queda en manos del API de XNA, mejorando así la utilización del teclado, ratón, gráficos (Direcxt y Open GL) del monitor, sonido, etc. Además de optimizar el código para la ejecución del videojuego.

Desarrollar un videojuego es casi similar a desarrollar cualquier otro software en general, además lo que implica desarrollar un videojuego donde interviene la creatividad para implementar otros componentes, como video, sonido, texturas entre otros.

Debido a la creciente complejidad en el proceso de diseño de los videojuegos, mucha gente joven proceden del campo de la informática o del desarrollo de programas, ya que son los más aptos para afrontar el difícil reto de desarrollar videojuegos.

6. Temas Afines Realizados

El presente proyecto no tiene temas similares.

7. Temario

1. Introducción
    a. Introducción al desarrollo de videojuegos
    b. Introducción a XNA
    c. Herramientas adicionales para la creación de videojuegos.
2. Antecedentes
    a. XNA
    b. Entorno de ejecución de XNA
3. Marco Teórico
    a. Introducción.
    b. Teoría de juegos.
    c. Componentes.
4. Análisis de la herramienta a utilizar.
    a. Herramientas comerciales.
    b. Software libre.

5. Descripción de la solución
    a. Género de videojuegos
    b. Proceso lógico para el desarrollo del sistema.
    c. Animación, sonido y video.
    d. Motor de Física
    e. Inteligencia Artificial (IA)
    f. Sistema de partículas
    g. Soporte de red
    h. Detección de colisiones
    i. Documentos de diseño
    j. Bocetos del videojuego
6. Desarrollo del sistema
    a. Metodología de desarrollo de software.
    b. RUP.
7. Conclusiones y recomendaciones
8. Trabajos Futuros
9. Referencias
10. Anexos.
    a. Diseño de la solución
    b. Manual de usuario
    c. Código fuente
    d. Software

8. Bibliografía

Libros y manuales

- Aaron Reed**,** Learning XNA 3.0, Primera edición, O'Reilly, Estados Unidos de América, Noviembre 2008.
- Aaron Reed**,** XNA 3.0 Unleashed, Primera edición, O'Reilly, Estados Unidos de América, Enero 2009.
- Aaron Reed**,** Beginning XNA 3.0 Game Programming, Primera edición, O'Reilly, Estados Unidos de América, Junio 2009.

Direcciones electrónicas

- http://msdn.microsoft.com/en-us/xna/default.aspx
- http://forums.xna.com/forums/

- http://www.codeplex.com/site/search?projectSearchText=xna
- http://www.ziggyware.com/
- http://xnacommunity.codeplex.com/
- http://www.tombraiderforums.com/showpost.php?p=3473801&postcount=1
- http://code-spot.co.za/2009/04/08/how-to-turn-xsi-mod-tool-into-a-level-editor-for-your-xna-games-updated-for-xna-30/
- http://mpe.codeplex.com/
- http://jiglibx.codeplex.com/
- http://www.hilva.com/

9. Cronograma de Actividades

N	Actividad	Duración (Semanas)	Mayo				Junio				Julio				Agosto				Septiembre				Octubre				Noviembre			
1	Investigación	6																												
2	Capacitación	8																												
3	Desarrollo	10																												
4	Pruebas	4																												
5	Documentación	14																												
6	Instalación	1																												

10. Presupuesto

	Descripción	Costo
Hardware	2 Computadores	--
	1 Cable de red cruzado	--
	1 Switch	--
Software		
Herramientas Privativas	Framework XNA de Microsoft	--
	Microsoft Visual Studio Profesional 2008	--
	3D Studio Max	--
	Adobe Photoshop	--
	Frutty Loops	--
Herramientas libres	Blender 2.49	--
	Gimp	--
Capacitación	Libros	$ 250
	Internet	$ 300
	Asesoría	--
	Total	$ 550