Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Laboratorio de Computación Gráfica

Desarrollo de Videojuegos en XNA

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
Por el los Bachilleres
José Luis Andrade y
Francisco Suárez
para optar al título de
Licenciado en Computación

Prof. Ernesto Coto
Prof. Esmitt Ramírez

Caracas, 23 / 05 / 2014

Resumen

Los videojuegos comenzaron siendo una fuente básica de entretenimiento, desarrollados con pocos recursos y más que todo como un hobby. Hoy en día la industria de los videojuegos mueve millones de dólares a nivel mundial, debido al gran interés que despiertan. Existen máquinas especializadas para estos y empresas dedicadas exclusivamente al desarrollo de los mismos.

El impacto social de los videojuegos es muy amplio, desde videojuegos que ayudan a los niños a aprender ciertas habilidades (leer, escribir, operaciones matemáticas, etc.), hasta videojuegos especializados en el entrenamiento de personal militar mediante simulaciones de combates, aunque su mayor impacto se encuentra en la industria del entretenimiento, formando parte del día a día de muchas personas.

Tomando todo esto en cuenta, hemos decidido desarrollar un videojuego de disparos en primera persona multijugador online llamado "PainTint". El objetivo de este documento es analizar todos los procesos involucrados en el desarrollo e implementación de dicho juego bajo la metodología de desarrollo Scrum utilizando la tecnología XNA.

Palabras claves: Videojuego, XNA, Scrum, multijugador, en línea.

Abstract

Video games started out as a basic source of entertainment, developed with few resources and mostly as a hobby. Today video game industry generates billions of dollars worldwide, due to the great interest aroused. There are specialized machines exclusively dedicated to video games development.

Social impact of games is wide, from video games to help children learn certain skills (reading, writing , math, etc. .) , to video games specialized in training military personnel through simulations of combat, although greater impact is in the entertainment industry as part of many people's everyday life.

Taking all this into account, we decided to develop a first person shooter multiplayer online video game named "Paintint ". The aim of this paper is to analyze all the processes involved in the development and implementation of the game under Scrum development methodology using XNA technology.

Keywords: Video game, XNA, Scrum, multiplayer, online.

Agradecimientos

Francisco Suarez:

Primeramente quiero mostrar mi más eterno agradecimiento a mi mamá, quien siempre ha estado ahí, brindándome su apoyo y soporte en todo momento. Siempre me ha demostrado lo grande que puede ser una persona, salir adelante ante las dificultades, a nunca rendirse, a ser una persona correcta y de bien. Es imposible demostrar en palabras lo agradecido que estoy por tenerte como madre y padre, por salir adelante conmigo. Gracias mamá.

A Viviana, mi compañera de vida, mi apoyo en las buenas y en las malas. Gracias por siempre estar a mi lado, viviendo mis buenos y malos momentos, por ayudarme en cualquier cosa que necesitase, y quiero agradecerte especialmente tu inmensa ayuda en este Trabajo Especial de Grado. Sencillamente no se puede pedir más de una persona, gracias amor.

A Venus, quien pasó todas mis noches de programación a mi lado con una pelota de tenis en la boca, sin importar si tenía tiempo o no para jugar, simplemente haciéndome compañía.

Quiero también agradecer a todas aquellas personas que me brindaron su ayuda durante a lo largo de mi carrera, mis compañeros, mis amigos, y a todos los profesores que de alguna u otra manera intervinieron en mi formación, bien sea teniendo la dicha de ver alguna materia con ellos, o simplemente con una charla o un consejo.

A todos los amigos con los que tuve el placer de compartir el Centro de Estudiantes de Computación, donde viví experiencias inolvidables e invaluables, así como a los profesores y estudiantes del Consejo de Escuela de Computación, incluyendo a Jeanette por supuesto. Todas personas excelentes que no solo se conformaban con seguir un fin académico, sino darle el máximo a nuestra casa de estudios y luchar por el beneficio de todos en la facultad.

A mis profesores y preparadores del primer semestre de la licenciatura de Matemática, quienes me dieron la bienvenida a la Universidad, por mostrarme la calidad de docentes que tiene la UCV y motivarme aún más a conseguir lo que en aquel entonces era mi meta, un cupo en la escuela de Computación.

Gracias a todos ustedes por una excelente formación académica, y más importante, por sus enseñanzas de vida.

José Andrade:

Primero que nada, quiero darle las gracias a mis padres, los cuales me guiaron y no desistieron en apoyarme hasta llegar a cumplir esta etapa de mi vida. Desde el principio de mi vida han demostrado que confían en mí y por ello les agradezco todo lo que me han brindado al máximo. A mis hermanos, los cuales fueron fuente de inspiración y aliento en los momentos más difíciles de la carrera, me ayudaron a seguir cuando la cuesta era más empinada, incluso en mis principios de la computación, cuando me ayudaron a aprender cómo era este maravilloso mundo.

A Greisy Guzman, la cual fue mi persona de inspiración y motivo de superación, la cual siempre estuvo para ayudarme en lo que fuese y busco siempre lo mejor de mí. Espero tener la oportunidad y el tiempo para recompensar y demostrar lo mucho que significas para mí.

Al grupo docente del Centro de Computación Grafica (CCG), los cuales nos brindaron sus conocimientos y apoyo para culminar con éxito la última fase de la carrera, especialmente a los profesores Ernesto Coto y Esmitt Ramírez, que sin su constancia y paciencia como tutores, no nos hubiésemos podido graduar.

A todos mis amigos de Modulo 2, los cuales siempre estuvieron dándome apoyo, sin importar que estuviese estudiando, siempre estuvieron pendientes en todo momento.

Al Team, que a pesar de la distancia, espero verlos pronto algún día y relatarles todo lo que logre con el apoyo que me brindaron todo el tiempo que estuve con ustedes.

Índice General

Introducción

Los videojuegos comenzaron siendo una fuente básica de entrenamiento, desarrollados con pocos recursos y más que todo como un hobby. Hoy en día la industria de los videojuegos mueve millones de dólares a nivel mundial, existen máquinas especializadas para estos y empresas dedicadas exclusivamente al desarrollo de los mismos.

El área de investigación referente a los videojuegos ha tomado gran importancia no solo en el área de las ciencias de la computación y el área comercial, sino también en otras áreas como lo son la sociología, la medicina, educación e incluso para el entrenamiento de personal en diversos sectores de la sociedad. El impacto social de los videojuegos es muy amplio, desde videojuegos que ayudan a los niños a aprender ciertas habilidades (leer, escribir, operaciones matemáticas, etc.), hasta videojuegos especializados en el entrenamiento de personal militar mediante simulaciones de combates.

Tomando todo esto en cuenta, hemos decidido desarrollar un videojuego de disparos en primera persona multijugador online. El cual permitió poner a prueba los conocimientos adquiridos a lo largo de la licenciatura.

"Paintint" es el nombre del juego de acción desarrollado como parte del Trabajo Especial de Grado para optar por el título de Licenciatura en Computación. El objetivo de este documento es analizar todos los procesos involucrados en el desarrollo e implementación de dicho juego utilizando la metodología Scrum. En el transcurso de este tiempo, se estudiaron diversos factores que influyen en el desarrollo y planificación de un videojuego bajo varios enfoques, y qué elementos afectan su crecimiento. El resultado fue un videojuego de primera persona basado en el popular juego de Paintball, con módulos bien documentados, lo que permite el desarrollo de nuevos componentes para la aplicación a futuro.

En el Capítulo 1 se explican tanto el problema que se buscó solventar, así como los objetivos generales y específicos a cumplir en este Trabajo Especial de Grado. El Capítulo 2 presenta una reseña de la historia de los videojuegos, desde su concepción en las décadas de los 40, hasta los tiempos modernos. Luego, en el Capítulo 3, se explican brevemente cada género que existe en los videojuegos, con el objetivo de dejar claro los diversos tipos de juegos que existen y como diferenciar cada uno. El Capítulo 4 describe las etapas de desarrollo tradicional, las cuales solo se utilizan de referencia en este trabajo. El Capítulo 5 describe la metodología Scrum, la cual es la base para el desarrollo del videojuego final. En el Capítulo 6 se describe el framework a utilizar, el cual es llamado XNA.

A continuación, en el Capítulo 7, se desglosa el plan de desarrollo del Trabajo Especial de Grado, donde se explica cada iteración y los componentes que se desarrollaron en cada una. En el Capítulo 8 se muestran el diseño e implementación del juego, donde se ve cada artefacto generado, como lo son los diagramas de clases, y como fue desarrollado cada componente del videojuego en sí. Luego de culminar con este trabajo se procedió a realizar un conjunto de pruebas, las cuales están descritas en el Capítulo 9 y para culminar, en el Capítulo 10 se encuentran las conclusiones y posibles trabajos futuros que pueden surgir a partir de este Trabajo Especial de Grado.

Además de todos estos capítulos, se cuenta con dos apéndices, los cuales contienen información específica del videojuego, recopilada en el Documento de Diseño de Juego (GDD) y el instrumento de evaluación utilizado en la fase de pruebas para calificar la calidad del software generado.

Capítulo 1. Propuesta de TEG

1.1 Planteamiento del Problema

En la actualidad se ha formado un reciente interés por el desarrollo de videojuegos en Latinoamérica, y en particular en Venezuela por su creciente demanda y por las facilidades que ofrecen las nuevas herramientas de desarrollado de videojuegos, no solo en el ámbito del entretenimiento sino como herramienta de enseñanza y aprendizaje de nuevas destrezas y habilidades.

El problema que se plantea en este Trabajo Especial de Grado, es la utilización de la metodología Scrum en el desarrollo de videojuegos para la plataforma *Microsoft Windows*, usando la tecnología *XNA*.

Hemos seleccionado la metodología SCRUM ya que es una metodología ágil que permite el desarrollo de productos de software sin enfocarse principalmente en la documentación que se genere, sino al desarrollo del producto final, segmentando la carga de desarrollo en varios elementos del grupo de trabajo.

Seleccionamos *XNA* porque nos permite abstraernos de ciertas funciones que ya están predefinidas, como lo es el manejo de algunos dispositivos de entrada y salida (ratón, teclado, controles *Microsoft* de *Xbox 360* y *PC*, micrófonos, etc.), manejo de estructuras que permiten el despliegue de mallas en 3D, su librería de audio integrada *XACT,* entre otras librerías.

1.2 Objetivo General

Utilización de la metodología SCRUM para el desarrollo de un videojuego no-bélico en la plataforma *Microsoft Windows,* usando la tecnología *XNA*, así como producir un documento que detalle los elementos más importantes al momento de realizar dicho proyecto.

1.3 Objetivos Específicos

- Utilizar la metodología de desarrollo de software SCRUM para el desarrollo del videojuego
- Producir toda la documentación requerida por la metodología: Documento de diseño del juego, lista de iteraciones, elementos concluidos en cada iteración.
- Producir los prototipos de las mecánicas (física, navegación, etc.) usadas en el producto.
- Producción de las versiones alfa, beta y la versión para distribución del videojuego, usando la tecnología *XNA*.

1.4 Plataforma de Hardware y Software

Para la realización de este trabajo especial de grado se planteó como plataforma de hardware mínimo el computador descrito a continuación, con el fin de poder utilizar el sistema operativo Windows XP, el cual es el requisito mínimo para el uso del framework XNA 4.0 y además tener un nivel de procesamiento grafico suficiente para desplegar el videojuego.

Las características sugeridas que debe tener un computador para ejecutar "PainTint" son las siguientes:

- Procesador Intel Pentium Core 2 Duo de 2.6 Ghz
- 2 Gb de Memoria Ram
- Tarjeta de video NVidia GeForce 9800 GTX
- Sistema Operativo Windows XP o superior.
- .Net Framework 4.0
- Games for Windows (solo necesario bajo Windows 8)
- XNA 4.0 Redistributable

Capítulo 2. Historia de los Videojuegos

En este capítulo narraremos los hitos más importantes de la historia de los videojuegos, desde los títulos de los videojuegos y su temática, hasta detalles del hardware que utilizan, o utilizaban, haciendo mención especial a las consolas, ya que son aparatos especialmente diseñados para los videojuegos. Esto sin olvidarnos por supuesto de los videojuegos para computadores personales, que le han dado un gran impulso a la industria del videojuego. Muchas de las personas que compran un computador personal no lo hacen con la idea de utilizarlo para jugar, pero dada su versatilidad, gran parte de ellas instalan videojuegos en su computador personal y en algunos casos, jugar se convierte en su uso principal. Para un mejor entendimiento, dividiremos la historia por décadas.

2.1 Décadas de los 1940´s, 1950´s y 1960´s

En estas décadas se marca el inicio de los videojuegos, algunos de ellos como Tennis for Two valiéndose de ingeniosas ideas para convertir objetos de la época en un videojuego. A continuación describiremos los juegos más importantes de estas décadas.

Cathode Ray Tube Amusement Device (1947)

Mientras se especializaban en el desarrollo de lecturas de señales electrónicas en tubos de rayos catódicos, los físicos *Thomas T. Goldsmith Jr.* y *Estle Ray Mann* tuvieron la idea de crear un juego electrónico inspirado en las pantallas de los radares de la *Segunda Guerra Mundial*. Conectando un tubo de rayos catódicos a un osciloscopio y colocándole botones que servían para controlar el ángulo y la trayectoria de las líneas desplegadas por el osciloscopio (ver Figura 2.1.1), fueron capaces de crear un juego que recreaba el efecto de disparar misiles a varios objetivos.

NIMROD (1951)

Fue diseñado y construido por la compañía británica *Ferranti* y fue expuesto en la *Exhibición de Ciencia del Festival de Gran Bretaña* en el año 1951. Es un computador cuyo único propósito era ejecutar el juego *"Nim*". En este popular juego matemático se tienen pilas de objetos, y cada jugador por turnos toma uno o más objetos de una pila. El juego continúa hasta que el último objeto es tomado, resultando ganador o perdedor, dependiendo de la variante del juego, el jugador que tome el último objeto. Podemos ver en la Figura 2.1.2 un diagrama explicativo del computador NIMROD.

OXO (1952)

También conocido como *Noughts and Crosses*, era una versión gráfica del juego *Tres en Raya* (también se le conoce en Venezuela como *La Vieja* o *Tic-Tac-Toe* en Estados Unidos), creado por *Alexander Douglas* en 1952 en la *Universidad de Cambridge.* En la Figura 2.1.3 vemos una recreación de este juego dentro del emulador EdsacPC.

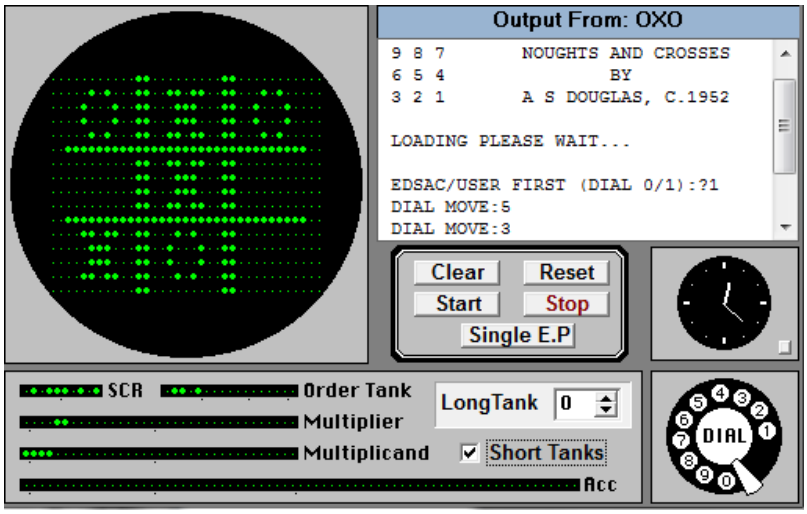


Fig 2.1.3. Recreación del juego *OXO* en el emulador *EdsacPC*.

Tennis for Two (1958)

Fue creado en 1958 por *William Higinbotham*, el cual desarrolló el juego como un entretenimiento para los visitantes del *Brookhaven National Laboratory* (*Laboratorio Nacional de Brookhaven*), en donde él trabajaba. El juego utiliza un osciloscopio como dispositivo gráfico para mostrar la simulación de la trayectoria de una pelota en un campo de tenis. El campo era visto lateralmente, una raya horizontal representaba el piso y una raya vertical más pequeña representaba la red (ver Figura 2.1.4). Fue el primer juego multijugador, permitiendo que dos personas jugaran a la vez.

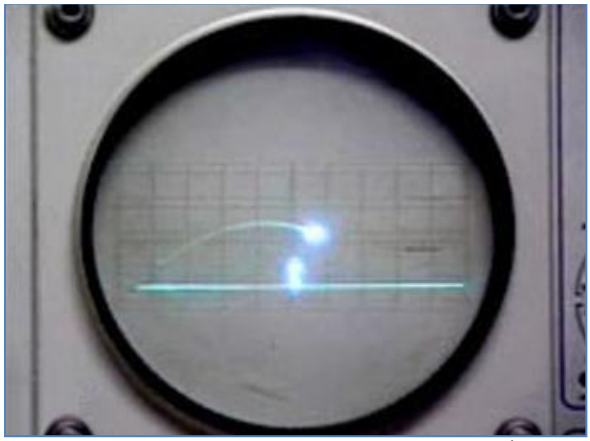


Fig. 2.1.4. Recreación de *Tennis for Two* [1].

Spacewar! (1962)

El juego en principio fue programado por *Steve Russell*, quien utilizando como base unas rutinas de senos y cosenos programadas por *Alan Kotok* se dio a la tarea de crear las naves, sus movimientos, disparos y colisiones, así como las estrellas que se verían en el fondo. La primera versión del juego estuvo lista para febrero de 1962. Luego algunos alumnos del MIT (donde Russell

[1] http://technabob.com/blog/2010/12/14/tennis-for-two-1958-video-game-restored/

creo el juego) hicieron algunas contribuciones, escribiendo subrutinas para el mismo. La versión final estuvo lista para abril de 1962. Fue el primer juego de disparos. En la Fig. 2.1.5 se muestra como lucia la pantalla de *Spacewar!*.

2.2 Década de los 1970´s

En esta década, surgen y llegan a tener gran éxito las máquinas Arcade[1]. Se crean las primeras videoconsolas, abriendo un nuevo mercado de consumidores, dando cabida a dos generaciones de videoconsolas. También se crean videojuegos para computadores personales así como también para mainframes en las universidades.

Galaxy Game (1971)

Bill Pitts reprogramó el juego *Spacewar!* con el objetivo de crear una máquina activada por monedas, agregándole naves enemigas, y limitando las oportunidades en las que el jugador podía ser alcanzado por balas o chocar contra otras naves. Así, al quedarse sin combustible la nave, o al perder todas las oportunidades, se acaba el juego, limitando así el tiempo. *Tuck* por su parte, diseño el gabinete en el que se colocaría el sistema.

Fig. 2.2.1. Primera versión de *Galaxy Game*.[2].

Computer Space (1971)

Nolan Bushnell, luego de quedar maravillado tras jugar *Spacewar!,* entonces planifico crear un juego que corriera simplemente sobre transistores y diodos que se conectaran a un televisor, colocar todo en un gabinete y entonces tendría un videojuego de bajo costo.

[1] Arcade: es el término genérico de las máquinas recreativas de videojuegos. Son similares a las máquinas de Pinball y a las máquinas tragamonedas de los casinos, pero debido a que no son juegos de azar ni de apuestas (ya que se basan en la destreza del jugador) por lo general no tienen las limitaciones legales de éstas.
[2] http://www.arcade-history.com/?n=galaxy-game-first-version&page=detail&id=26984

Fig. 2.2.1. Gabinete de *Computer Space* (modelo azul) [1].

Odyssey (1972)

En 1966 el ingeniero de televisión *Ralph Baer*, quien trabajaba para la empresa *Sanders Associates*, creó un juego de video simple de dos jugadores que podía ser desplegado en cualquier televisión estándar llamado *Chase* (*persecución* en español), en donde dos puntos se perseguían el uno al otro en la pantalla. Este prototipo, llevado de la mano de la compañía Magnavox, se convirtió en la primera consola.

Fig. 2.2.4. *Odyssey* y su juego *Tennis* [2].

[1] http://www.abadiadigital.com/articulo/computer-space-la-primera-maquina-recreativa-de-la-historia/

[2] http://www.gamasutra.com/features/20070119/odyssey.jpg

PONG (1972)

El juego en un comienzo fue lanzado en 1972 en gabinetes Arcade, como se muestra en la Fig. 2.2.5, con un televisor en blanco y negro y toda la lógica del juego incorporada en un chip. El Arcade fue todo un éxito y dada su gran popularidad, *Atari* decidió convertirlo en un sistema de entretenimiento para el hogar. Se creó así en 1975 *Home PONG*, el cual era un aparato electrónico que se conectaba al televisor y permitía jugar únicamente *PONG*. Tenía dos perillas que permitían controlar el movimiento de las barras que representaban las raquetas.

Atari 2600 (1977)

Atari pensó en una consola de videojuegos flexible, que permitiese jugar más de un único videojuego. Así surgió la *Atari VCS* (*Video Computer System*, o en español, *Sistema de Video-computadora*), luego renombrada *Atari 2600* con la salida del nuevo modelo *Atari 5200* en 1982.

Fig. 2.2.7. *Atari 2600*[1].

Space Invaders (1978)

Fue un juego creado por *Toshihiro Nishikado* para la compañía Japonesa *Taito*. El objetivo del juego era acabar con un grupo de alienígenas que se acercaban a la tierra, con la ayuda de un cañón laser. En la pantalla se apreciaban colores, gracias a la superposición de láminas de colores, aunque el monitor del gabinete era en blanco y negro (ver Fig. 2.2.8).

Activision (1979)

En la compañía *Atari*, algunos desarrolladores disconformes renunciaron y cofundaron la empresa *Activision* en 1979, la primera empresa en publicar videojuegos para terceros. Los fundadores de *Activision* fueron los exprogramadores de *Atari: David Crane*, *Larry Kaplan*, *Alan Miller* y *Bob Whitehead*, junto con el ejecutivo de la industria musical *Jim Levy*. La empresa continúa desarrollando videojuegos en la actualidad.

[1] http://www.gamertell.com/gaming/comment/atari-2600-inducted-into-the-national-toy-hall-of-fame

2.3 Década de los 1980´s

En esta época se unen a las consolas de segunda generación el *Intellivision*, que fue fabricado por *Mattel* en 1980 y el *ColecoVision* que fue introducido al mercado por la compañía *Coleco* en 1982. Estos dos incluían las características del *Atari 2600* como los son los cartuchos independientes y tener procesadores de 8-bits.

Esta es la década donde se definen la mayoría de los géneros de los videojuegos, ya que las nuevas compañías comienzan a producir más y más títulos, y cada uno de estos cae bajo un reglón especifico. Para resumir una lista de géneros y títulos emblemáticos tenemos la siguiente tabla:

Géneros	Videojuegos
Aventura	*Zork* (1980), *Mistery House* (1980), *Maniac Mansion* (1987)
Beat'em Up	*Kung-Fu Masters* (1984), *Renegade* (1986)
Pelea	*Karate Champ* (1984), *Street Figther* (1987)
Laberintos	*Pac-Man* (1980), *3D Monster Maze* (1981)
Plataformas	*Donkey Kong* (1981), *Jump Bug* (1981), *Mario Bros.* (1983), *Prince of Persia* (1989)
Carreras	*Turbo* (1981), *Pole Position* (1982)
Rol	*Akalabeth* (1980), *Ultima* (1981), *The Legend of Zelda* (1986), *Dragon Quest* (1986), *Final Fantasy* (1987), *Phantasy Star* (1987)

Además de las videoconsolas y los Arcades, aparecen en el mercado una serie de videoconsolas de mano que permitían jugar un juego sin necesidad de grandes equipos. La primera compañía en hacer esto es *Milton Bradley Company* que crea un sistema llamado *Microvision* en 1979, pero por la falta de juegos el sistema pierde interés en el mercado. En cambio la compañía *Nintendo* creó los *Game & Watch* (ver Fig. 2.3.3), una serie de juegos de bolsillo basados en juegos famosos de Arcades que dieron a la popularización de estos juegos y muchos clones de estos *Game & Watch* fueron creados.

Tercera Generación de videoconsolas

Para completar esta década, aparece la tercera generación de videoconsolas, que a pesar de aún poseer tecnología de 8-bits, crearon un estándar que fue evolucionando en las generaciones siguientes, que es la manera de jugar, ya que desaparecen las palancas y similares para dar paso a un control genérico, con flechas de dirección de cuatro u ocho direcciones, y botones de acción, siendo dos botones el estándar. Otra gran diferencia de esta generación con las anteriores es la inclusión de circuitos de sonido especiales y la gran mejora de los gráficos que había en los juegos de ese entonces.

Entre las consolas que aparecen en esta generación se encuentran la *NES* (*Nintendo Entretainment System*), la cual salió al mercado en 1985 y la *Sega Master System* que surgió en 1986, al igual que la *Atari 7800*.

2.4 Década de los 1990´s

El costo del hardware fue disminuyendo considerablemente y comenzaron a producirse los primeros procesadores *Intel 386* y *486* para uso doméstico, lo que provoco que el público tuviera mejores equipos no solo como herramientas de trabajo, sino como dispositivos para jugar. Esto también incluyo aproximadamente por el año 1995, la aparición de las primeras tarjetas 2D/3D, las cuales permitían acelerar los cálculos de geometrías de 2 y 3 dimensiones realizados por el computador, obteniendo una mejor calidad y velocidad de las imágenes mostradas en el monitor.

También comienzan a surgir los primeros juegos en línea, donde dos o más jugadores interactúan en el mismo juego.

Otro género que nació en esta época es el de disparos en primera persona(o por sus siglas en inglés FPS – *First Person Shooter*) con la salida del juego *Wolfenstein 3D* en 1992, y en ese mismo año su secuela, *Wolfenstein: Spear of Destiny* . En 1996 debutó la franquicia Quake.

Cuarta y Quinta Generación de videoconsolas

En esta época surgieron dos generaciones de consolas. La cuarta generación de videoconsolas inicia en 1989 con la salida de la *SEGA Genesis* y en 1991 con la *Súper NES* (*Nintendo Entretainment System*) de *Nintendo*. Otras consolas de esta generación fueron la *Neo Geo*, la *TurboGrafx-16* y la *Commodore Amiga CDTV*. Aspectos que remarcaron esta generación son los procesadores de 16-bits que tienen integrados y la inclusión de nuevas tecnologías que ya se habían desarrollado anteriormente para PC, como lo es la de los gráficos 3D en los nuevos juegos de consolas.

La quinta generación de videoconsolas fue un poco mixta, ya que incluye consolas de 32-bits (*Sega Saturn* en el 1994, *Atari Jaguar* en el 1993, y la Sony *PlayStation* en el 1994) y una consola de 64-bits (la *Nintendo 64,* que salió a la venta en 1996). La novedad de estos es que incluyen verdaderas imágenes 3D, cosa que era muy básica y burda en la cuarta generación.

2.5 Década del 2000 hasta la actualidad

En esta época no hay tanta diversidad de cambios como en épocas anteriores, pero son cambios de gran significancia. Continúa la sexta generación de videoconsolas, con el lanzamiento en el año 2000 de la *Sony PlayStation 2*. Luego de esta, Microsoft incursiona en el mercado con su Xbox, un poderoso equipo basado en un procesador Pentium III. Por último está la *GameCube*, que salió al mercado en el año 2001, la cual atrajo al mercado infantil y familiar sobre todo.

Luego de algunos años, surge la séptima generación de videoconsolas, que es la generación actual, e incluye principalmente a tres representantes: la *Playstation3* de *Sony*, la *Xbox 360* de *Microsoft*, y el *Nintendo Wii.*

En cuanto a las consolas caseras, la *PlayStation3(PS3)* salió al mercado en el año 2006. *Microsoft* siguió el desarrollo de su éxito y su sucesor fue la *Xbox 360*, la cual salió al mercado en el 2005. Uno de los atractivos del *Xbox 360* era que traía incluido, al igual que en el *PlayStation 3*, un sistema de juego online, en este caso llamado *Xbox Live*, el cual permite descargar demos, películas, series y juegos completos desde la red de *Microsoft*.

Por otro lado, *Nintendo* aposto a un mercado completamente distinto y se enfocó en el público familiar al lanzar su consola *Wii* en el 2006, la cual es inferior tecnológicamente con las otras consolas de su época, por poseer un procesador *PowerPc* "*Broadway*" de 32-bits. A pesar de no poseer mucha capacidad de procesamiento, esta gonsola gano un gran margen del mercado gracias a que *Nintendo* mostró un sistema revolucionario donde, gracias a su novedoso *Wii Remote* o *Wiimote*, el jugador se siente inmerso en el juego al seguir los movimientos de los juegos.

Surge también una nueva generación de consolas portátiles, como lo son el *PlayStation Portable* (*PSP*, 2004) y el *Nintendo DS* (2004). En el año 2009 salió la secuela de la consola de sony, la *PSP Go!*. La *Nintendo DS* sale al mercado en el 2004 para mejorar el equipo portátil anterior, el *Game Boy Advance*, y revolucionan completamente el género al agregar 2 pantallas, una de ellas táctil, con lo que permite un nuevo nivel de interacción con los usuarios.

En el año 2012, surge la octava generación de videoconsolas, destacándose de nuevos los tres competidores principales. Nintendo fue el primero en presentar su WiiU, a finales del 2012, con la novedad de una pantalla táctil como control adicional para la consola. Sony presento su PlayStation 4 a finales del 2013, el cual mostro como cambio radical, el pase de arquitectura de sus procesadores Cell a un procesador AMD, por lo cual permite ejecutar instrucciones x86-x64, y es mucho más atractivo para los desarrolladores actuales. Microsoft mostro su XboxOne también a finales de 2013, el cual integra un centro de entretenimiento completo dentro de la misma consola.

Así termina el recuento de la historia de los videojuegos. Este capítulo no cubre todo lo ocurrido desde 1947 hasta la fecha, pero nos brinda una buena idea de los puntos más resaltantes.

Capítulo 3. Géneros de los videojuegos

El punto de partida para desarrollar un videojuego es el concepto del videojuego, y de la mano con el concepto va el género al que apunta el videojuego. En la actualidad muchas empresas que elaboran videojuegos fusionan dos o más géneros populares como punto inicial de sus nuevos proyectos, buscando asegurar un número alto de ventas para el nuevo videojuego.

En este capítulo explicaremos los géneros más populares de videojuegos. Sin embargo, es importante aclarar que son pocos los videojuegos que encajan perfectamente en un único género. Prácticamente todos los juegos se pueden ver como una combinación de estos.

3.1 Agilidad Mental

Son juegos cuyo objetivo es desarrollar la agilidad mental del jugador, presentándole rompecabezas que van aumentando en grado de complejidad a medida que avanza el juego.

Un ejemplo es *World of Goo*, juego en el cual el jugador debe construir una estructura que le permita alcanzar su objetivo, sorteando obstáculos y sin que esta se derrumbe. Otros representantes de este género son *Tetris*, *Brain Age*, *Dr.Mario*, *Brain Academy*, *Scribblenauts* y *Zuma*.

3.2 Arcade

Son videojuegos con reglas simples, con poca o ninguna historia, largos y con acciones repetitivas. Se caracterizan por tener un gran nivel de jugabilidad para enganchar al jugador, muchas veces sacrificando el realismo. Algunos ejemplos son: *Space Invaders*, *Galaxian*, *Pac-man*, *Breakout*, *Tetris*, *Mario Super Sluggers* y prácticamente cualquier juego de la saga *Mario* que involucre deportes.

3.3 Aventura

Este tipos de juegos no están definidos por su trama, son más bien una forma de jugar, en donde el jugador debe resolver rompecabezas mediante la interacción con otros personajes o con el entorno. Si en el juego se incluyen confrontaciones, se llaman juegos de Acción-Aventura. Ejemplos de este género son los clásicos *Adventure* y *Myst* y los más recientes *The Longest Journey*, y las sagas *Leisure Suit Larry* , *Ace Attorney* y *Resident Evil* (en su trama principal, es decir los juegos del 1 al 5).

3.4 Carreras

Consisten en alcanzar primero que el resto de los jugadores un punto de llegada. Generalmente se utilizan vehículos para este fin, como carros, motos o incluso animales. Algunos juegos introducen ítems en la trayectoria que le otorgan cierta ventaja al jugador que los recoja, como darle más velocidad, o dándole la posibilidad de tenderle una trampa al resto de los jugadores. Ejemplos de este género son *Need For Speed*, *Gran Turismo*, *Ridge Racer*, *GRID*, *DIRT*, *Mario Kart* y *Crash Team Racing.*

3.5 Deportes

Son videojuegos que emulan deportes reales como el fútbol, béisbol, básquetbol, tenis, deportes extremos, etc. No siempre son basados en simulación, algunos videojuegos le agregan o eliminan elementos al deporte real, por ejemplo *Battle Tennis* de la empresa *Gasp*, se basa en el tenis pero también incorpora elementos del género de peleas. Ejemplos de este género son las sagas *Pro Evolution*, *FIFA*, *NBA 2K11*, *NHL*, *Madden*, *Grand Slam Tennis*, *Top Spin* y *MLB 2K10*.

3.6 Educativos

En este tipo de videojuegos la intención es dejar una enseñanza al jugador mientras este se entretiene, como por ejemplo ayudarlo a desarrollar una habilidad, enseñarle sobre un tópico en específico o ayudarlo a entender un hecho histórico.

Un ejemplo bastante particular de este género es *The Typing of the Dead*, juego en el cual debemos teclear correctamente las palabras que aparecen en pantalla para acabar con una horda de zombis que nos atacan. Otros exponentes de este género son: *InLiving, Urban Jungle, My Virtual Tutor, Spelling Challenges and more, Personal Trainer: Math, Food Force,* la saga *Medal of Honor, America's Army, President Forever 2008 + Primaries*.

3.7 Estrategia

Son juegos en los que son fundamentales la inteligencia y la planificación para utilizar los recursos de los que se dispone. Colocan al jugador en una posición de mando, dándole el cargo de presidente de un país, alcalde de una ciudad, director de un cuerpo de infantería o inclusive el de un emperador. Existen dos grandes subgéneros, la estrategia por turnos y la estrategia en tiempo real.

Algunos ejemplos son las sagas *Age of Empires*, *Civilization*, *StarCraft*, *Command & Conquer* y *Warcraft* (exceptuando *World of Warcraft*) o los juegos *Caesar IV*, *Final Fantasy Tactics* y *Diplomacy*.

3.8 Juegos de Mesa

Son adaptaciones de los juegos de mesa al mundo digital. Prácticamente todos los juegos de mesa han pasado por esta conversión. Ejemplos de esto son los juegos de cartas (siendo los más populares el *Poker*, y el *Blackjack*, ambos en su versión online), *Dominó, Monopoly, Yatzee, Scrable* y *Battleship*.

3.9 Pelea

El objetivo es vencer a otro personaje (bien sea manejado por otro jugador o por el computador) en combates cuerpo a cuerpo. Generalmente la pelea se desarrolla en un área reducida, por ejemplo un ring de boxeo, un octágono, etc. En algunos títulos se incorpora además el uso de armas blancas o ataques mágicos. Algunos exponentes de este género son las sagas *Street Fighter*, *Mortal Kombat*, *Tekken*, *Soul Calibur*, *Ready 2 Rumble*, *Super Smash Bros*, *Ultimate Fighting Championship* y *Fight Night*.

3.9.1 Pelea Beat'em Up

El nombre de este género traducido al español seria "dales una paliza". A diferencia de los juegos de pelea, el jugador debe recorrer un escenario luchando cuerpo a cuerpo contra oleadas de enemigos y generalmente al final de cada nivel aparece un jefe final[1]. Muchos *Beat'em Up* permiten que varios jugadores participen de forma cooperativa (lo cual es gran parte del atractivo de estos juegos). Algunos ejemplos son: *Double Dragon, Final Fight, Fighting Force, Urban Reign* y *God of War*.

3.10 Plataformas

El jugador debe sortear en su camino una serie de plataformas, escaleras, obstáculos y enemigos para llegar a su objetivo. Ejemplos clásicos son *Donkey Kong* y *Super Mario Bros* de la compañía *Nintendo*, o *Sonic the Hedgehog* de *Sega.*

3.11 Preguntas

Se basan en el formato de preguntas y respuestas, en donde el jugador debe contestar correctamente para poder seguir avanzando en el juego. Algunos ejemplos son: *Who Wants to be a Millionaire?*, *Scene It?*, *Brain Quest* y *Are You Smarter Than a 5th Grader?*.

3.12 Disparos

Mejor conocido por su nombre en inglés *Shooter*. En este género el jugador debe vencer a sus oponentes por medio de disparos. En casi todos los juegos de este género se añaden objetos extra que sirven para vencer al enemigo, como armas blancas o granadas. La gran mayoría son de corte bélico, aunque existen shooters desde los más sangrientos hasta otros en los que se arrojan bolas de nieve o pintura.

Existen dos grandes vertientes en el género, los shooters en primera persona, mejor conocidos como FPS (por sus siglas en inglés "*First Person Shooter*") y los shooters en tercera persona o TPS ("*Third Person Shooter*").

3.12.1 FPS

En los FPS el jugador se ve inmerso en el juego al ver la acción en primera persona, viendo el ambiente desde la perspectiva del personaje, dándole mayor precisión a la hora de disparar. Generalmente la trama del juego no es muy elaborada (en algunos casos inexistente) y en cambio se aboga por una gran calidad gráfica. Este tipo de juegos requieren del jugador una buena habilidad motriz y buenos reflejos. Algunos ejemplos son las sagas *Wolfenstein*, *Doom*, *Quake*, *Half Life*, *Unreal*, *Medal of Honor*, *Halo*, *Ghost Recon*, *Call of Duty* y *Crysis* o los juegos *Renegade Paintball*, *Wolfteam* y *Alliance Of Valiant Arms*.

[1] Jefe final: Enemigo más difícil de vencer que los enemigos normales. Generalmente los ataques del jugador le generan menos daño y sus ataques son más potentes.

3.12.2 TPS

En estos juegos la acción se ve en tercera persona, lo cual a diferencia de los FPS no involucra tanto al jugador con el personaje y se sacrifica precisión en los disparos, en cambio esto nos permite observar e interactuar mejor con el escenario. Exponentes de este género son *Tomb Raider*, *Gunz The Duel* , *Max Payne, Army of Two* y *Gears of War*.

3.12.3 Shoot'em Up

También son juegos basados en disparar, pero a diferencia de los dos subgéneros anteriores, la pantalla está repleta de enemigos y disparos la mayor parte del tiempo. En la mayoría de estos juegos el avance en el escenario es automático o lineal, es decir no permite tomar decisiones sobre la dirección a seguir. Ejemplos de este subgénero son *Contra*, *R-Type*, *Metal Slug*, *Time Crisis* y *Area 51*.

3.13 Sigilo

En estos videojuegos la prioridad es pasar desapercibidos y tratar de evitar en lo posible confrontaciones con el enemigo. También son llamados juegos de espionaje. La historia de estos juegos suele ser impresionante, desenvolviéndose a medida que avanzamos, semejante a una novela. Grandes exponentes son *Metal Gear*, *Splinter Cell*, *Syphon Filter*, *Hitman* y *Tenchu*.

3.14 Simuladores

Son videojuegos que simulan actividades de la vida real, intentando que el jugador sienta que es real lo que está realizando, o al menos lo más parecido posible a la realidad. Este género por su naturaleza debe ser combinado con otro género, por lo que existen simuladores de todo tipo, desde simuladores de baile, hasta simuladores de vida, en los cuales se simulan las etapas de la vida humana, desde las actividades cotidianas como comer y dormir, hasta la elección de los estudios a realizar. Algunos simuladores son creados para entrenamiento de personal, por ejemplo, simuladores de vuelo, manejo e incluso de combate.

Algunos ejemplos de este género son: *America's Army* (combate), *Flight Simulator* (vuelo) y *Test Drive* (manejo).

3.15 Rol

También llamados RPG (por su nombre en inglés, *Role Playing Game*). Tienen su origen en los juegos de mesa de rol como *Calabozos y Dragones*, de los cuales heredaron gran parte de su terminología. En este estilo el jugador controla por medio de comandos a uno o varios personajes, los cuales tienen atributos como fuerza, velocidad, vida, etc. Cada personaje pertenece a una clase (magos, arqueros, espadachines, etc.) y a su vez tiene habilidades especiales (hechizos, combos, etc.) y objetos propios del personaje, con lo cual se pueden crear gran variedad de personajes dentro del mismo juego.

Tienen una historia larga y profunda. El jugador se ve envuelto en búsquedas de objetos u otros personajes (amigos o enemigos), en las cuales tiene que recorrer grandes mapas sorteando las dificultades que se susciten (generalmente batallas contra otros personajes). En los juegos de Rol el ganar o perder se basa más en el desarrollo del personaje que en la coordinación motriz del jugador.

Grandes exponentes del juego de Rol son por ejemplo *World of Warcraft* o las sagas *Final Fantasy* y *Diablo*.

3.16 Terror de Supervivencia

Mejor conocido por su nombre en inglés *Survival Horror*. Son juegos en los cuales el objetivo principal es permanecer con vida mientras se desenvuelve la historia del juego y se tiene que resolver algún misterio o cumplir con un objetivo dado, evitando ser atacado por los enemigos (generalmente muertos vivos o seres paranormales). Este tipo de juegos tienen una atmósfera de terror psicológico, buscando mantener al jugador concentrado y en tensión, interrumpida solamente por sucesos repentinos que pretenden asustar al jugador.

Resident Evil es considerado el juego que le dio el impulso inicial a este género (aunque no fue el primero). Grandes exponentes de este género son las sagas *Silent Hill*, *Parasite Eve*, *F.E.A.R.*, *Proyect Zero*, *Obscure* , *Left 4 Dead*, y *BioShock*.

Capítulo 4. SCRUM

En este capítulo se describirán las bases teóricas que fundamentan la metodología de desarrollo de software utilizada para la realización del Trabajo Especial de Grado.

4.1 El Manifiesto Ágil

En marzo de 2001, un grupo de expertos en el desarrollo del software se reunieron en Salt Lake City para discutir sobre los procesos que ya existían en el desarrollo del software y los que planteaban como nuevas soluciones. Ahí, se acuño el término "Método Ágil", como una alternativa a los modelos formales ya existentes, los cuales se consideraban excesivamente "pesados" y rígidos por su carácter normativo y fuerte dependencia a la panificación en vez de al producto final.

La reunión tuvo como conclusión cuatro postulados que quedaron como principios para definir el denominado "Manifiesto Ágil" [2]:

"Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

- ***Individuos e interacciones** sobre procesos y herramientas.*
- ***Software funcionando** sobre documentación extensiva.*
- ***Colaboración con el cliente** sobre negociación contractual.*
- ***Respuesta ante el cambio** sobre seguir un plan.*

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda."

4.2 Scrum

Scrum es una herramienta de trabajo que nace de la necesidad de tener un modelo de desarrollo ágil para la producción de proyectos complejos. Estos proyectos complejos podemos asociarlos con el desarrollo de videojuegos ya que, históricamente hablando, los videojuegos han crecido exponencialmente en cuanto a costos y tiempos de desarrollo. Los modelos que se utilizaban tradicionalmente, más que todo orientados a un estilo cascada, como el presentado en la Figura 4.2.1, se enfrentaron al problema fundamental de que estos siguen un patrón de acierto o fallo, es decir, buscan planificar el proyecto y luego de desarrollarlo y culminarlo, le es mostrado al cliente potencial y se analiza si se tuvo éxito o no.

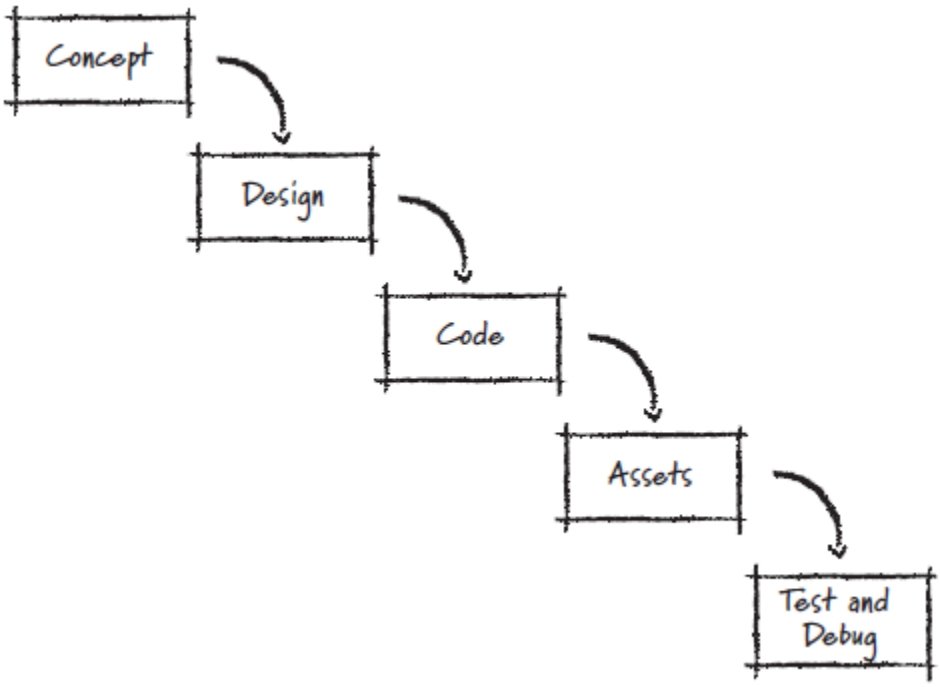


Fig. 4.2.1 Desarrollo basado en cascada [6].

En los primeros videojuegos esto no era tan importante, ya que un juego se podía desarrollar en poco tiempo, dadas las bajas expectativas que existían en ese entonces y el impedimento tecnológico de esa generación de videojuegos. Pero actualmente el costo y la cantidad de contenido que contiene un videojuego moderno son proporcionales al nivel de cómputo de las consolas y computadores actuales. El tiempo necesario para desarrollar un videojuego actual de alto contenido siguiendo el modelo cascada, sería muy elevado y riesgoso en el caso que dicho proyecto no se vendiera como se esperaba. Se desperdiciaría todo ese tiempo en lugar de enfocar esos esfuerzos en otra dirección en el momento adecuado.

La manera de trabajar con Scrum consiste en iteraciones o "*sprints*" de aproximadamente dos a cuatro semanas donde se desarrollan diversos elementos del juego en cuestión. Luego de cada *sprint* se analiza cuanto avance ha tenido el proyecto y que cambios deben hacerse para cumplir con el objetivo final. Podemos ver un diagrama de cómo es el ciclo de vida de un proyecto realizado con Scrum en la Figura 4.3.1.

4.3 Sprint o Iteraciones

Cada iteración contiene un resumen de las actividades realizadas en dicha iteración, una lista de los elementos desarrollados (llamados en inglés "*Sprint Backlog*") y de cada uno de estos cómo se subdividió para determinar qué fue alcanzado de ese objetivo. Estos elementos son tomados de una lista llamada Backlog de Productos (llamada en inglés "*Product Backlog*"), en el cual están todas las características, derivadas de las historias de usuario, que serán implementadas en el producto final. Esta lista puede ser modificada entre cada *sprint* con el fin de agregar, modificar o eliminar

elementos para darle más calidad al producto, teniendo cuidado de no saturar la lista con demasiados elementos. En cada iteración se toman las actividades que tengan más prioridad, y entre cada *sprint* se pueden definir otras nuevas.

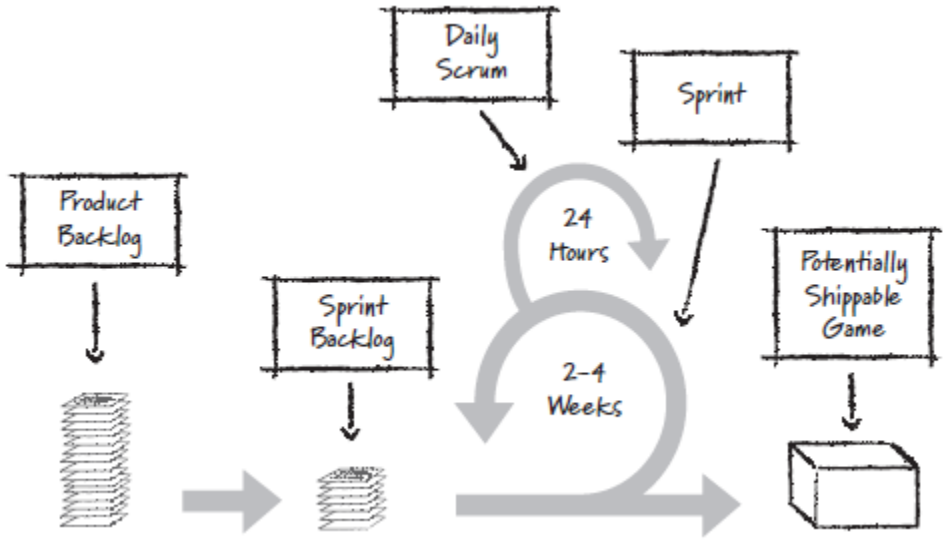


Fig. 4.3.1 Ciclo de vida de un proyecto en SCRUM [9].

El ciclo de vida de un *sprint* es el siguiente:

1. Se busca en el *Product Backlog* el elemento de mayor prioridad.
2. Se identifican y se estiman todas las tareas para completar ese elemento.
3. Si las horas/tareas exceden la capacidad del equipo de trabajo se devuelve el elemento y se toma el siguiente elemento del *Product Backlog*. Se regresa al paso 1.
4. Se agregan las tareas definidas al *Sprint Backlog*.
5. Si la capacidad del equipo llego a su límite se procede a empezar la iteración, en caso contrario se regresa al paso 1.
6. Luego del tiempo de desarrollo (2 a 4 semanas aproximadamente) se evalúan los resultados de los elementos a desarrollar. Se analizan los avances e impedimentos que ocurrieron en esta iteración.
7. El cliente o dueño del producto (cuyo perfil es explicado más adelante) analiza que elementos faltan en el producto final y agrega nuevas historias o actualiza las historias existentes en el *Product Backlog*. En caso que aún queden elementos que desarrollar se regresa al paso 1, de lo contrario se finaliza el proceso de desarrollo.

4.4 Historias de usuarios

Estas son las características que se desarrollaron en la planificación del producto. De ellas podrán derivarse nuevos requerimientos entre iteraciones y dividir estas características en elementos más granulares y fáciles de implementar.

Las historias de usuario están escritas en un lenguaje que sea entendible por todos, es decir, tanto como para el cliente, que es el que las elabora, como también por los miembros del equipo, que son los que implementan estas historias.

Para desarrollar estas historias se sigue el formato "Como <rol de usuario>, deseo que <objetivo> [razones del objetivo]" donde:

- <rol de usuario> es el protagonista o jugador que se beneficiara de la historia.
- <objetivo> es el fin de la historia, la cual puede ser una característica, función o herramienta del juego.
- <razones> son los motivos por los cuales se desea esta historia.

Un aspecto importante es el de los roles. Normalmente el rol se aplica a un individuo que utilizará el producto o juego final de forma genérica, pero se puede agregar más detalle y prioridad especificando que tipo de jugador tomara ventaja de esa característica. Además de esto, podemos incluir en las historias de usuario a miembros del equipo de desarrollo que se beneficiarían de alguna característica en particular para el avance del proyecto. Por ejemplo, un diseñador podría agregar la historia "Como diseñador, quiero que el exportador de animaciones sea optimizado para poder crear más animaciones". En esta historia, el diseñador requiere que una característica del juego tenga cierto nivel de calidad para poder mejorar su trabajo de diseñar las animaciones del juego.

Luego de esto, por cada historia de usuario se pueden crear condiciones de satisfacción (como se ve en la Figura 5.4.1), las cuales son versiones más detalladas de las historias de usuario, con el fin de agregar un detalle particular a una historia sin tener que reescribirla.

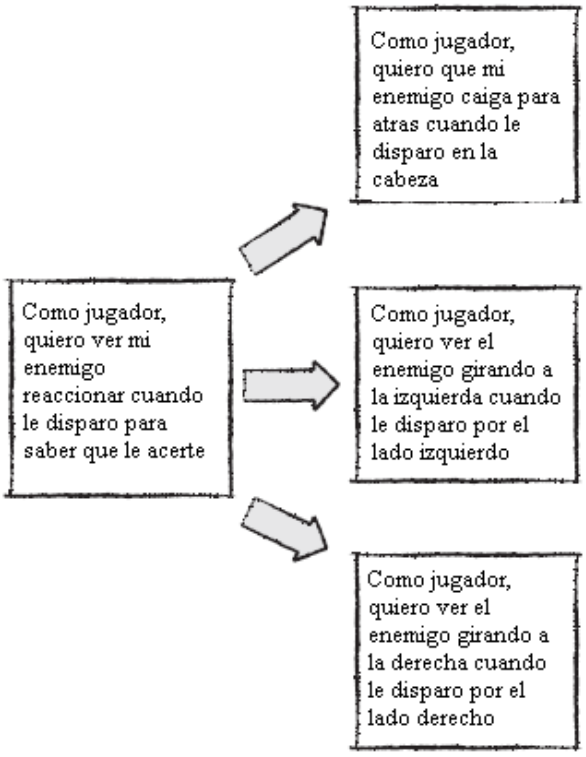


Fig. 4.4.1 Historia de usuario con sus condiciones de satisfacción.

Después de definir las historias de usuario, se procede a traducir estas en características que irán en el *Product Backlog* y en el *Sprint Backlog*. Al principio de cada *sprint*, el *Product Owner* tomará las historias de usuario existentes y agregará los *Product Backlog* necesarios y volverá a evaluar la prioridad de los aún existentes. Luego de esto el *Scrum Master* selecciona los elementos en los que trabajará en esta iteración y le dará a cada elemento un conjunto de tareas que podrán ser tomadas por el equipo de desarrollo (como se representa en la Figura 4.4.2), las cuales conforman el *Sprint Backlog*.

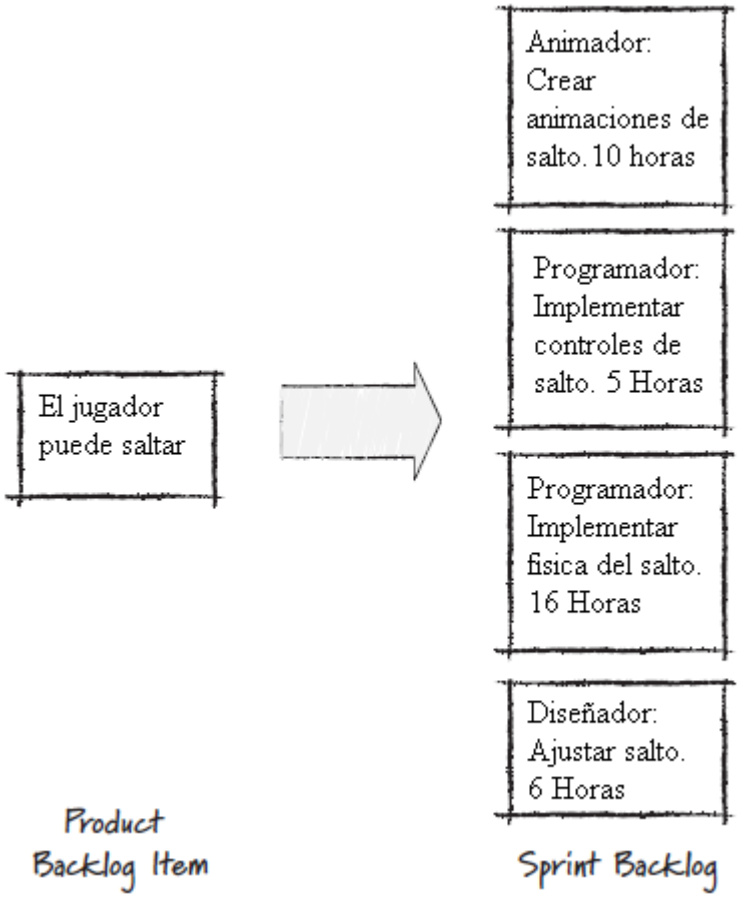


Fig. 4.4.2 Transición de un *Product Backlog Item* en un *Sprint Backlog.*

4.5 Roles en SCRUM

En la metodología SCRUM existe un conjunto de roles que permiten distribuir las responsabilidades entre varias personas, a pesar que una sola persona puede ocupar varios roles a la vez.

4.5.1 Scrum Master

El *Scrum Master* es el responsable de hacer cumplir las prácticas de SCRUM y orientar al equipo de trabajo en estas prácticas con el fin de mejorar el ambiente de trabajo. Esto no quiere decir que el *Scrum Master* es el líder del equipo, sin embargo, este estará monitoreando el avance del equipo, ubicando posibles impedimentos, facilitando la planificación y revisión del trabajo realizado y creando un canal de comunicación entre los inversionistas y el equipo.

Usualmente, cuando los proyectos están divididos en varios equipos, los *Scrum Masters* no pertenecen a ningún grupo de desarrollo, para asegurar que sus responsabilidades en el equipo no estorben con sus responsabilidades al momento de observar a los demás grupos. Pero en proyectos donde hay pocos equipos (hasta dos o 3 equipos como máximo) es posible ver al *Scrum Master* como miembro de algún de estos equipos.

4.5.2 Dueño del producto (*Product Owner*)

El dueño del producto será el encargado de comunicar la visión del juego (lo que se desea obtener del proyecto y/o juego) y buscar maximizar el retorno de inversión (*Return of Investment* o ROI). Este individuo representara no solo a la empresa productora, sino también las ideas que los clientes tienen acerca del producto final, dando así opiniones y expectativas acerca del avance del juego para agregar nuevas características.

El dueño del producto será responsable de los siguientes eventos:

- Manejar el ROI del juego y otras métricas que representen el éxito del juego final.
- Establecer la visión del juego que será común entre clientes e inversores.
- Saber qué características se van a hacer y en qué orden se deben hacer. Esto se consigue directamente a través del *Product Backlog*.
- Crear la planificación de las entregas y las fechas de publicación correspondientes. Esto incluye el participar en los *Sprint Reviews* y aceptar o rechazar los resultados de dichas entregas. Con esta información puede planificar próximas iteraciones e incluir nuevos elementos a las iteraciones.

4.5.3 Equipo de desarrollo

El equipo es el conjunto de personas que se encargan de realizar todas las peticiones que haya en el *Product Backlog*. Este equipo consiste en varios expertos en diversas áreas del desarrollo de videojuegos y los cuales se enfocaran en las características que seleccionaron implementar en dicha iteración. Esto es importante recalcarlo, ya que en estudios o proyectos de gran alcance, los equipos de trabajo se dividen en grupos similares al descrito, en vez de separar a los desarrolladores por áreas de trabajo. Esto es con el fin de aumentar la cohesión entre los miembros del equipo y evitar retrasos al momento de transmitir la información entre diversos departamentos.

Un aspecto importante de los equipos de desarrollo es la cantidad de miembros que posee. Un grupo de trabajo, según la literatura de Scrum [10], está conformada idealmente por siete personas, lo cual en el desarrollo de videojuegos queda corto al momento de incluir todos los posibles cargos. Un ejemplo de esto es el siguiente equipo:

- Dos artistas de niveles
- Un artista de texturas
- Un animador
- Un diseñador de audio
- Un artista conceptual
- Un diseñador de niveles
- Un diseñador de jugabilidad
- Un programador de jugabilidad
- Un programador de gráficos
- Un programador de inteligencia artificial

En este equipo de 11 personas es muy probable que la comunicación entre el equipo no sea la ideal, ya que por la naturaleza de cada integrante, se tienden a formar sub-equipos entre ellos, donde solo existe comunicación entre estos grupos internamente.

Para solventar este tipo de problemas podemos formar equipos de trabajo según las necesidades del proyecto:

- Equipos de características: Esta organización es en la que se apoya *Scrum* originalmente, ya que los miembros del equipo son de diversas disciplinas y permiten, en un solo núcleo de trabajo, solventar la mayoría de las características que se pueden suscitar en la elaboración de un videojuego.
- Equipos Funcionales: Este tipo de equipos concentra a sus miembros en una sola disciplina, permitiendo que trabajen en una sola funcionalidad muy específica o técnica. Normalmente se usa esta distribución cuando existe alguna característica particular que debe ser explotada por el videojuego, como utilizar cierto tipo de hardware.
- Equipos de infraestructuras compartidas: Este tipo de equipo es muy similar al equipo funcional, con la diferencia de que no trabajan en una característica específica, sino que dan soporte a equipos en varios proyectos simultáneamente. Además, la mayoría de sus integrantes pertenecen a una disciplina en particular pero es posible que haya diversidad de disciplinas para lograr el soporte de esas funcionalidades. La ventaja de esta organización es que un solo grupo especializado puede ayudar en varias características que son similares. La desventaja es que, al no pertenecer a ningún proyecto concreto, tienden a perder el control de las cosas. Es por esto que este tipo de equipos debe tener su propia organización, esto es, su propio *backlog* y su propio dueño de producto, con el fin de que todas las peticiones que se le hagan al equipo estén centralizadas.
- Equipos de Herramientas: En este tipo de equipos, el trabajo no se enfoca en terminar una característica en particular, sino en crear herramientas y soportes para

que los otros equipos puedan culminar exitosamente su trabajo. Aquí, la ventaja fundamental es que los productos que el equipo desarrolla están orientados para sus propios compañeros de trabajo en otros equipos, por lo que obtienen mejor respuesta acerca de las herramientas, lo cual conlleva a crear productos de mayor calidad. Al igual que los grupos de infraestructura compartida, estos tienen su propio *backlog,* priorizado en las herramientas en las que están trabajando.

- Equipos de Apoyo: Un equipo de apoyo es un conjunto de desarrolladores de una misma disciplina cuyo objetivo es dar soporte a los equipos cuando exista alguna necesidad mayor, por ejemplo, un equipo de apoyo de artistas puede crear un conjunto de imágenes para un equipo que esté trabajando en una animación compleja.
- Equipos de Integración: Los equipos de integración son los encargados de unificar los esfuerzos de los demás equipos en un producto final, tomando las características desarrolladas cuando están completas y poder integrarlas y lograr que funcionen en conjunto. Este tipo de equipos es primordial cuando existen varios grupos de trabajo y se requiere de un ente que logre hacer coexistir cada sección del proyecto.

Capítulo 5. Arquitectura de trabajo con XNA

XNA es un framework de desarrollo creado por la empresa Microsoft para la elaboración de videojuegos utilizando la librería gráfica DirectX y el framework .Net, con el fin de facilitar la creación de juegos para las plataformas Windows, la consola Xbox y Windows Phone.

La idea original fue la de crear un conjunto de librerías que permitiesen la creación de videojuegos sin necesidad de re-escribir gran cantidad de código para gestionar el ambiente de trabajo, lo que permite que el desarrollador se pueda enfocar en la lógica y diseño del juego más que en la implementación de la plataforma como tal. Actualmente este framework está en su versión 4.0 y permite trabajar no solo con las consolas Xbox 360, sino también con dispositivos móviles que tengan como sistema operativo Windows Phone 7, lo que genera un gran mercado para los desarrolladores de XNA.

XNA contiene las funciones básicas para la creación de un videojuego, como lo son el manejo de la ventana, el despliegue de imágenes y polígonos, captura de eventos generados desde un teclado o control, y comunicación con otros jugadores a través de una red. Todo lo demás debe ser implementado por los desarrolladores para cada juego en específico.

5.1 Métodos dentro de una aplicación en XNA

Toda aplicación en XNA contiene por lo menos dos archivos: Uno llamado Program.cs, el cual contiene el método Main, que es el punto de partida de toda aplicación en C# e invoca al segundo archivo, usualmente llamado Game1.cs, el cual contendrá toda la lógica del juego. Apartando toda la programación adicional que un juego requiere, un juego básico en XNA tendrá los métodos descritos a continuación:

5.1.1 Initialize

Este método es invocado antes de correr el juego. Su función es crear las estructuras necesarias e inicializar cualquier servicio o cargar clases y librerías.

5.1.2 LoadContent

Este método se invoca luego del método Initialize, y permite cargar todo el contenido grafico de la aplicación. Esta carga de contenidos se hace mediante el *Content Pipeline* (ver Figura 5.1.2), el cual permite cargar una gran cantidad de formatos gráficos y de audio por defecto, e incluir nuestros propios formatos. Para esto debemos crear una clase que importe esos datos y los convierta a un formato entendible por el *Content Pipeline* llamado *Content* DOM (*Document Object Model*). Luego un procesador de contenidos lee estos datos y se los proporciona al compilador de contenidos para generar un archivo XNB, el cual será el utilizable por nuestra aplicación.

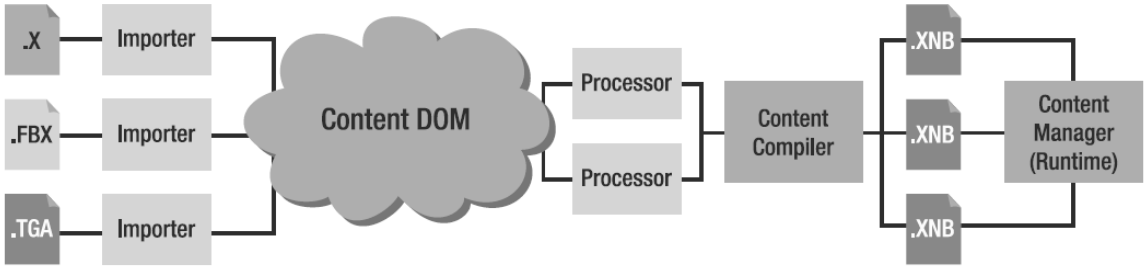


Fig. 5.1.2 Proceso de carga de un recurso en el Content Pipeline [11].

5.1.3 Update

La función *Update* es la encargada de actualizar el estado del juego. Allí se capturarán los eventos de teclado o controles y dependiendo de esto se tomarán las decisiones pertinentes. Como esta función es la que define el comportamiento del juego, debe ser invocada constantemente para percibir nuevos cambios. XNA invoca esta función 60 veces por segundo, sin embargo, este valor puede ser modificado, de tal forma que si es invocado más frecuentemente, se podrán realizar más operaciones por el costo de reducir el rendimiento y la velocidad del juego.

5.1.4 Draw

La función *Draw* es la que permite desplegar en pantalla todos los gráficos de nuestro juego. Al igual que la función *Update*, esta se invoca 60 veces por segundo, asegurando una frecuencia de 60 FPS. Sin embargo, si queremos aprovechar nuestro hardware gráfico podemos aumentar este valor para mostrar más imágenes por segundo.

5.1.5 UnloadContent

Esta función es análoga a la función *LoadContent*, ya que aquí se libera el espacio de los elementos gráficos que se hayan usado en el juego.

5.2 Manejo de cámara en XNA

Al momento de desarrollar juegos más grandes de una sola vista o "pantalla" es necesario buscar un mecanismo que permita desplazarnos sobre el escenario virtual que estamos construyendo. XNA permite cambiar la "posición" de esa pantalla a través del *Viewport*, pero para realizar estos cambios, debemos crear una clase que nos permita simular el movimiento de cámara y aplicarlo a este *Viewport*. Una cámara posee tres movimientos básicos: *Pitch*, *Yaw* y *Roll*; los cuales representan la rotación utilizando los ejes Y, X y Z respectivamente y la cual podemos ver en la figura 7.2.1.

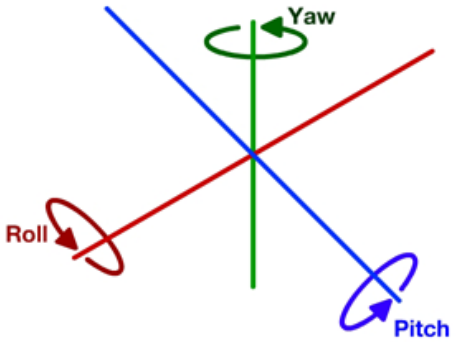


Fig. 5.2.1 Movimientos de cámara.

Además de estas tres variables, necesitamos saber la posición de la cámara en sí y un punto que llamaremos Target, el cual será un punto en el espacio que indica a donde estamos viendo. La diferencia de este punto y los valores Pitch, Yaw y Roll es que estos representan es cuanto se ha rotado desde un punto en particular, y con estos valores podemos construir el punto Target en cada iteración.

Una vez definidos estas variables, se deben actualizar una vez por ciclo para poder generar la nueva posición y vista de la cámara. Creamos un método Update() para poder actualizar dichos valores. Podemos ver su contenido a continuacion:

```
public float Yaw { get; set; }
public float Pitch { get; set; }
public Vector3 Position;
public Vector3 Target { get; private set; }
private Vector3 translation;

public override void Update()
{
//Si lo valores del picth(altura) pasan de los limites(-0.30,
0.70) pongo el borde
if (Pitch < -0.90) Pitch =(float) -0.90;
if (Pitch > 0.90) Pitch =(float) 0.90;

if(Yaw < -MathHelper.ToRadians(360))
    Yaw = MathHelper.ToRadians(360) % Yaw;
if (Yaw > MathHelper.ToRadians(360))
    Yaw = -MathHelper.ToRadians(360) % Yaw;
// Calculamos la matriz de rotacion
Matrix rotation = Matrix.CreateFromYawPitchRoll(Yaw, Pitch, 0);
// Actualizamos la posicion y reiniciamos la translacion
translation = Vector3.Transform(translation, rotation);
Position += translation;
translation = Vector3.Zero;
// Calculamos el nuevo objetivo "Target"
Vector3 forward = Vector3.Transform(Vector3.Forward, rotation);
Target = Position + forward;
// Calculamos el vector "Up"
Vector3 up = Vector3.Transform(Vector3.Up, rotation);
// Calculamos la matriz "View"
View = Matrix.CreateLookAt(Position, Target, up);
}
```

5.3 Trabajando con dispositivos de entrada

Para generar interactividad por parte del jugador, es necesario capturar los comandos introducidos por los dispositivos de entrada, sean teclado y ratón o un control de Xbox360.

Para poder capturar estos elementos se creó una capa de abstracción que permita determinar el estado de un dispositivo en un momento actual, en caso de que este fuese actualizado, su momento inmediato anterior, y dentro de estas variables podemos verificar si alguna de sus secciones (botones, flechas de desplazamiento, palanca) fue utilizado. Por lo tanto, se pueden realizar validaciones como conocer cuando un botón es presionado, cuando un botón se ha mantenido presionado, o la presión que se le hace a una palanca, lo cual evita que el juego pueda tener comportamientos equivocados como avanzar muy rápido en un menú de selección, capturar texto mientras se mantiene presionado un botón del teclado en secciones del juego que se requiera introducir texto, o por el contrario, sea necesario mantener presionado determinado botón para realizar una acción. En la figura 9.1.2.10 de la sección *Descripción y diseño de las clases* podemos ver un diagrama de clases que nos muestra todos los métodos que posee esta clase auxiliar.

5.4 Carga de objetos y texturas

Para representar a los jugadores y a los objetos con los que interactúan en el escenario, es necesario cargar modelos creados previamente con otras herramientas y utilizarlos en el escenario. XNA posee una clase llamada Model para almacenar estos modelos (como el que vemos en la figura 7.3.1), pero como es necesario manejar un conjunto de variables para capturar su posición, rotación, escala y texturas, se creó una clase llamada CModel, donde podremos cargar un modelo en los formatos FBX (Filmbox) o X (DirectX Model) y realizar estas operación a un nivel superior para poder abstraernos al momento de manipular los modelos.

Además de esto, se utilizó la librería XNAnimation[12], la cual permite activar y mostrar las animaciones que deben poseer previamente los modelos dentro del escenario, con lo que se pueden representar diferentes situaciones con el mismo modelo según las circunstancias.

Estas animaciones deben estar delimitadas en un archivo XML, donde se especificará en que cuadro (frame) empieza y en que cuadro termina la animación en particular, por lo que en un solo modelo podemos tener almacenadas varias animaciones juntas, y con esto ahorrar espacio de memoria.

Luego de haber cargado estos modelos procedemos a mostrarlos en pantalla. Para hacer esto, se calculan previamente las matrices que corresponden a la escala, rotación y traslación del modelo, y luego se multiplican de la forma (Escala*Rotación*Traslación) para poder calcular la matriz de transformaciones del modelo. Esta matriz será aplicada en cada sección del modelo (conocido como mesh) para transformarlo acorde a su ubicación actual en el escenario.

```
foreach (ModelMesh mesh in skinnedModel.Model.Meshes)
{
   foreach (SkinnedEffect effect in mesh.Effects)
   {

effect.SetBoneTransforms(animationController.SkinnedBoneTransforms);
      //La matriz de transformaciones se calcula para cada sección
del modelo
      effect.World = this.scaleMatrix * this.rotationMatrix *
this.positionMatrix;
      effect.View = v;
      effect.Projection = p;
   }
   mesh.Draw();
}
```

5.5 Trabajo en red

XNA posee un conjunto de herramientas que permiten que un juego pueda ser compartido en varios equipos en simultaneo, gracias al servicio *Live for Windows*. Al momento de que un jugador crea una sesión, crea una instancia de la clase *NetworkSession*, y es visible por todos los jugadores que busquen en la misma red local utilizando el método NetworkSession.Find, limitando la entrada de estos jugadores a un máximo de 32 personas, según se haya estipulado al momento de crear esta sesión.

Luego de esto es posible enviar paquetes de información utilizando la misma instancia de NetworkSession e invocando al método *SendData(PacketWriter, SendDataOptions, NetworkGamer)*, donde PacketWritter es un buffer que almacena toda la información que será enviada, SendDataoptions es el tipo de paquete que será enviado(esto esta explicado en mayor detalle en el Capítulo 7, sección 4 Trabajo en red), y por ultimo NetworkGamer es el jugador al que se le envían estos paquetes, por lo que es necesario indicar cada uno de los jugadores que recibirán información.

XNA posee un protocolo de transferencia de datos donde existen cinco tipos de prioridades para los mensajes:

- **None or Chat**: Estos mensajes son aquellos que son enviados entre jugadores, así que su contenido no afecta ninguna funcionalidad del juego, por lo tanto, estos no poseen ninguna prioridad de envío.

- **Unreliable Out of Order**: Este tipo de mensajes permiten mandar un conjunto de datos sin ninguna garantía, por lo que el contenido de estos paquetes podría perderse bajo algún problema de conexión.

- **Unreliable In Order**: Este tipo de mensajes permiten mandar un conjunto de datos con la condición de que lleguen en el orden apropiado, la desventaja principal de estos es para asegurar el correcto orden es posible que sean demorados otros datos para poder enviar toda la información.

- **Reliable Out of Order**: Este tipo de mensajes están diseñados para asegurarse que todo su contenido sea entregado, para lograr este objetivo es posible que se envíen varias veces los paquetes en cado de que no lleguen y tienen la desventaja que no está asegurado el orden correcto de la información, por lo que al llegar se realiza una verificación para revisar el correcto orden de los datos.

- **Reliable In Order**: El ultimo tipo de envío de datos es una mezcla de los dos anteriores, por lo que consigue enviar toda la información del paquete y asegura su orden correcto, con lo que tendremos la información ya verificada, donde su problema es que para lograr esto se debe dedicar el canal de comunicación entre los jugadores, y ralentiza el envío de datos con respecto a los otros métodos.

5.6 Almacenamiento Local y uso de archivos externos

Para poder editar los escenarios del juego, y que perduren a través del tiempo, es necesario utilizar una estructura en el disco local que permita almacenar todos los elementos que intervienen en el.

Se diseñó una estructura de matriz, que almacena en cada posición un elemento único que representara obstáculos, zonas de salida inicial de cada equipo, o las banderas dentro del escenario. Para crear estas estructuras, se diseñó un editor de mapas dentro del juego para que cualquier jugador pueda definir sus propios mapas. Una ventaja de realizar el manejo de los mapas de esta forma, es que pueden ser editados en cualquier programa externo (Notepad, por ejemplo), con lo que se eliminaría los límites visuales que impone el editor de niveles (dentro del juego, es posible editar un mapa con un tamaño máximo de 100 unidades de ancho por 70 de alto dadas las limitaciones de tamaño de la pantalla).

Luego de esto, el escenario finalizado se envía como cadenas de texto a los demás jugadores, se descompone dicha información y se construye el mismo escenario en su máquina local. Una vez recibido el mapa, los jugadores remotos pueden guardar este mapa que les fue enviado, para en un futuro crear sus propias partidas sobre este mismo mapa.

En el momento en que el cliente recibe el mapa, se actualizan los BoundingBox del mapa almacenados en el cliente, para coincidir con el nuevo mapa. También se actualiza la posición inicial de cada equipo, así como la posición de la bandera.

5.7 Manejo de colisiones

El manejo de colisiones es el mecanismo que permite realizar acciones en caso de que dos objetos colisionen, y así poder tomar decisiones como por ejemplo evitar que dos objetos ocupen el mismo espacio físico dentro del escenario, o eliminar a un jugador alcanzado por un proyectil.

XNA ofrece clases específicas que nos pueden ayudar para estas verificaciones. Las más resaltantes son las clases *BoundingBox*, *BoundingSphere* y *Ray*, las cuales tienen métodos *Intersect()* los cuales determinan si se produjo una colisión o no con otro objeto de estos tipos. Para instanciar un objeto de estos tipos, nosotros debemos calcular las coordenadas máximas y mínimas del objeto, por lo que sus ventajas se reducen a sus métodos de intersección. Además la clase BoundingBox tiene la desventaja de que la caja envolvente generada siempre es orientada a los ejes cartesianos canónicos, por lo que al utilizar esta clase no estaremos utilizando la caja envolvente mínima, sino la mínima orientada a los ejes cartesianos canónicos.

5.8 Manejo de audio

Para poder utilizar efectos de sonido y pistas de audio, XNA viene incorporado con clases que permiten reproducir archivos en formatos específicos (mp3, wma, y wav) y ejecutarlos al momento. Para efectos de corta duración tenemos la clase *SoundEffect*, la cual carga un archivo de audio y cuando termina de tenerlo en memoria lo reproduce. Para evitar esta carga, sobre todo en pistas de audio extendido, como la música de fondo de cualquier juego, existe la clase *Song*, la cual permite reproducir en "streaming" cualquier pista, al costo de que ocupa más espacio en memoria, ya que considera este archivo parte de una "biblioteca de multimedia", por lo cual también carga toda la información adicional que contenga el audio, así como lo es el artista, el nombre del álbum que pertenece, y permite realizar acciones similares al reproductor de Windows Media.

Capítulo 6. Proceso de desarrollo del T.E.G.

En este capítulo se explicaran detalladamente todas las actividades realizadas para realizar el producto siguiendo la metodología Scrum descrita previamente.

6.1 Back-log de Productos

Aquí se mostrarán todos los elementos que se desarrollaran a lo largo del desarrollo del videojuego. Estas características se derivan de las historias de usuario. Los elementos son:

- Crear el Documento de Diseño de Juego
- Crear personajes
- Crear menús de navegación principal
- Cargar mapas
- Crear escenarios
- Crear sistema de colisiones
- Manejar diversos dispositivos de entrada
- Manejar cada modo de juego
- Saltar
- Disparar
- Conectar en red
- Enviar datos en red
- Guardar datos del jugador en el sistema
- Personalizar el jugador

6.2 Plan de Iteraciones

Iteración 1

Se procedió con la elaboración del Documento de Diseño de Juego (ver Apéndice A), el cual contiene información acerca de la descripción del juego, características clave, modos de juego, mecánicas del juego, las posibles acciones de un jugador, los casos de uso generados en el juego, el diagrama de actividades donde indica cómo interactúan todas las opciones del menú principal (visto en la Figura 6.2.1), una imagen inicial de cómo debe ser la interfaz del usuario y un diagrama con los controles tanto para Xbox como para teclado.

Se creó un prototipo donde se expone la navegación del juego en un mapa sencillo creado en *3D Studio Max* (ver Figura 6.2.4).

Sprint Back-Log

- Crear el Documento de Diseño del Juego (ver Apéndice A).

  Historia de Usuario: "Como desarrollador, deseo tener un documento que contenga todas las características a desarrollar del videojuego."

  Condiciones de satisfacción:

  - Indicar todas las características del juego.
  - Crear todos los diagramas correspondientes a los diversos artefactos del documento.

- Crear menús de navegación principal.

  Historia de Usuario: "Como usuario, deseo poder navegar entre las diversas opciones del juego."

  Condiciones de satisfacción:

  - El usuario debe poder navegar entre los diversos menús que ofrece el juego, como los presentados en las Figuras 6.2.1, 6.2.2 y 6.2.3.
  - El usuario debe conocer en que sección del sistema de menús se encuentra.
  - El usuario tiene un mecanismo para volver a los menús anteriores.

Fig. 6.2.1 Menú Principal.

Fig. 6.2.2 Menú Nuevo Juego.

Fig. 6.2.3 Menú Crear Partida.

- Cargar mapas.

  Historia de Usuario: "Como usuario, debe existir un espacio físico en el cual desplazarse."

  Condiciones de satisfacción:

  - Visualizar un escenario (como el mostrado en la Figura 6.2.4) con todos sus obstáculos y elementos.

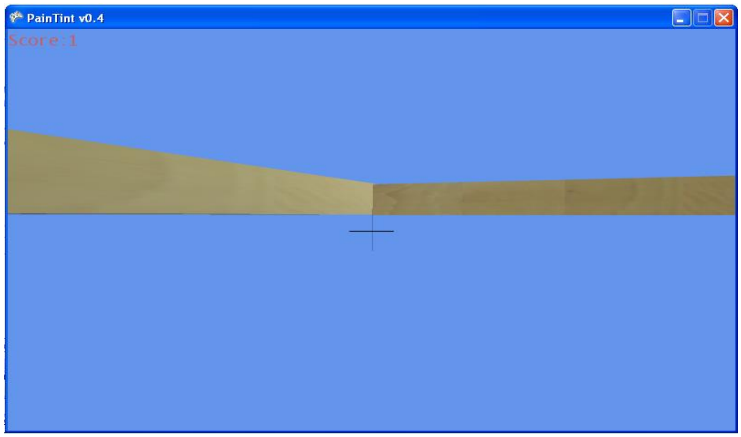


Fig. 6.2.4 Primer Mapa creado.

Iteración 2

Se mejoró el prototipo de navegación con una versión más completa. Se le añadió un conjunto de menús para poder seleccionar entre jugar como cliente o servidor en una partida.

Se desarrolló un mapa más completo para las pruebas de navegación utilizando la herramienta *Google Sketch Up*. Además se agregaron las funcionalidades para disparar y se agregaron sonidos en el menú y al momento de presionar la tecla de disparo.

Los diagramas de clases fueron actualizados y algunos de estos diagramas fueron eliminados por representar acciones que se manejan directamente dentro de las mecánicas del juego, como lo son las diversas interacciones con los proyectiles y los demás objetos de la escena.

Se le agregó al prototipo la posibilidad de conectarse a una partida en red. Para esto, debe existir una partida ya creada donde algún jugador escogió ser *Host* (Servidor). En caso de que un jugador haya escogido ser servidor, no podrá conectarse a otra partida hasta culminar la sesión actual.

El prototipo valida todo tipo de colisiones de la cámara con el escenario, por lo que se puede proceder a crear las interacciones de los proyectiles con el escenario y con otros jugadores.

Sprint Back-Log

- Crear menús de navegación principal.

  Historia de Usuario: "Como usuario, deseo que pueda navegar por las diversas opciones del juego."

  Condiciones de satisfacción:

  - El usuario debe poder navegar entre los diversos menús que ofrece el juego.
  - El usuario debe conocer en que sección del sistema de menús se encuentra.
  - El usuario tiene un mecanismo para devolverse a través de los menús.

- Reproducir sonidos.

  Historia de Usuario: "Como usuario, deseo que el juego tenga elementos de audio."

  Condiciones de satisfacción:

  - Reproducir música en los menús.
  - Reproducir sonidos en las partidas (como el disparo de los marcadores, las colisiones de las esferas, las pisadas de los jugadores, etc.).
  - Reproducir comandos de voz mediante un menú.

- Cargar mapas.

Historia de Usuario: "Como usuario, deseo que el juego tenga diversos escenarios para jugar."

Condiciones de satisfacción:

- El usuario puede visualizar un escenario con todos sus obstáculos y elementos.

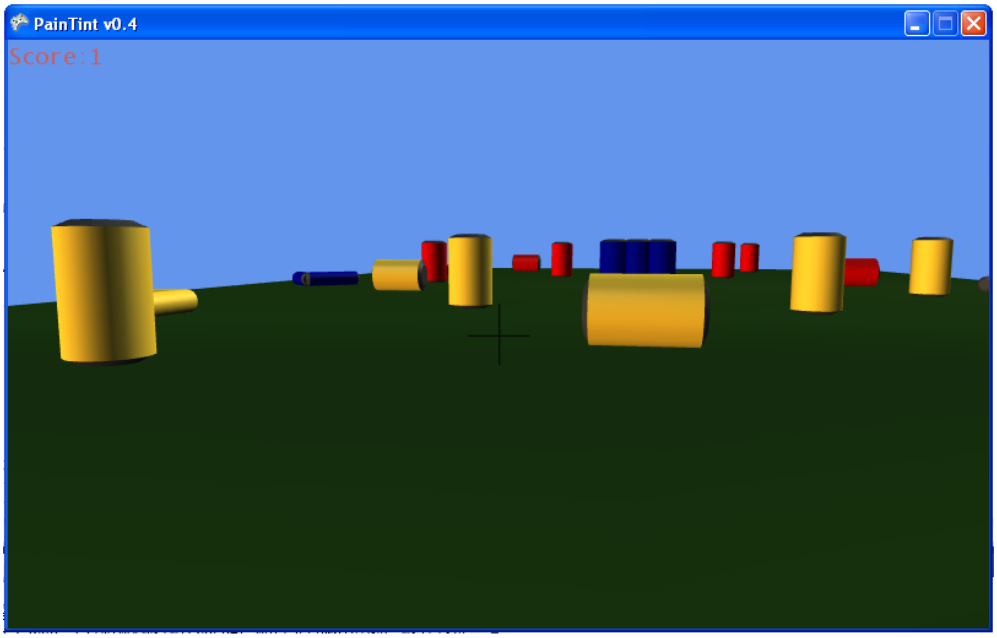


Fig. 6.2.5 Segundo Mapa Creado.

- Crear sistema de colisiones.

  Historia de Usuario: "Como desarrollador, deseo que los jugadores no puedan atravesar objetos sólidos."

  Condiciones de satisfacción:

  - El usuario no podrá atravesar objetos que se encuentren en el mapa (como los mostrados en la Figura 6.2.5).
  - El jugador que sea alcanzado por una esfera de pintura del equipo contrario deberá ser eliminado del escenario.

- Saltar.

  Historia de Usuario: "Como usuario, deseo que al momento de participar en un encuentro pueda saltar por el escenario."

  Condiciones de satisfacción:

  - El usuario podrá saltar obstáculos que se encuentren en el mapa.

- Conectar en red.

Historia de Usuario: "Como usuario, deseo poder participar en una partida con varios jugadores a través de una conexión de red."

Condiciones de satisfacción:

- El usuario podrá crear una partida siendo el anfitrión de dicho juego.
- El usuario podrá ingresar a una partida previamente creada por otro jugador.

Iteración 3

Se mejoró el prototipo de navegación para generar movimientos alternos cuando se generan colisiones con objetos, con el fin de rodearlos sin girar específicamente.

La capacidad Multijugador fue mejorada, ahora el límite máximo de jugadores es de 20 jugadores, a través de la plataforma *Windows Live*. Antes de empezar cada partida los jugadores se encuentran en una sala de espera o *lobby*, donde podrán escoger el equipo al cual pertenecen antes de iniciar. Cada jugador tendrá un avatar (elemento que representa al jugador actual) dentro del juego, el cual en estos momentos es un modelo usado solo para pruebas.

Se agregó la dinámica física para disparar esferas de pintura. Esto se hace desde la cámara del jugador y es transmitido a los demás jugadores para conocer si fueron impactados. Para ello, se agregó la detección de colisiones entre una esfera de pintura y un jugador o un objeto del mapa, permitiendo saber si el jugador debe salir de la partida o continuar.

Sprint Back-log

- Conectar en red.

  Historia de Usuario: "Como usuario, deseo poder participar en una partida con varios jugadores a través de una conexión de red."

  Condiciones de satisfacción:

  - El usuario podrá crear una partida siendo el anfitrión de dicho juego.
  - El usuario podrá ingresar a una partida previamente creada por otro jugador.
  - El jugador podrá cambiarse de equipo antes de iniciar la partida.
  - Todos los jugadores tendrán información acerca de los demás participantes.

- Mejorar el sistema de colisiones actual.

  Historia de Usuario: "Como desarrollador, deseo que los jugadores no puedan atravesar objetos sólidos."

  Condiciones de satisfacción:

  - El usuario no podrá atravesar objetos que se encuentren en el mapa.

- El jugador que sea alcanzado por una esfera de pintura del equipo contrario deberá ser eliminado del escenario.
- El jugador al chocar con una pared será capaz de avanzar bordeando la pared donde chocó.
- Cuando una esfera alcanza un edificio es eliminada del juego.

- Crear un modelo para cada jugador

  Historia de Usuario: "Como jugador, deseo que mi personaje tenga un avatar distintivo dentro del juego"

  Condiciones de satisfacción:

  - El usuario tendrá un avatar que lo representará dentro del mapa (ver Figura 6.2.6).
  - El jugador podrá ver a los otros jugadores a través de sus avatares.
  - Los avatares de los otros jugadores tendrán animaciones que representaran sus acciones, como lo es saltar, atacar, agacharse y caminar.

Fig. 6.2.6 Modelo 3D para representar al jugador.

Iteración 4

Se incorporó la capacidad de escoger entre diversos mapas al momento de crear una partida, por lo que se generó un segundo mapa para probar esta función en detalle.

Se creó una interfaz de usuario para la pantalla de *lobby* (como el presentado en la Figura 6.2.7), donde los jugadores podrán escoger en que equipo participaran.

Sprint Back-log

- Creación de un segundo escenario.

  Historia de Usuario: "Como desarrollador, deseo que haya un escenario con espacios cerrados."

  Condiciones de satisfacción:

  - El jugador podrá navegar dentro de un mapa cerrado, el cual será un edificio que tendrá una gran cantidad de pasillos.
  - El jugador podrá reclamar la victoria si su equipo logra eliminar a todos los miembros del equipo contrario.

- Crear Interfaz de usuario para el *lobby*.

  Historia de Usuario: "Como usuario, deseo poder crear y acceder a una partida de forma intuitiva."

  Condiciones de satisfacción:

  - El usuario podrá cambiarse de equipo en cualquier momento antes de empezar la partida.
  - El jugador podrá ver que jugadores están en cierto equipo al momento de entrar a la partida, tal como es mostrado en la Figura 6.2.7.

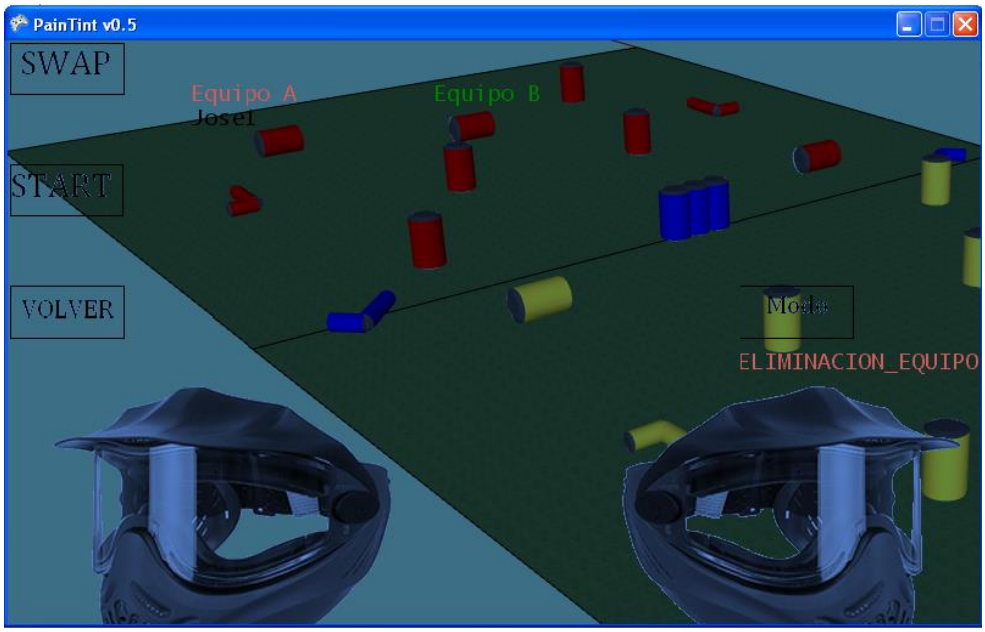


Fig. 6.2.7 Menú Lobby.

Iteración 5

Se creó un sistema de reloj para que al iniciar cada encuentro se muestre una cuenta regresiva, con el fin de que los jugadores se preparen antes de iniciar el enfrentamiento. Luego de que un jugador es alcanzado con una esfera, el juego lo remueve de la partida actual hasta que no queden jugadores según el tipo de juego que se esté realizando. Para conocer cuál es la puntuación actual de los jugadores, se desarrolló un sistema que permite ver la lista de los equipos en tiempo real con solo presionar la tecla de tabulación.

Sprint Back-log

- Creación de un sistema de relojes.

  Historia de Usuario: "Como usuario, deseo conocer cuánto tiempo resta para finalizar una partida".

  Condiciones de satisfacción:

  - Al momento de iniciar un encuentro, debe aparecer un reloj indicando que la partida empieza en 5 segundos (tal como indica la Figura 6.2.8).
  - En cada encuentro, a menos que todos los participantes hayan sido vencidos, aparecerá un reloj mostrando el tiempo límite de la partida, el cual será de 2 minutos y 30 segundos.

- Mejorar interfaz de usuario para el *Lobby*.

  Historia de Usuario: "Como usuario, deseo poder crear y acceder a una partida de forma intuitiva."

  Condiciones de satisfacción:

  - El jugador podrá ver qué jugadores están en cada equipo al momento de entrar a la partida.
  - El jugador que creó la partida podrá cambiar el modo de juego que se usará para la partida actual.
  - El jugador que creó la partida podrá cambiar el nombre de la partida actual.

- Añadir diversos modos de juego.

  Historia de Usuario: "Como usuario, deseo poder crear una partida y seleccionar diversos modos de juego."

  Condiciones de satisfacción:

  - El jugador que creó la partida podrá elegir entre los modos Eliminación, Eliminación por equipos y Entrenamiento.

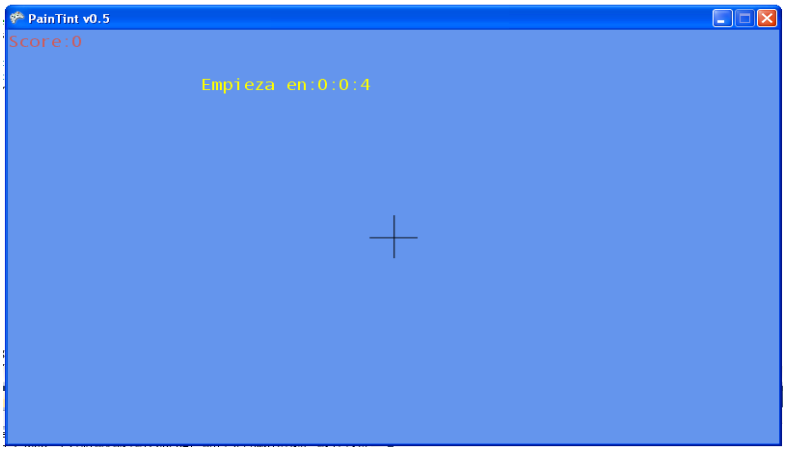


Fig. 6.2.8 Reloj inicial.

Iteración 6

En esta iteración se creó un escenario externo (*Skybox*) para crear un ambiente sin depender de las geometrías del escenario, con el fin de agilizar el renderizado del mismo. Además de eso, se hizo una revisión del sistema de disparos, con el fin de poder validar cuando inutilizar una esfera cuando encuentra un objeto dentro de su interpolación de movimiento. También se agregó gravedad al movimiento de las esferas.

Sprint Back-log

- Creación de un *Skybox*.

  Historia de Usuario: "Como desarrollador, deseo ver en el entorno un escenario predefinido".

  Condiciones de satisfacción:

  - Al momento de iniciar un encuentro, debe aparecer una imagen rodeando el escenario (tal como indica la Figura 6.2.9).

- Mejora en el sistema de disparos.

  Historia de Usuario: "Como sistema, se desea saber cuándo un disparo impacta con un objeto antes de un jugador"

  Condiciones de satisfacción:

  - Al disparar a otro jugador, se podrá identificar cuando la esfera interseca a un objeto que está por delante del jugador objetivo. Cuando la esfera

Fig. 6.2.9 Escenario con *Skybox*.

Iteración 7

En esta iteración se agregaron detalles visuales del juego y se implementó un sistema que permite recoger objetos que se encuentran a lo largo del escenario, con el objetivo de mejorar alguna variable del juego, como lo puede ser la salud del personaje o su capacidad de disparo, además de permitir la implementación del modo de juego "Captura la Bandera". Además de eso, se implementó un sistema de mensajes para poder comunicar a los jugadores.

Sprint Back-log

- Creación de un sistema de objetos que se puedan capturar.

  Historia de Usuario: "Como usuario, deseo poder navegar por el escenario y tomar del piso diversos tipos de objetos".

  Condiciones de satisfacción:

  - Mientras el jugador este navegando por el escenario, puede colisionar con objetos para tomarlos y llevarlos consigo, como por ejemplo una bandera.

- Creación de un sistema de mensajería de texto.

  Historia de Usuario: "Como usuario, se desea poder comunicarse con los demás jugadores a través de un sistema de mensajes de texto"

Condiciones de satisfacción:

- Mientras se esté en la partida, el usuario podrá chatear con los otros jugadores a través de una pequeña consola que aparecerá en la pantalla (como en la Figura 6.2.10). Mientras se está escribiendo se pueden ver los últimos 5 mensajes hechos al usuario

Fig. 6.2.10 Mensajes de texto en pantalla.

Iteración 8

En esta iteración se procedió a descentralizar todo el código del juego en pantallas, clases que representan cada estado dentro del juego, y que siguiendo un patrón de polimorfismo, permiten simplificar la lógica del motor de juego a un algoritmo sencillo, invocando al objeto pantalla actual, y permitiendo que al momento de desarrollar nuevas funcionalidades y eventos al juego, puedan hacerse a través de este patrón.

También se utilizaron las bondades de las clases virtuales para agregar soporte para varios idiomas, creando una clase virtual *Language*, la cual contiene la definición de todas las variables necesarias y de la cual sus descendientes (las clases *English* y *Spanish*) definen dichos atributos.

Además de esto, se mejoró el cálculo del algoritmo para el movimiento y colisión de las esferas, con lo que se obtiene un comportamiento más realista al momento de lanzar las esferas por espacios estrechos.

Sprint Back-log

- Dividir el código por pantallas.

  Historia de Usuario: "Como desarrollador, necesito una forma ordenada de desarrollar en un proyecto que está tomando mayores dimensiones".

  Condiciones de satisfacción:

  - Poder realizar cambios en una pantalla fácilmente sin afectar el resto del videojuego.

- Agregar soporte para varios idiomas.

  Historia de Usuario: "Como usuario, quiero que el juego tenga soporte para varios idiomas".

  Condiciones de satisfacción:

  - Que el juego tenga por lo menos dos posibles idiomas.

Fig. 6.2.11 Incorporación de clases virtuales.

Iteración 9

En esta iteración se creó un editor de mapas integrado al juego, con lo que cualquier jugador puede crear su propio mapa antes de empezar su partida y compartirla con los demás jugadores.

Sprint Back-log

- Creación de un editor de mapas.

  Historia de Usuario: "Como usuario, deseo poder crear los escenarios".

  Condiciones de satisfacción:

  - El jugador puede modificar el tamaño del escenario, además de agregarle diversos obstáculos.
  - El jugador puede modificar la ubicación de elementos claves, como lo son el punto de origen de los equipos, o la posición de la bandera en el modo "Captura La Bandera", tal como podemos ver en la figura 6.2.12.
  - Una vez creado el mapa, el jugador puede probar inmediatamente su nuevo mapa antes de guardarlo en el disco.

- Carga dinámica de mapas.

  Historia de Usuario: "Como usuario, deseo poder incluir otros mapas que reciba por otros medios a mis juegos".

  Condiciones de satisfacción:

  - El jugador puede agregar otros mapas creados por otras personas en su propia versión.

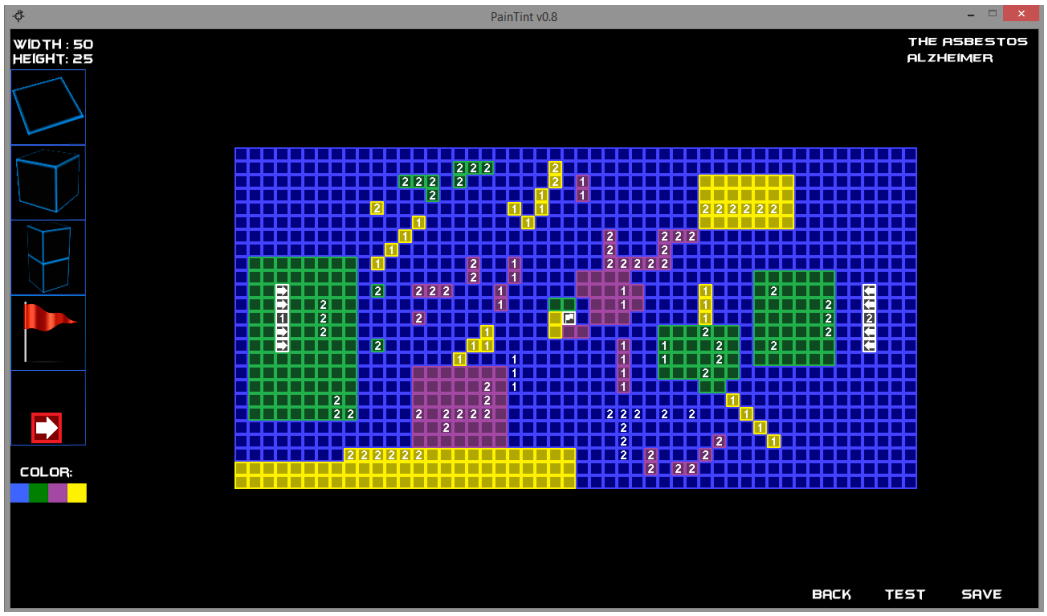


Fig. 6.2.12 Editor de mapas

Iteración 10

En esta iteración se realizó un cambio de imagen de todo el juego con el fin de adaptarlo al cambio de escenario generado por la creación de mapas, además de hacerlo mucho más llamativo al espectador, cambiando la forma de presentar los objetos dentro de un mapa. Además de esto, se adaptaron los menús del juego a cualquier tamaño de pantalla, con lo que jugadores con monitores de diferentes tamaños y resoluciones tendrán una experiencia similar. También se cambió el diseño del menú principal.

Al modelar los objetos con Google Sketchup, al momento de cargarlos en el proyecto los mismos tenían problemas con las texturas, mostrando únicamente las texturas en algunas caras del obstáculo. Por esta razón, luego de intensas pruebas con varios programas de modelado 3D, se decidió utilizar el Blender en su versión 2.68.

Sprint Back-log

- Editor de mapas.

  Historia de Usuario: "Como desarrollador, deseo adaptar los obstáculos que agrego con el editor de mapas a las colisiones que se generan dentro del juego, además de simplificar su representación en el espacio".

  Condiciones de satisfacción:

  - Los objetos creados por el editor de mapas serán de forma rectangular, con el fin de adaptarse a la estructura BoundingBox de XNA, tal como podemos apreciarlos en la figura 6.2.12.
  - Las texturas utilizadas por los obstáculos del escenario estarán previamente creadas y podrán ser reutilizadas libremente.
  - Se planifico un rediseño en la presentación del juego, como vemos en la figura 6.2.13, con el fin de hacerlo más comprensible para el jugador.

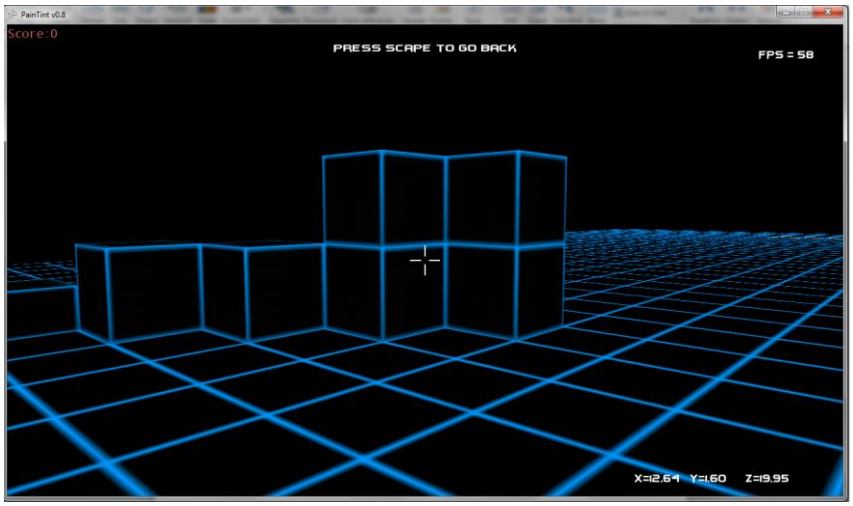


Fig. 6.2.13 Nuevo Diseño de los mapas.

- Menús responsivos.

  Historia de Usuario: "Como usuario, deseo ubicar todas las secciones de los menús, sin importar el tamaño de mi pantalla".

  Condiciones de satisfacción:

  - Los menús deben adaptarse a cualquier resolución de pantalla, con el fin de que todos los usuarios tengan experiencias similares sin importar el tamaño de la pantalla.

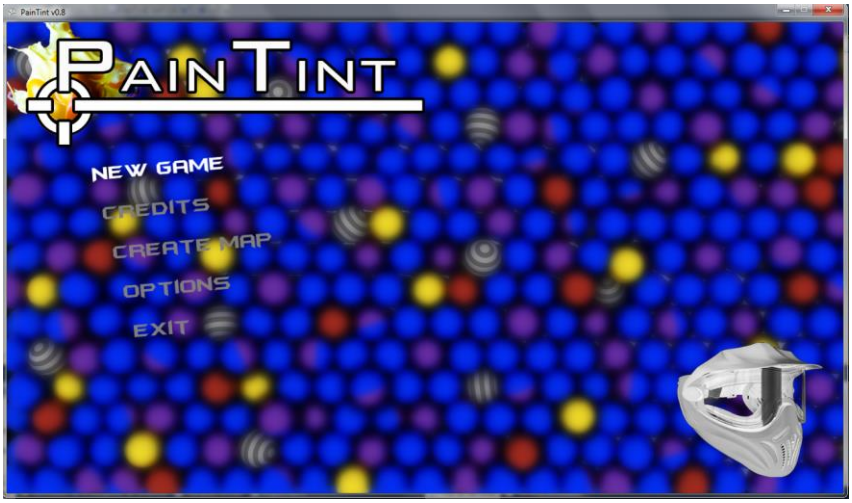


Fig. 6.2.14 Menú principal

Iteración 11

En esta iteración se agregó el envío de mapas personalizados desde el servidor de una partida a los clientes. Se agregó también la posibilidad de guardar el mapa recibido por los clientes para su uso a futuro en otras partidas por parte de los clientes. Además se realizaron pruebas y

corrección de errores en el funcionamiento general del videojuego, desde detalles de jugabilidad hasta problemas de sincronización entre el host y los clientes.

Sprint Back-log

- Envío de mapas por red.

  Historia de Usuario: "Como desarrollador, es necesario enviar los mapas creados localmente a los clientes que se conectan a la partida, de modo que todos jueguen en el mismo mapa. Como funcionalidad adicional, se debe agregar la posibilidad de guardar el mapa recibido".

  Condiciones de satisfacción:

  - Enviar y recibir el mapa actual por red.

- Solución de errores.

  Historia de Usuario: "Como desarrollador, es vital que todos los jugadores estén sincronizados en tiempo, y que sea posible jugar sin que se presente ningún tipo de errores".

  Condiciones de satisfacción:

  - Juego totalmente funcional.

Iteración 12

Se cambió el diseño de todos los menús con la ayuda de la Arquitecto Viviana Da Silva, quien realizó el diseño de todos los menús. Además de esto, se reorganizaron los elementos del HUD.

Sprint Back-log

- Rediseño de menús.

  Historia de Usuario: "Como Usuario, quiero que los menús sean más atractivos visualmente".

  Condiciones de satisfacción:

  - Menús atractivos.

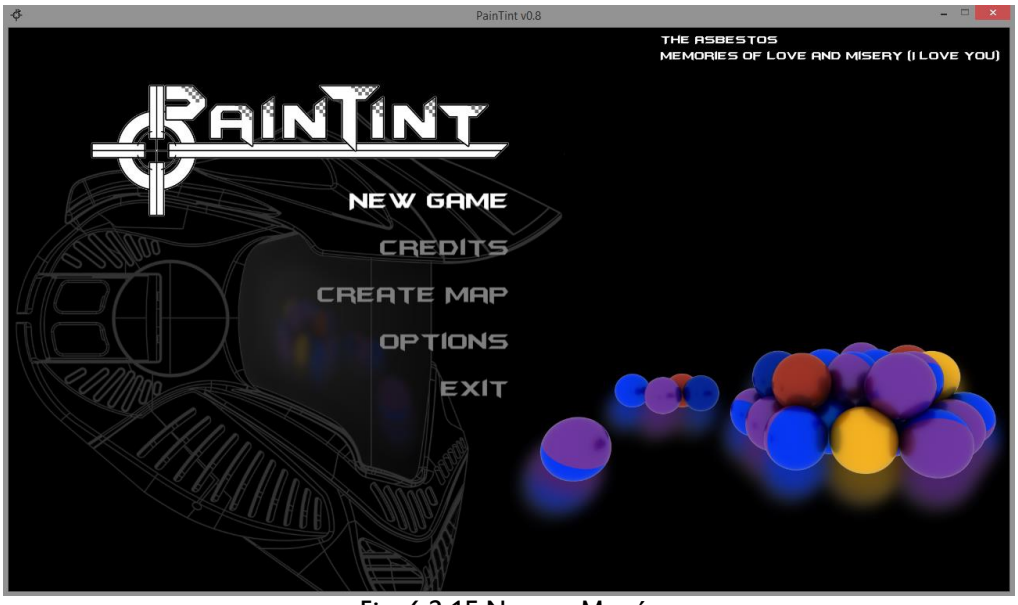


Fig. 6.2.15 Nuevos Menús

- Rediseño del HUD y Skybox.

  Historia de Usuario: "Como Usuario, quiero que el HUD muestre la información necesaria para conocer el estado del juego, además de hacerlo más atractivo. El Skybox debe ser más atractivo también".

  Condiciones de satisfacción:

  - Poder visualizar la información necesaria en el HUD y obtener un Skybox más atractivo.

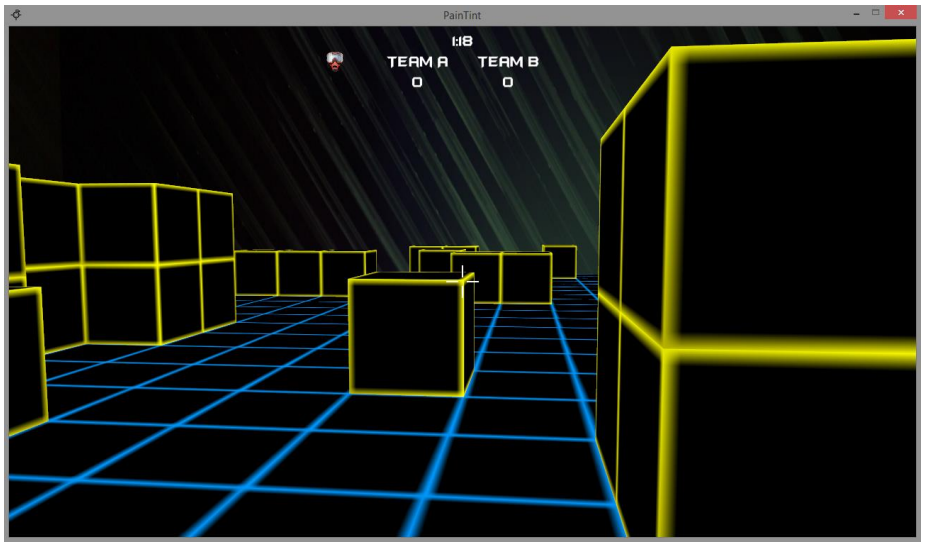


Fig. 6.2.16 Información del HUD y nuevo Skybox

Capítulo 7. Diseño e Implementación

En este capítulo se explicarán detalladamente todos los pasos que se siguieron para la realización de este trabajo, además de todos los artefactos que se generaron a través de la metodología Scrum antes descrita.

7.1 Diseño de la aplicación

Para poder realizar la aplicación, se crearon varias estructuras de datos y clases que ayudaron a complementar las funciones proporcionadas por el framework XNA, además de ayudar a definir toda la lógica del juego en cuestión.

7.1.1 Diagrama de clases

A continuación se muestra el diagrama de clases (ver Figura 7.1.1.1) que se utilizó como base para representar la lógica del juego. Además de esto se explicará la funcionalidad de cada una de estas clases.

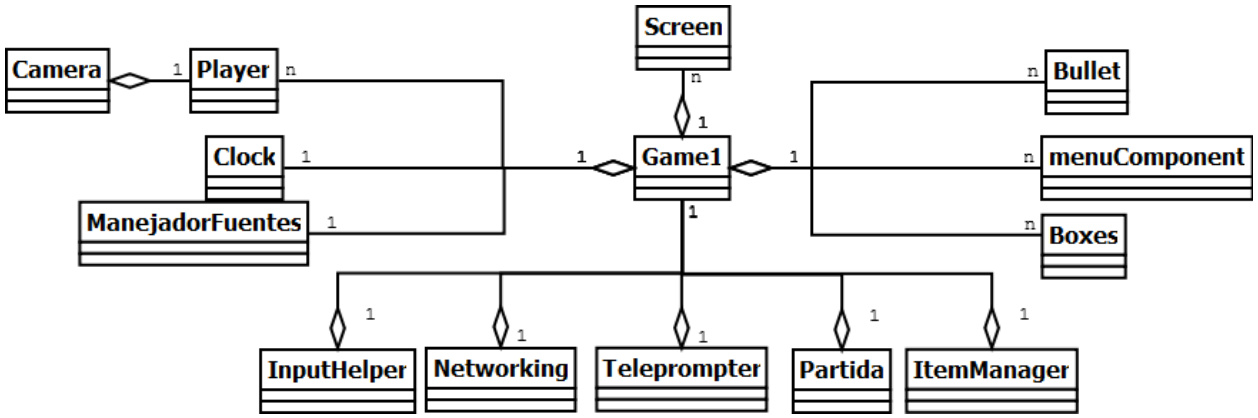


Fig. 7.1.1.1 Diagrama de clases del juego "PainTint".

- Clase Game1: Esta es la clase principal de la aplicación. Esta se encarga de la inicialización y creación de objetos, métodos y del desarrollo de la lógica del juego. En esta clase se maneja un enumerado "*tipoEstados*", el cual permite identificar en que punto de la ejecución del juego se encuentra el jugador y capturar correctamente los eventos correspondientes a esa sección.
- Clase Screen: Esta es una clase virtual de donde se generarán cada sección del juego. Una vez que una clase herede de esta clase virtual, puede ser llamada por el programa principal, por lo que resulta sencillo la continuación de una determinada sección a otra, sin la necesidad de tener todas las secciones una gran clase y que las transiciones de una sección a otra.
- Clase Player: Se encarga de representar a un jugador en la sesión de juego. Al iniciar la aplicación se crea una instancia para representar al jugador local y luego de conectarse en

red, se solicitan las instancias de los demás jugadores conectados para almacenarlas en una lista, la cual permitirá conocer todas sus características.

- Clase Networking: Esta clase permite la interconexión entre esta aplicación y otras instancias de ella que estén ejecutándose a través de una red local. Para realizar una conexión de red se utilizan los servicios de Windows Live para el envío y recepción de mensajes y de la identificación de los jugadores a través de la red local.
- Clase InputHelper: Se encarga de capturar los eventos generados por teclado y ratón y traducirlos en eventos dentro de la lógica del juego.
- Clase Partida: Esta clase administra la lista de jugadores conectados, que tipo de partida se está realizando (Eliminación, Eliminación con equipos, Capturar la bandera y Entrenamiento) y valida que, según el tipo de partida, se cumpla el objetivo de culminación respectivo.
- Clase MenuComponent: Esta clase genera un menú al cual se le pueden agregar varios elementos para seleccionar. Este menú se puede navegar a través del teclado utilizando las flechas de movimiento "arriba" y "abajo".
- Clase Clock: Esta clase permite conocer el tiempo restante luego de determinado evento.
- Clase Camera: Esta clase es la encargada de visualizar todos los elementos dentro de una partida
- Clase Bullet: En esta clase se representa toda la información de los proyectiles de pintura que se utilizan en el juego.
- Clase TelePrompter: En esta clase se manejaran los métodos para poder mostrar por pantalla mensajes de texto que pueden ser enviados por otros participantes. Los mensajes de texto se borran con el tiempo y pueden revisarse los últimos 5 mensajes recibidos.
- Clase Boxes: La clase Boxes representa un Bounding Box en el espacio, además de diversos métodos para interactuar y detectar colisiones con el mismo.
- Clase ManejadorFuentes: Esta clase permite mostrar por pantalla texto con una fuente de texto en específico. La idea fundamental de la clase es que se pueda utilizar cualquier tipo de fuente dentro de la aplicación, previamente instalándola dentro de la aplicación.
- Clase ItemManager: Esta clase permite incluir objetos autónomos a los escenarios del juego. Además de esto se pueden verificar interacciones como colisiones entre los objetos.

7.1.2 Descripción y diseño de las clases

A continuación se explican con detalle las clases nombradas anteriormente mostrando sus atributos y métodos correspondientes.

Clase Game1: Esta es la clase principal de la aplicación. Esta se encarga de la inicialización y creación de objetos, métodos y del desarrollo de la lógica del juego. En esta clase se maneja un enumerado llamado "tipoEstados", el cual permite identificar en que punto de la ejecución del juego se encuentra el jugador y capturar correctamente los eventos correspondientes a esa sección. Se puede apreciar en detalle en la Figura 7.1.2.1.

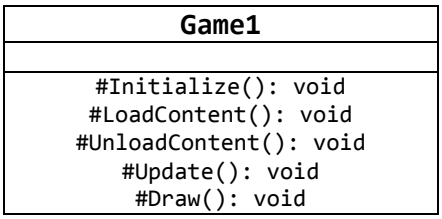


Fig. 7.1.2.1 Diagrama de la clase Game1.

Clase Screen: Se encarga de representar a un jugador en la sesión de juego. Al iniciar la aplicación se crea una instancia para representar al jugador local y luego de conectarse en red, se solicitan las instancias de los demás jugadores conectados para almacenarlas en una lista, la cual permitirá conocer todas sus características, las cuales se pueden ver en la Figura 7.1.2.2.

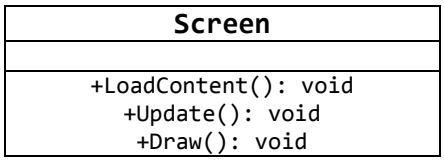


Fig. 7.1.2.2 Diagrama de la clase Screen.

Clase Player: Se encarga de representar a un jugador en la sesión de juego. Al iniciar la aplicación se crea una instancia para representar al jugador local y luego de conectarse en red, se solicitan las instancias de los demás jugadores conectados para almacenarlas en una lista, la cual permitirá conocer todas sus características, las cuales se pueden ver en la Figura 7.1.2.3.

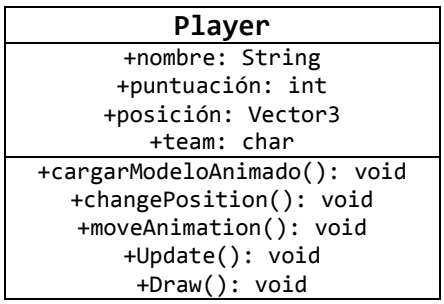


Fig. 7.1.2.3 Diagrama de la clase Player.

Clase Networking: Esta clase permite la interconexión entre esta aplicación y otras instancias de ella que estén ejecutándose a través de una red local. Para realizar una conexión de red se utilizan los servicios de Windows Live para el envío y recepción de mensajes y de la identificación de los jugadores a través de la red local, a través de los métodos y atributos que existen en esta clase, los cuales pueden apreciarse en la Figura 9.1.2.4.

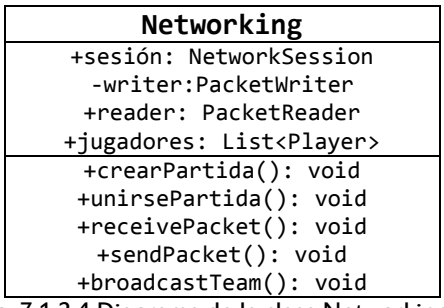


Fig. 7.1.2.4 Diagrama de la clase Networking.

Clase Partida: Esta clase administra la lista de jugadores conectados, que tipo de partida se está realizando (Eliminación, Eliminación con equipos, Capturar la bandera y Entrenamiento) y valida que, según el tipo de partida, se cumpla el objetivo de culminación respectivo. Podemos ver un diagrama con sus atributos y métodos en la Figura 7.1.2.5.

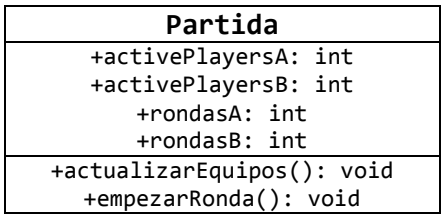


Fig. 7.1.2.5 Diagrama de la clase Partida.

Clase Bullet: Esta clase representa cada esférica que se dispara desde los jugadores y contiene información acerca de su origen, punto de destino, y distancia de impacto desde ella hasta un objeto. Un diagrama con sus atributos y métodos se puede apreciar en la Figura 7.1.2.6.

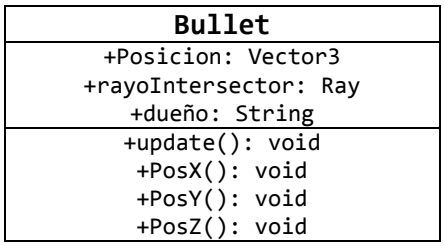


Fig. 7.1.2.5 Diagrama de la clase Bullet.

Clase TelePrompter: Esta clase contiene un conjunto de mensajes que se van intercambiando los jugadores a lo largo de la partida, permitiendo ver los últimos 5 mensajes y desapareciendo gradualmente de la pantalla. Un diagrama con sus atributos y métodos se puede apreciar en la Figura 7.1.2.7.

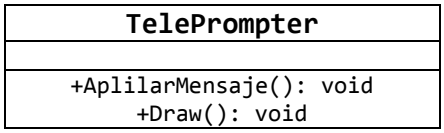


Fig. 7.1.2.7 Diagrama de la clase TelePrompter.

Clase Boxes: Esta clase representa un Bounding Box y los métodos necesarios para conocer sus intersecciones con otros objetos y su Bounding Sphere equivalente. Un diagrama con sus atributos y métodos se puede apreciar en la Figura 7.1.2.8.

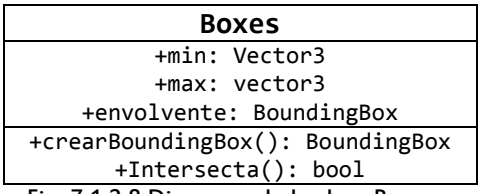


Fig. 7.1.2.8 Diagrama de la clase Boxes.

Clase FontManager: Esta clase permite mostrar texto sobre la pantalla utilizando una fuente en particular. Un diagrama con sus atributos y métodos se puede apreciar en la Figura 7.1.2.9.

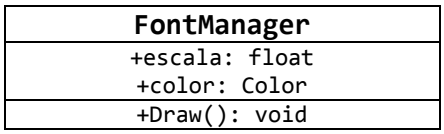


Fig. 7.1.2.9 Diagrama de la clase Boxes.

Clase ItemManager: Esta clase permite incluir objetos autónomos a los escenarios del juego. Además de esto se pueden verificar interacciones como colisiones entre los objetos. Un diagrama con sus atributos y métodos se puede apreciar en la Figura 7.1.2.10.

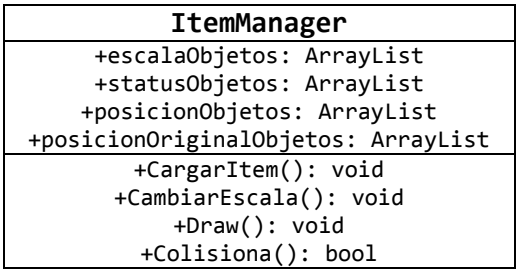


Fig. 7.1.2.10 Diagrama de la clase Boxes.

Clase InputHelper: Esta clase permite incluir objetos autónomos a los escenarios del juego. Además de esto se pueden verificar interacciones como colisiones entre los objetos. Un diagrama con sus atributos y métodos se puede apreciar en la Figura 7.1.2.11.

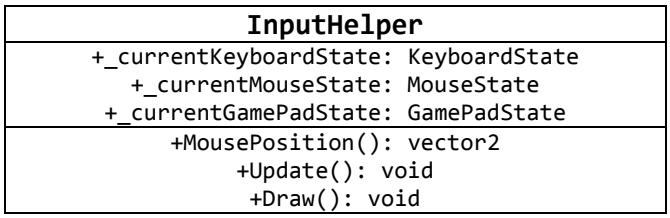


Fig. 7.1.2.11 Diagrama de la clase InputHelper.

7.1.3 Diagrama de Actividades

A continuación se muestra (en la Figura 7.1.3) el diagrama de actividades que representa las interacciones existentes entre los estados generados en la clase principal de la aplicación; desde que inicia la aplicación, siguiendo por la posible toma de decisiones en cuanto a crear una partida o ingresar a una partida existente, el ingresar al juego como tal; la posibilidad de repetir el ciclo ingresando a otra partida o creando una partida propia; y por último la culminación del programa.

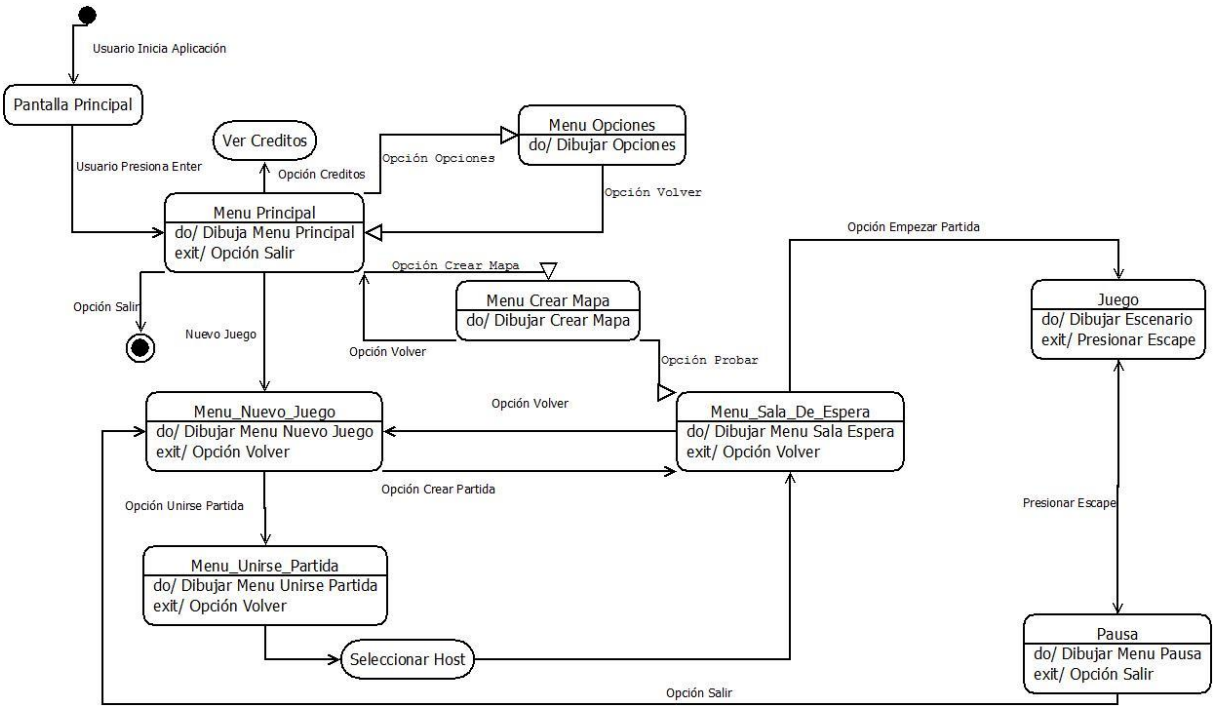


Fig. 7.1.3 Diagrama de Actividades.

7.2 Desarrollo de la Aplicación

El ciclo de vida del videojuego se divide en tres fases. La fase de inicialización, en la cual se crean todas las estructuras de datos para el correcto funcionamiento del juego y se cargan en memoria los objetos y recursos del juego. Luego viene la fase de desarrollo, la cual es la fase más compleja del juego, ya que es responsable de todos los eventos que se suscitan mientras se está ejecutando la aplicación. Por último, viene la fase de finalización, en la cual se libera de la memoria todos los recursos que se utilizaron para la realización del juego.

7.2.1 Fase de Inicialización

En esta fase se inicializa el entorno de XNA a través del método constructor de la clase principal (*Game1*) y de los métodos *Initialize()* y *LoadContent()*.

El método constructor solo se utiliza para crear un objeto *GraphicsDeviceManager*, el cual permite interactuar con el hardware gráfico y desplegar cualquier componente visual, sea tanto texto como geometrías, tales como las que se utilizan para desplegar los modelos 3D.

El método *Initialize()* se invoca para cargar todos los recursos necesarios por la aplicación, entre ellos, recursos gráficos como texturas, elementos estáticos como menús, y recursos lógicos como los objetos para almacenar a los jugadores y un objeto *Networking* para manejar todo el intercambio en red entre los jugadores. Este objeto *Networking* se encargara de las siguientes funciones:

- Gestionar la conexión del jugador con su cuenta de Xbox Live

- Crear una partida a través de una conexión de red de área local
- Consultar las partidas que están en funcionamiento en la red local
- Permitir conectarse a una partida que se encuentre activa.

En el método *LoadContent()* se procede a asignar todos los recursos en sus respectivos objetos, así como inicializar estructuras importantes en el jugador local, como lo son crear las cajas y esferas envolventes (*BoundingBoxes* y *BoundingSphere* respectivamente) que serán usadas en la próxima fase para el cálculo de las colisiones.

7.2.2 Fase de Ejecución

Una vez inicializada la aplicación comenzara el despliegue de menús para dirigir al jugador en las diversas opciones que tiene el juego. Luego de haber escogido el tipo de juego correspondiente y verificar que todos los jugadores están conectados en la partida, se procede con el inicio del juego, donde cada evento de los jugadores, tales como cambiar de posición, disparar, o salirse de la partida, generan eventos que son comunicados a los demás jugadores por medio de mensajes que se envían a través del objeto *Networking* creado en la fase de inicialización.

Dependiendo de la habilidad de los jugadores, el juego culminará cuando se cumplan los objetivos planteados al principio de la misma, los cuales pueden ser eliminar a todos los jugadores del equipo contrario, eliminar a todos los jugadores del mapa o resguardar cierta cantidad de zonas de los oponentes.

Las mecánicas que se desarrollan dentro de esta fase son las siguientes:

- Detección de colisiones: La detección de colisiones se lleva a cabo en tres etapas:

  Colisiones jugador – escenario: Este tipo de colisiones es verificada por cada jugador para evitar que este atraviese cualquier obstáculo, esto se hace mediante el archivo del mapa que fue cargado antes de empezar la partida (como se explicó en la sección Almacenamiento), el cual es revisado cada vez que el jugador se mueve para verificar si en la dirección a la que se va a desplazar existe un obstáculo. En caso afirmativo, realizamos el mismo cálculo, pero esta vez intentando realizar el desplazamiento solo en el eje X, si persiste la colisión, se verifica por último el desplazamiento únicamente en el eje Z, como se puede observar en la figura 7.2.2.1. Esta verificación por ejes se realiza para mantener el movimiento en el eje en donde no exista colisión, y así poder bordear objetos.

```
foreach (Boxes model in mapaColisiones){
    if (model.mapCollision(p))
        cambio = false;

    if (cambioX){
        if (model.mapCollision(new Vector3(p.X,p.Y, posAnterior.Z)))
            cambioX = false;//no Logre moverme en X
        if (cambioZ)
            if (model.mapCollision(new Vector3(posAnterior.X, p.Y, p.Z)))
                cambioZ = false;//no Logre moverme en Z

    if (cambioX){
        p = new Vector3(p.X, p.Y, posAnterior.Z);
    }
    else if (cambioZ){
        p = new Vector3(posAnterior.X, p.Y, p.Z);
    }
    else{
        p = posAnterior;
    }
```

Fig. 7.2.2.1 Código de colisiones Jugador-Escenario.

Si es posible realizar algún desplazamiento, se procede a enviar la posición de este jugador a los demás clientes para actualizarlo.

Los obstáculos usados son formas rectangulares que se adaptan a la estructura *BoundingBox* de XNA, cuyo tamaños son los indicados en el editor, y se les incluye un tamaño adicional de 0.86 centímetros por lado para validar las detecciones de colisiones entre las esferas y el escenario, ya que las esferas miden 1.72cm. De esta forma la verificación de colisiones puede hacerse entre un rayo y el objeto simplemente, ya que el volumen de la esfera ya es considerado en el tamaño extra de los objetos.

Colisiones jugador – jugador: Un BoundingBox es generado en la posición actual de cada jugador, tomando como centro la posición actual del jugador, como puede verse en la figura 7.2.2.2. Se verifica si existe o no colisión entre el BoundingBox del jugador local y los demás jugadores. Al igual que en el caso de las colisiones jugador – escenario, se verifica si es posible avanzar solo en uno de los ejes.

```
BoundingBox y;
BoundingBox estePlayer =
    new BoundingBox(new Vector3(p.X-0.2f, p.Y-1.6f, p.Z-0.2f),
    new Vector3(p.X+0.2f, p.Y, p.Z+0.2f));

foreach (Player x in manejadorRed.jugadores)
{
    y = new BoundingBox(
        new Vector3(x.Position.X-0.2f, x.Position.Y-1.6f, x.Position.Z-0.2f),
        new Vector3(x.Position.X+0.2f, x.Position.Y, x.Position.Z+0.2f));
    if (y.Intersects(estePlayer))
        cambio = false;
    if (cambioX)
    {
```

Fig. 7.2.2.2 BoundingBox de los jugadores.

Colisiones jugador-esferas: La detección de colisiones entre el jugador local y las esferas activas que existan en el escenario se realiza utilizando las cajas envolventes del jugador y un rayo generado entre la posición anterior y la posición actual de la bala. Primero se determina si el rayo generado colisiona con algún jugador y en caso afirmativo, se calcula la distancia de la colisión con respecto al punto inicial del rayo (como se ve en el código de la Figura 7.2.2.2), la cual se almacena en cada bala. Luego de eso se revisa por cada elemento del escenario si dicho elemento es impactado por el rayo generado en el paso anterior. En caso de ser afirmativo y en caso de que la distancia de la esfera al obstáculo sea menor que la distancia de la esfera al jugador, se procede a eliminar dicha bala del sistema.

```
    DateTime dateTimeNow2 = DateTime.Now;
    foreach (Bullet bull in bullets){
        bull.Update(dateTimeNow2);
        Nullable<float> dist = bull.rayoIntersector.Intersects(BBPlayer1);

        if ((manejadorRed.JugadorLocal.active) &&
        (!bull.owner.Equals(manejadorRed.JugadorLocal.nombre))
        && (dist != null) && (dist.Value <= bull.distanciaRayo)){
            bull.distanciaImpacto = dist.Value;
        }
    }
    //Si alguna esfera choca con alguna pared, la elimino
    foreach (Boxes model in mapaColisiones){
        float posibleEmisor = float.MaxValue;
        Vector3 minEmisor = new Vector3();
        foreach (Bullet bull in bullets){
            Nullable<float> dist2 =
            bull.rayoIntersector.Intersects(model.boundingBox);
            if (bull.esferaBajoPiso()
                || ((dist2 != null) && (dist2.Value < bull.distanciaImpacto)
                && (dist2.Value <= bull.distanciaRayo))
                || ((dist2 != null) && (dist2.Value <= bull.distanciaRayo))
                || esferaFueraDelMapa)
            {
                if ((dist2 != null) && dist2.Value < posibleEmisor)
                {
                    posibleEmisor = dist2.Value;
                    minEmisor = bull.actualPosition;
                }
                eliminables.AddLast(bull);
                break;
            }
        }
        //Coloco el sistema de partículas donde hubo un choque de pared
        if (posibleEmisor != float.MaxValue){
            particleSystem.changeEmissor(minEmisor);
            particleSystem.Activate();
        }
        //Elimino las esferas que hayan chocado
        foreach (Bullet b in eliminables){
            bullets.Remove(b);
        }
    }
```

Fig. 7.2.2.3 Código de colisiones Jugador-Balas.

- Lanzamiento de proyectiles: Para el manejo de la física de los proyectiles se aplicaron formulas sencillas de física dentro de la clase *Bullet*, para poder calcular la posición de la esfera dado un momento inicial, un momento actual y la posición inicial de la esfera (donde se disparó). Además de esto, antes de actualizar la posición actual, se conserva esa posición "anterior" para poder conocer cuánto avanzo el proyectil desde la iteración anterior, con el fin de generar un rayo entre la posición anterior y la posición actual de la esfera, y así poder saber si colisionó un objeto (como lo podemos ver en la Figura 7.2.2.4). En caso de Colisionar con dos o más objetos o jugadores, se verifican la distancia a la que colisionó con cada objeto, tomando el objeto más cercano a la posición inicial (posición anterior) de la esfera.

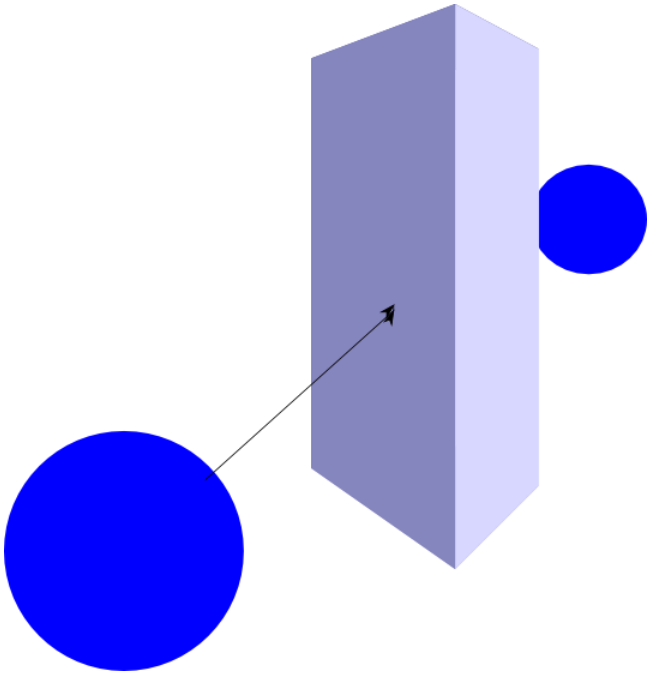


Fig. 7.2.2.4 Interacción Rayo-Objeto entre dos posiciones de una esfera.

- Manejo de modelos 3D: Para el manejo de los modelos animados se utilizó la librería XNAnimation [12], la cual permite tomar un modelo en formato FBX (formato gráfico de Autodesk para representar geometrías), y cargar las coordenadas de un mallado 3D, además un conjunto de transformaciones que permiten que un esqueleto dentro del mismo modelo pueda ser animado, y por lo tanto permitir mover las diversas partes del modelo. En dicho archivo FBX están almacenados en cuadros (*frames*) todas las transformaciones de los movimientos que hará el modelo. Además de esto, se utiliza un archivo XML (como el fragmento mostrado en la Figura 7.2.2.4), el cual contiene primero como se llama la secuencia completa de animaciones y la velocidad con la que se reproducirán estos *frames*. Luego de esto, por cada sub-animación, se indica el nombre de la misma y desde que cuadro inicial y final corresponde a cada sub-animación.

```
<Animations>
  <Animation>
    <Name>Take 001</Name>
    <Framerate>30</Framerate>

    <SplitTask>
      <Name>Walk</Name>
      <StartFrame>2</StartFrame>
      <EndFrame>14</EndFrame>
    </SplitTask>

    <SplitTask>
      <Name>Run</Name>
      <StartFrame>16</StartFrame>
      <EndFrame>26</EndFrame>
    </SplitTask>

    <SplitTask>
      <Name>Jump</Name>
      <StartFrame>28</StartFrame>
      <EndFrame>40</EndFrame>
    </SplitTask>
  </Animation>
</Animations>
```

Fig. 7.2.2.5 Asignación de animaciones de un modelo en formato FBX.

- Carga circunstancial de objetos: Cuando se selecciona el modo "Captura la Bandera" el juego se encarga de cargar un conjunto de ítems que se usaran para dicho modo.

```
if (juego.tipo == Partida.tipoPartida.CAPTURAR_BANDERA)
{
    manejadorObjetos.CargarItem("Models\\flag",      "Models\\ford",      new
Vector3(25.0f, 0.0f, -18.0f));
    manejadorObjetos.cambiarEscala(0, new Vector3(0.05f, 0.05f, 0.05f));
}
```

Fig. 7.2.2.6 Carga de objetos.

Además de esto, en este modo se implementó la asignación de una bandera a un jugador cuando este choca con ella, con el fin, de cuando dicho jugador sea alcanzado por una esfera, suelta la bandera justo en el lugar donde fue alcanzado.

```
if(manejadorRed.JugadorLocal.tieneFlag==true)
{
   int posObj = 0;//la posicion del objeto actual
   foreach (ItemManager.tipoObjeto tO in manejadorObjetos.tipoObjetos)
   {
      if     (     (tO     ==     ItemManager.tipoObjeto.Bandera)     &&     (
(bool)manejadorObjetos.statusObjetos[posObj]==false))
      {
         manejadorObjetos.statusObjetos[posObj] = true;
         manejadorObjetos.posicionObjetos[posObj]                          =
manejadorRed.JugadorLocal.Position;
      }
      posObj++;
   }
   manejadorRed.sendPacket('U',' ');//Mando un mensaje para soltar la bandera
}
```

Fig. 7.2.2.7 Código para soltar la bandera.

El cargador de objetos de XNA es sumamente restrictivo, solo modelos creados con ciertas opciones son cargados correctamente. Luego de intentar con varios programas de diseño 3D, solamente fueron cargados correctamente los modelos exportados con Blender versión 2.68, ya que desde su versión 2.59 hasta la 2.68 incluye opciones específicas para exportar modelos FBX para su uso con XNA. Esta opción de Blender fue removida desde la versión 2.69 hasta la actual 2.70.

- **Sonidos:** Los jugadores pueden enviar comandos de voz a los otros jugadores. Para simular el volumen de la voz, se calcula la distancia actual del jugador que envío el sonido con el jugador local y basado en esta distancia se asigna el volumen al sonido a reproducir. Así, si un jugador está más lejos de cierta distancia, no escuchara el sonido, y a medida que esté más cerca lo escuchara con mayor volumen. Lo mismo aplicamos para el sonido de los marcadores disparando, con la única diferencia de que a partir de cierta distancia el volumen se coloca constante para hacer que todos los disparos se escuchen así sea en un volumen bajo.

```
if (messageType == 'S')
{
    String com = this.reader.ReadString();
    float volumen=0.0f;
    volumen=Vector3.Distance(this.JugadorLocal.Position,
            jugadorActual.Position);
    volumen = volumen / 30.0f; //solo es audible a menos de 30 metros
    if (volumen < 0)
       volumen *= -1.0f;
    volumen = 1.0f - volumen;
    if (volumen > 0.0f)
    {
       comandosDeVoz[com[0]-48, com[1]-48,
       partidaActual.idioma].Play(volumen, 0.0f, 0.0f);
    }
}
```

Fig. 7.2.2.8 Asignación del volumen según la distancia del emisor.

Se utilizaron dos tipos de sonidos, canciones y sonidos cortos para eventos. La carga de cada tipo de sonidos se puede observar en el siguiente fragmento de código:

```
comandosDeVoz[0, 1, 0] = Content.Load<SoundEffect>("Sounds/Radio/Yes");
deFondo[0] = Content.Load<Song>("Sounds/01 Lights, camera and action");
```

Fig. 7.2.2.9 Carga de sonidos.

Las canciones son seleccionadas al azar y se reproducen en todas las pantallas de menús.

```
if (MediaPlayer.State == MediaState.Stopped)
{
    int temp = new Random(DateTime.Now.Second).Next(0, deFondo.Length);
    while (temp == actualSong)
        temp = new Random(DateTime.Now.Second).Next(0, deFondo.Length);
    actualSong = temp;
    MediaPlayer.Play(deFondo[temp]);
}
```

Fig. 7.2.2.10 Reproducción aleatoria de canciones.

Los comandos de voz fueron creados utilizando la *Grabadora de sonidos de Windows* y luego convirtiendo los archivos de MP4 a MP3 utilizando el software *Free M4a to MP3 Converter*[1].

- **Idiomas:** Se creó una clase virtual *Languages*, en la cual se realiza la declaración de todas las variables referentes a los textos en las pantallas. De esta clase base, deriban las clases *English* y *Spanish*, las cuales instancian dichas variables con el texto en el idioma correspondiente. Para realizar el cambio de idiomas simplemente podemos crear una variable de tipo *Languages* y asignarle una variable de tipo *English* o *Spanish*. Esta forma de manejo de idiomas facilita la inclusión de nuevos idiomas, ya que solamente es necesario agregar la nueva clase derivada correspondiente y realizar la traducción de los textos.

- **Modelado 3D:** El modelado de los mapas y la creación sus texturas desde cero fue realizado utilizando el software *Blender* para el modelado 3d y *Paint.Net* para las texturas. Este trabajo tomó un tiempo considerable ya que no teníamos experiencia alguna en el uso de programas de modelado, añadido a los innumerables intentos de exportar el modelo en un archivo que XNA cargase correctamente. Se realizaron pruebas con 3 programas de modelado diferentes (*Sketchup*, *Autodesk* y *Revit*), varios pluggins para exportar tanto en .FBX como .X sin lograr resultados favorables. Únicamente fue posible con *Blender* en una versión antigua del mismo, en la cual se incluían opciones específicas para la exportación a XNA.

[1] www.audiotoaudio.com

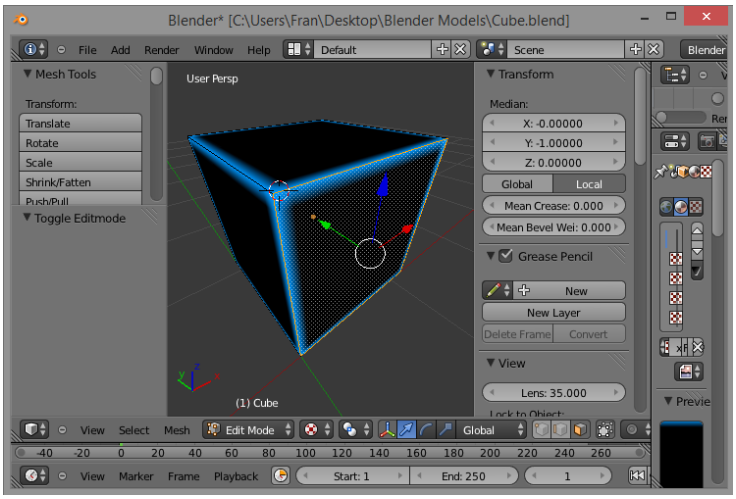


Fig. 7.2.2.11 Modelado en Blender.

La cantidad de modelos de personajes 3d gratuitos que pudimos encontrar que incluyan animaciones es escaso, más aun tomando en cuenta que estos personajes deben adecuarse a la temática del videojuego tanto en apariencia como en sus animaciones. El primer modelo cargado al juego es el que se muestra en la figura 7.2.2.12 el cual contenía animaciones pero no era acorde con la temática.

Fig. 7.2.2.12 Primer modelo de jugador.

Luego utilizamos varios modelos gratuitos que tuviesen esqueleto (para poder animarlos) e intentamos realizar nosotros mismos las animaciones de un modelo más acorde con el juego, utilizando el software *3D Max 2012* y *FragMotion*.

Decidimos finalmente modificar el modelo existente, modificando su textura y escalando el modelo en el eje Y. De esta forma el personaje es más acorde con la temática. En la figura 9.2.2.13 se puede observar el cambio realizado en la textura así como el resultado final.

Fig. 7.2.2.13 Cambio de texturas y escala del modelo.

7.2.3 Fase de Finalización

Luego de culminar una partida, el usuario que creó la partida es conducido de nuevo a una pantalla de menú donde puede escoger volver a crear otro encuentro, o en el caso de los jugadores conectados remotamente a la partida, se les invita a unirse de nuevo a otra partida que este activa en el momento actual.

En caso de que el jugador desee finalizar su sesión al seleccionar la opción Salir del juego, lo cual invoca al método *UnloadContent()* (ver Figura 7.2.3), donde se precederá a vaciar de la memoria todas las estructuras de datos utilizadas por el juego y finalizara la ejecución del programa.

```
protected override void UnloadContent()
{
        // En el caso de que la música aun suene, se detiene el reproductor
        if (MediaPlayer.State == MediaState.Playing) MediaPlayer.Stop();
        disparo = null;
        deFondo = null;
        dePlay = null;
        jugador1 = null;
        Content.Unload();
}
```

Fig. 7.2.3 Código de liberación de recursos.

7.3 Motor de Juego

El videojuego fue creado utilizando el framework XNA, el motor de juego fue desarrollado iterando entre sus dos funciones básicas: Update() y Draw(). Sin embargo, se desarrollaron un conjunto de clases para poder modelar los diversos objetos que interactúan en el juego. A continuación se verá un algoritmo que representa la interacción de esas dos funciones y todas las llamadas internas que se realizan para ejecutar el motor de juego.

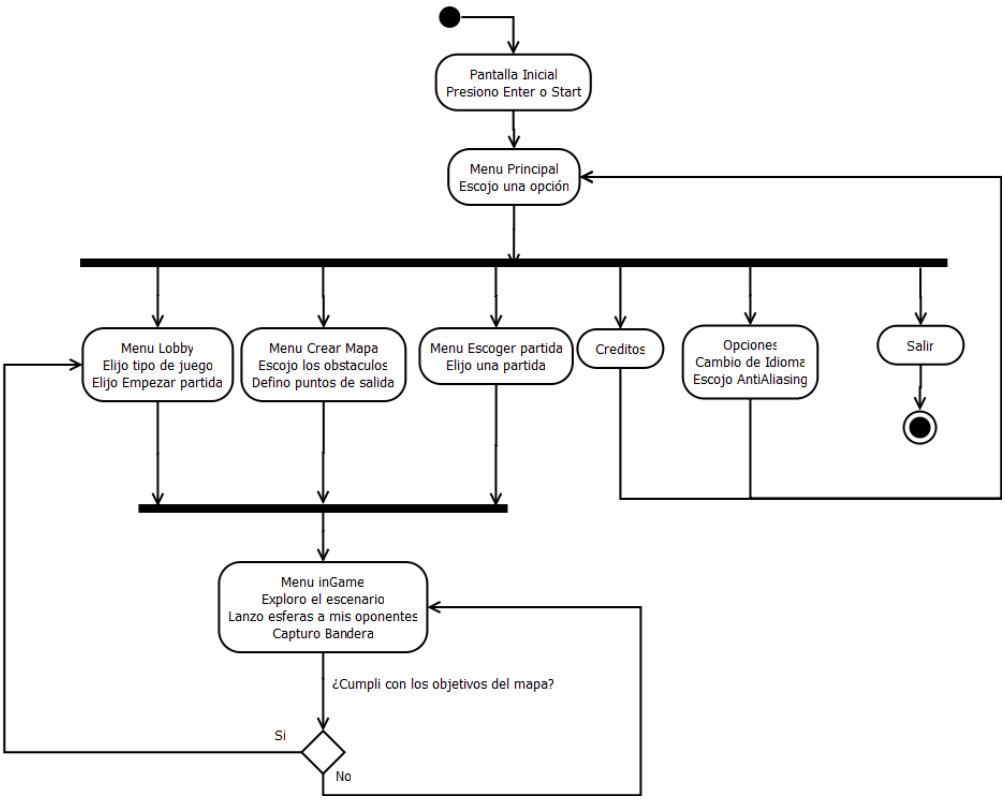


Fig. 7.3.1 Diagrama de eventos del motor del juego.

7.4 Red

Las interacciones de red se realizan a través del framework XNA y el servicio **Game for Windows - Live**, sin embargo se desarrollaron un conjunto de interfaces para poder manejar la transmisión de paquetes correctamente y el entendimiento de estos en una clase llamada "*Networking*"

7.4.1 Métodos utilizados en la clase networking.cs

- crearPartida(): Crea una sesión de red disponible para 31 personas configurándolo para SystemLink (permite conectarse a través de una red local tanto PC's como sistemas Xbox360).
- seleccionarPartida(menuComponent listaJuegos): Busca todas las sesiones de red que tengan capacidad para jugar a través de SystemLink y que tenga un espacio disponible. Estas se agregan a un objeto menuComponent y se devuelve al programa principal para ser mostrado por pantalla.
- unirsePartida(int partidaSeleccionada): permite al jugador conectarse con la sesión previamente seleccionada.
- sendPacket(): esta función permite enviar un paquete determinado a todos los jugadores que estén conectados en la sesión. Existen varias versiones de esta función con diversos tipos de datos.
- empezarPartida(): luego de que todos los jugadores están listos, el servidor envía un mensaje a todos de que comienza la partida. Esto se hace cambiando el estado de la sesión con el método de XNA StartGame()
- broadcastTeam(char Tipo): esta función permite avisarle a todos los demás jugadores que nuestro personaje se ha cambiado de equipo.
- broadcastGame(int Tipo): esta función permite avisarle a todos los demás jugadores que la partida tendrá otro modo de juego.
- receivePackets(): esta función se encarga de esperar todo tipo de paquetes y trabajara acorde al tipo de paquetes que reciba:

Tipo de paquete	Función	Datos enviados en el paquete
B	Nueva esfera disparada	Vector de dirección y vector de posición
C	Mensaje de chat	Cadena de caracteres
F	Indicar que el jugador local logro obtener la bandera	Nada
G	Punto anotado por bandera	Nada
H	Indicar que cierto jugador logro acertar un disparo	Quien disparó la bala

J	Salto de un jugador	Posición y rotación de la cámara
M	Mapa actual a los clientes conectados	Cadena de caracteres representativa del mapa
P	Cambio de posición de un jugador	Posición y rotación de la cámara
R	Reiniciar posición de la bandera	Nada
S	Indicar cambio de equipo de un jugador	Caracter que representa el equipo al que se cambió el jugador
S	Comando de voz	Entero representativo del comando de voz activado
T	Cambio del modo de juego	Modo de juego
U	Indicar que el jugador local soltó la bandera	Nada

- asignarEquipo(): asigna al jugador local un equipo según la cantidad de jugadores que haya y en que equipos se encuentren en ese instante.
- salirPartida(): finaliza la sesión actual utilizando el método de XNA Dispose(), además de limpiar correctamente las colecciones de jugadores conectados y esferas.
- Update(): actualiza la sesión. Este método es el más importante porque invoca al framework a actualizar todos los mecanismos de networking que se utilicen, ya se apara enviar o recibir paquetes.

7.5.2 Modelos

Para la utilización de los modelos animados se incluyó en el motor de juego la librería XNAnimation, la cual fue desarrollada por Bruno Evangelista [12]. Esta librería permite cargar en la aplicación modelos en formato FBX y animarlos mediante el uso de un script, el cual indica que secciones del modelo deben ser modificadas para realizar dicha animación. Como limitante, el modelo utilizado debe contener animaciones esqueléticas, es decir, modelos que posean objetos que guíen el movimiento del modelo entero.

7.6 Software Secundario

Para la realización de este videojuego se utilizaron los siguientes paquetes de software:

- Google Sketch Up 8: En esta aplicación se crearon los escenarios del juego. Se escogió esta herramienta por su facilidad de uso y la capacidad de agrupar polígonos para crear objetos más complejos que serán usados como barreras en el juego.

- Autodesk 3D Studio Max 2012: Este paquete permitió crear y editar los modelos 3D que se utilizaron en el juego. Se editaron en esta aplicación por la capacidad de crear animaciones para los modelos.
- Microsoft Visual Studio 2010: Este videojuego fue desarrollado enteramente a través de esta interfaz de desarrollo (IDE), ya que esta permite desarrollar aplicaciones bajo el lenguaje C# utilizando el framework XNA.
- Microsoft Word 2010: Con esta aplicación, fue posible redactar y editar todos los documentos necesarios para planificar y desarrollar este videojuego.
- Adobe Photoshop CS4: Esta aplicación permite editar imágenes y crear las texturas e imágenes que se utilizaron en el juego, como lo son las imágenes que aparecen en los diversos menús.
- Paint.Net: Utilizado también para crear y modificar texturas e imágenes.
- Grabadora de Sonidos de Windows: Esta aplicación se utilizó para grabar los sonidos de los comandos de voz.
- Free M4a to MP3 Converter: Utilizado para convertir de formato MP4 a MP3.
- NetBalancer: Este programa se utilizó para medir la carga de red
- VMWare Workstation 8: Se crearon máquinas virtuales para poder realizar pruebas locales durante el desarrollo. Se eligió este software por su soporte para tarjetas de video.

Capítulo 8. Pruebas y resultados

Pruebas de rendimiento de red:

Se realizaron pruebas del uso de ancho de banda de red, el cual fue medido con el software *NetBlancer* y con la ayuda de máquinas virtuales creadas en la aplicación *VMWare Workstation*.

La primera prueba fue con dos instancias del juego, arrojando un uso máximo de 9.2 KBps de carga y descarga en cada máquina. Este uso máximo se da cuando el jugador se desplaza y dispara, ya que por cada desplazamiento y por cada disparo se envía un paquete de red.

Se agregaron jugadores progresivamente hasta alcanzar 8 jugadores simultáneos, observando un aumento en el uso de red prácticamente lineal, sumando 9.2 KBps al uso máximo por cada jugador agregado, con un aumento adicional de 0.2 a 0.5 KBps atribuibles al aumento en la complejidad del manejo de la sesión por parte de XNA.

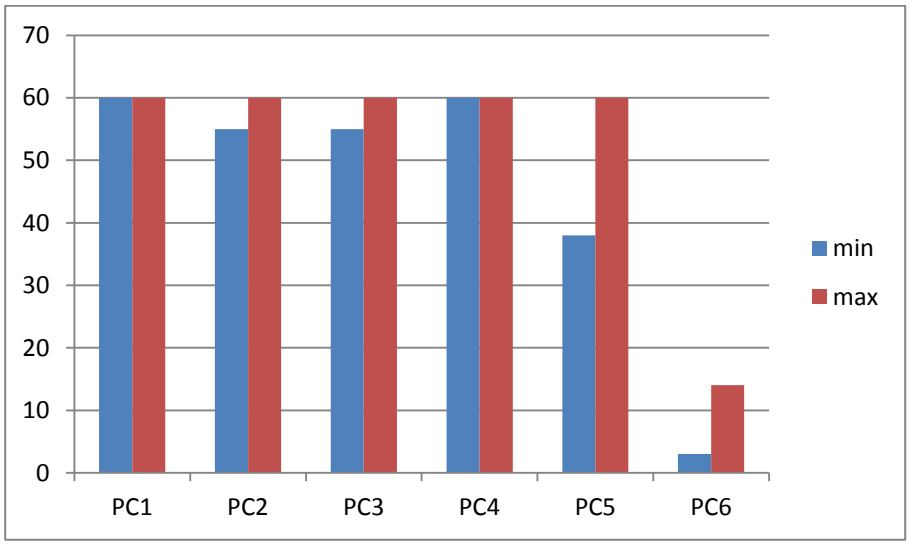
Pruebas de rendimiento gráfico

El objetivo de esta prueba es medir en cuadros por segundo el rendimiento del videojuego. Para la realización de estas pruebas se utilizaron 6 computadores con especificaciones diferentes, las cuales son se describen a continuación:

ID	Especificaciones
PC 1	Procesador Intel i7 2.8GHz 16GB de RAM DDR3 Nvidia GTX 760 2Gb VRAM
PC 2	Máquina Virtual corriendo sobre la PC 1 1 procesador* 1GB de RAM
PC 3	Máquina Virtual corriendo sobre la PC 1 2 procesadores* 1GB de RAM
PC 4	Procesador Intel Dual Core 2.7GHz 3GB de RAM DDR2 Nvidia 9800 GT 1Gb VRAM
PC 5	Procesador Intel i5 3.3GHz 4GB de RAM DDR3 Nvidia GT 520 1Gb VRAM
PC 6	Procesador Intel Dual Core 2.1GHz (laptop) 3GB de RAM DDR2 Intel Express Serie 4

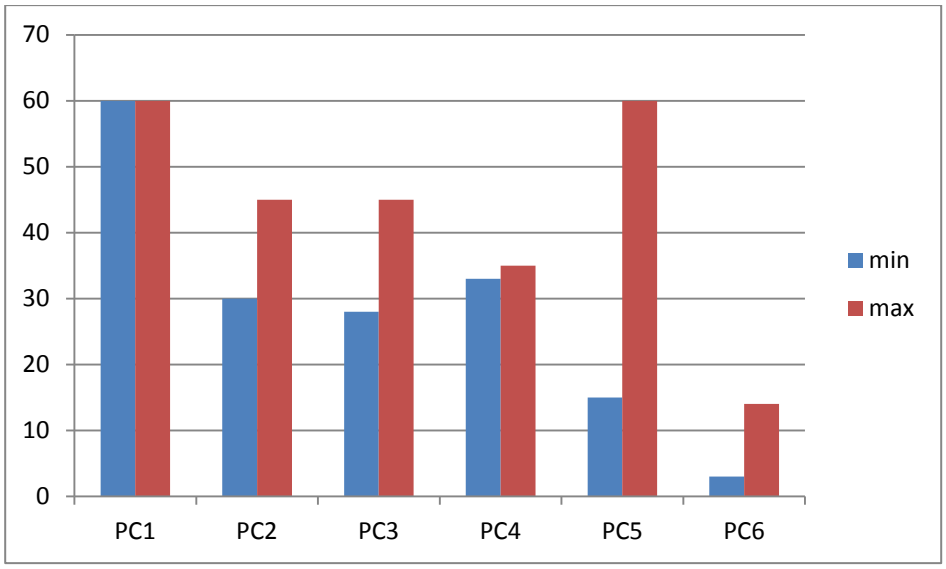
*Cada procesador en las opciones de *VMWare*, realmente significa que se está usando un hilo de los dos hilos que puede gestionar cada procesador físico de la tecnología i7.

Primera prueba: se utilizó un mapa aleatorio que se había creado durante el desarrollo del videojuego, sin cumplir características especiales más allá de ser un mapa grande con una densidad de obstáculos media. Los resultados del test medidos en cuadros por segundo fueron los siguientes:



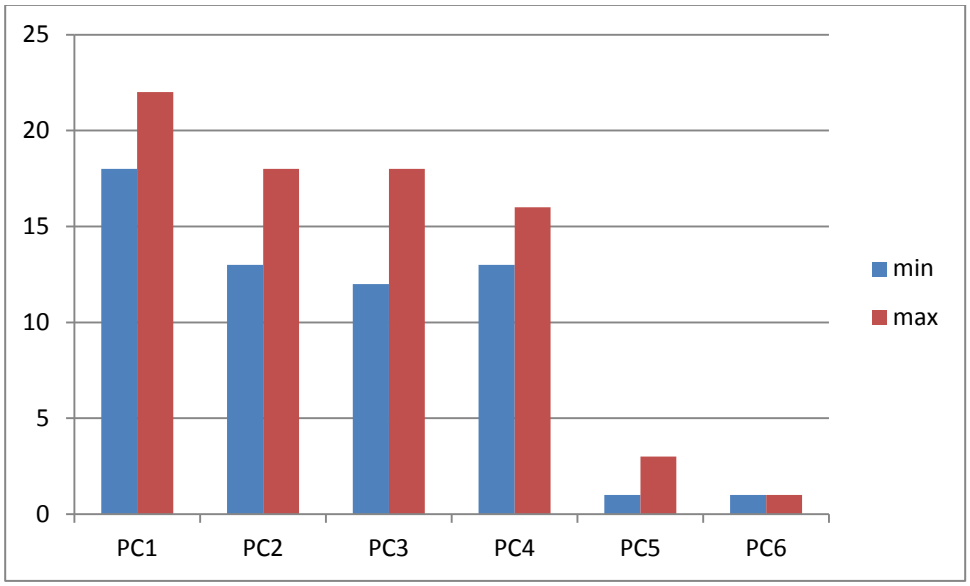
El rendimiento es el esperado, pues los cuadros por segundo están limitados dentro del juego a un máximo de 60 cuadros por segundo para evitar artefactos visuales mediante la sincronización vertical y además controlar el número máximo de paquetes enviados por red. El resultado del PC6, a pesar de tener un conteo de cuadros por segundo bajo, es el esperado por tener vídeo integrado y ser este algo antiguo, el cual ofrece pocas ventajas al procesamiento de modelos 3D.

Segunda prueba: Esta vez, el mapa elegido es uno con las dimensiones máximas permitidas en el videojuego, el cual se le colocaron obstáculos en un cuarto de la totalidad del mapa, lo cual puede considerarse como un mapa lleno de objetos que realmente sea práctico al momento de jugar. Se observaron los siguientes resultados:



El resultado es satisfactorio, dado que el PC 4 se mantiene en un estado en el que se puede jugar sin problemas, manteniéndose por encima de los 24 cuadros por segundo necesarios para una visualización continua, a pesar de que su tarjeta de vídeo fue lanzada al mercado en el 2008, teniendo esta tecnología seis años de antigüedad. La tarjeta de vídeo presente en el PC5 no está diseñada con el fin de acelerar videojuegos ya que es una tarjeta de vídeo de la gama baja de *Nvidia*. Estas tarjetas están diseñadas para acelerar trabajos multimedia, por lo que es totalmente comprensible el decremento en los cuadros por segundo al aumentar la cantidad de objetos en el mapa.

Tercera prueba: En esta ocasión se utilizó un mapa de nuevo con las dimensiones máximas permitidas, pero esta vez, la mitad de las posiciones del mapa se encuentran ocupadas por obstáculos. En un mapa con estas características la jugabilidad sería sumamente limitada por la falta de espacio. Los resultados fueron los siguientes:



No se esperaba una caída en los cuadros por segundo tan dramática, por lo cual se procedió a verificar las posibles causas:

Primeramente se desactivo la detección de colisiones, ya que este es el proceso asociado con la cantidad de objetos en el mapa con más carga de procesamiento dentro del juego. Se observaron exactamente los mismos resultados con una variación máxima de 2 cuadros por segundo.

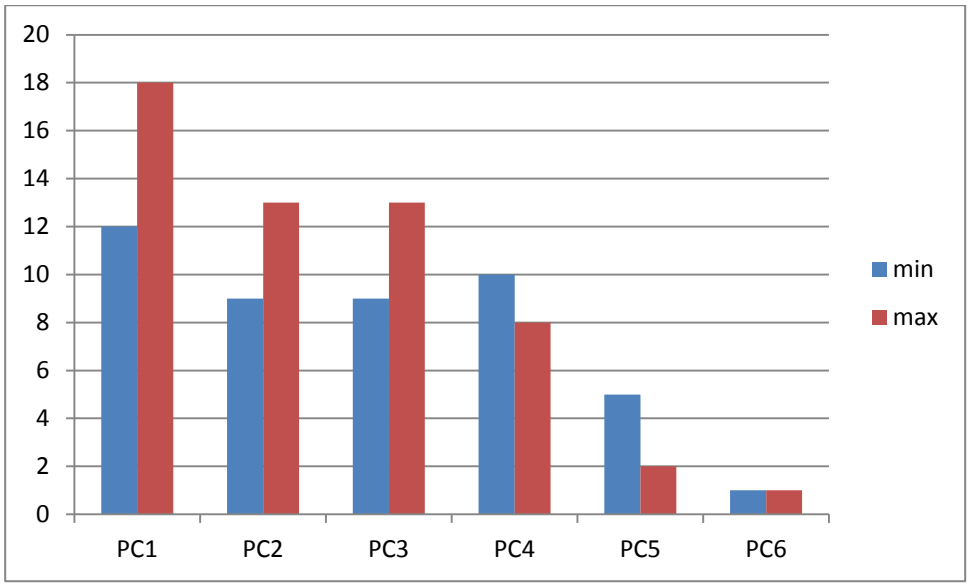
Luego se procedió a disminuir la resolución de las texturas de los obstáculos, llevándolas de un tamaño de 512 por 512 hasta 16 por 16. De nuevo se obtuvieron los mismos resultados. Además se realizó una prueba desactivando la detección de colisiones y con esta nueva resolución de las texturas observando de nuevo el mismo resultado.

Finalmente se realizó una prueba activando la detección de colisiones, pero esta vez, desactivando el desplegado del mapa. Los cuadros por segundo volvieron a los valores anteriores, demostrando así que la caída de rendimiento viene dada por la cantidad de polígonos desplegados. A pesar de la simpleza de los modelos, la cantidad de obstáculos es sumamente elevada, aunado al

modelo del piso, el cual fue necesario modelar con volumen y no un plano únicamente (al desplegarlo se mostraban artefactos visuales, los cuales se corrigieron al agregar volumen a los modelos).

Dado que las dimensiones máximas del mapa son de 70 por 50, el número de caras para el despliegue del piso es de 70x50x8 = 28.000. Ya que el mapa solo tiene la mitad del espacio ocupado por objetos, pero los objetos colocados son de doble altura (hechos con dos cubos), el total de caras para los objetos es el mismo que para el piso, sumando 56.000 caras. Cada cara contiene 2 triángulos, por lo que serían en total 112.000 triángulos a dibujar.

Cuarta prueba: Llevando al extremo la cantidad de obstáculos, se colocaron en cada una de las posiciones del mapa, solo dejando libres los puntos de inicio de cada equipo, obteniendo los siguientes resultados:



Como se esperaba, se obtuvieron resultados similares a la prueba anterior, de nuevo, atribuible a la cantidad de polígonos desplegados. También se hicieron pruebas desactivando la detección de colisiones y reduciendo la resolución de las texturas obteniendo los mismos resultados.

Consumo de memoria:

El consumo máximo de memoria RAM del videojuego es de 155 MB. La versión de distribución del videojuego ocupa en memoria secundaria 119MB.

Capítulo 9. Conclusiones y trabajos futuros

Al momento de finalizar este trabajo de investigación se logró culminar el desarrollo de un videojuego utilizando la metodología propuesta, además de una serie de conclusiones en varios aspectos.

Con respecto al desarrollo e implementación, se obtuvieron diversos resultados a lo largo del trabajo, los cuales se describen a continuación.

En un primer momento del desarrollo estábamos utilizando la metodología clásica de desarrollo de videojuegos (ver Capitulo 4) pero luego se decidió hacer un cambio en cuanto a la metodología de desarrollo, ya que la metodología clásica resultaba costosa en cuanto al tiempo invertido en las fases y posibles correcciones y avances, por lo cual se optó por cambiar a la metodología Scrum (ver Capitulo 6), la cual está basada en múltiples iteraciones para mejorar incrementalmente el producto final.

Al finalizar las dos primeras iteraciones se obtuvo un documento de diseño de juego con el cual se iban a basar las decisiones a tomar en el desarrollo del juego. Además de eso se crearon los primeros prototipos de cargar modelos y navegación del escenario.

Luego de la tercera iteración comenzaron a surgir productos donde se estaban completando las interfaces gráficas de usuario y los diversos menús que existen en el juego.

Desde la 5ta iteración, se comenzó el desarrollo de la plataforma de red, por lo que se comenzó a probar el juego entre varios computadores y mejorando la interacción entre ellos.

En la 7ma y 8va iteración se comenzó el desarrollo de pantallas heredades, esta decisión mejoro en gran medida el desarrollo y depuración del código, ya que cada sección del juego estaba separada de las otras y no en un gran bloque monolítico.

A partir de la 9ma iteración, se buscó el aspecto definitivo del juego, donde se diseñaron las interfaces gráficas, se buscaron las texturas acordes para la creación de los escenarios y el desarrollo del modelo animado.

Entre las ventajas y desventajas que conseguimos con trabajar con Scrum, las más resaltantes fueron la facilidad con la que un proyecto puede adaptarse a cualquier planificación emergente con el incremento constante de ciclos, lo cual también fue una desventaja ya que no permitió que planteáramos tiempos límites para la culminación del mismo. Esta situación no es posible con la metodología tradicional, ya que si no se cumplen los tiempos establecidos el proyecto completo se detiene, en cambio aquí un equipo puede quedarse en una iteración varios ciclos desarrollando nuevos componentes pero nunca deteniendo el proyecto.

Un aspecto positivo de Scrum es que permite enfocarse en elementos específicos del desarrollo para luego integrarlos al producto final, por lo que nos permitió armar los componentes del motor de juego poco a poco sin tener que desarrollar otros aspectos en la misma iteración, lo cual según la metodología tradicional solo es válido si dichos desarrollos fueron planificados anteriormente y en ese orden establecido, lo cual no hay cabida a otros cambios.

Como aspecto negativo importante, está el hecho de que sin contar en el equipo con alguien que se desempeñe como Scrum Master a tiempo completo, permite que las actividades no se cumplan o que los impedimentos no se solucionen debidamente, lo que ocasiona retrasos en la producción dentro de las iteraciones.

En cuanto al desarrollo visual, encontramos dificultades al momento de utilizar modelos 3D acordes para el juego, originalmente los modelos disponibles no presentaban animaciones o el modelo en si no coincidía con el tema del juego. Luego de hacerle varios ajustes, se consiguió adaptar un modelo existente con sus animaciones con la temática del juego, al editar el modelo en sí y las texturas que traía incluido. Esta situación pudo ser manejada de mejor manera de haber contado con un animador 3D que trabajara en los modelos de los personajes y el resto de la escenografía del juego.

Además del problema para encontrar modelos adecuados, al desarrollar nuestros propios modelos, encontramos grandes dificultades al momento de cargar los modelos. El cargador de XNA resulta sumamente estricto en cuanto al formato de los modelos que carga correctamente, lo cual resulta sorprendente tomando en cuenta que es una herramienta para desarrollo de juegos independientes y enfocada a un público con poca experiencia en la industria de los videojuegos.

Trabajos futuros

Luego de haber realizado este trabajo de investigación, salta a la vista lo extenso que puede llegar a ser el tópico de desarrollo de videojuegos, por lo que, según lo elaborado y profundo que sean los futuros proyectos a atacar, así de extenso deben ser los equipos sugeridos de trabajo que se dediquen a estos proyectos.

Basándonos en el resultado final del proyecto, las posibles recomendaciones que se hacen para mejorar el juego en si son:

- Modelos 3D: El proyecto se enfocó al correcto funcionamiento del videojuego desde el punto de vista de programación. Sin embargo, una parte esencial para el éxito de un videojuego radica en el aspecto visual. Una mejora importante del producto final sería la incorporación al equipo de un modelador/animador 3D para crear nuevos modelos para los jugadores y mapas.
- Estructuras de datos: Para el manejo de colisiones se utilizó la técnica de Cajas envolventes/esferas envolventes. A pesar de esto, en escenarios que sean más complejos y saturados de objetos, el tiempo de ejecución se verá reducido considerablemente. Una propuesta para solventar esto es el uso adicional de otras estructuras, como es el caso de los Octrees, para generar una mejora en el procesamiento de colisiones.
- Interconectividad: El juego está basado en el framework XNA 4.0, por lo que las conexiones de red se hicieron a través de este con la plataforma Games for Windows© lo cual simplifica la gestión de usuarios y manejos de paquetes, con el inconveniente que solo permite crear escenarios multijugador en redes locales. Una sugerencia de mejora es el cambiar todo este sistema por uno de más bajo nivel, pero que permita conexiones no solo en redes locales,

sino a través de conexiones IP o a través de un servidor, lo cual permitiría un mayor número de jugadores participando en la interacción.

- Adaptación del juego al Xbox 360: Realizando algunas modificaciones al proyecto, es posible portar el videojuego a la consola Xbox360.
    - **Directivas condicionales:** Muchas instrucciones usadas al momento de desarrollar para PC son distintas a las funcionalidades o mecánicas a una consola. Un ejemplo seria el uso y manejo de los dispositivos de entrada, en una PC se tiene obligatoriamente acceso a un ratón y teclado, mientras que en la consola no tenemos acceso a dichos periféricos, únicamente el control. Por lo tanto, debe validarse este tipo de diferencias según la plataforma donde se esté ejecutando la aplicación. Para este fin XNA ofrece símbolos condicionales de compilación, los cual se interpretan al momento de compilar el código del juego final, y según la plataforma donde se esté compilando, utilizara el conjunto de instrucciones adecuado.

```
#if(WINDOWS)
    System.IO.StreamReader file =
    new System.IO.StreamReader (mapName + ".txt");
#if(XBOX)
    String fullPath=StorageContainer.TitleLocation + "LevelIndex.txt";
    System.IO.Stream stream=TitleContainer.OpenStream(mapName
                            + ".txt");
    System.IO.StreamReader file = new System.IO.StreamReader(stream);
#endif
```

Fig. 9.1.1 Esquema de trabajo con directivas condicionales.

    - **Manejo de archivos:** Cuando se utiliza el sistema de archivos de Windows, los archivos están organizados bajo una estructura jerárquica, la cual parte de un disco raíz (comúnmente C:) hasta llegar a la carpeta donde reside el juego. Ahí, podemos utilizar instrucciones de las librerías de XNA o incluso de C# para acceder a los archivos, incluso utilizar referencias locales. En un Xbox no existe esta configuración de sistema de archivos, por lo que cada juego tiene un espacio propio donde almacena los archivos con los que haya sido desarrollado. Para solventar esta dificultad, podemos combinar las *directivas condicionales* con un conjunto de instrucciones de la clase *TitleContainer* (como podemos ver en la figura 9.1.2) para la lectura y escritura de archivos.

```
private void AbrirArchivo(){
   try{
       System.IO.Stream flujo = TitleContainer.OpenStream
                                ("archive.txt");
        System.IO.StreamReader lectorFlujo = new
             System.IO.StreamReader(flujo);
        Console.WriteLine(lectorFlujo.ReadLine());
        Console.WriteLine("Tamaño de archivo: " + flujo.Length);
        flujo.Close();
    }
    catch (System.IO.FileNotFoundException)
    {
        // Hubo un error al buscar al archivo archive.txt
    }
}
```

Fig. 9.1.2 Ejemplo de lectura de archivo utilizando la clase TitleContainer.

- **Gramática del sistema de nombres de archivos:** los nombres de archivos, como norma, deben tener máximo 40 caracteres, y entre ellos solo pueden usarse letras, números y espacios en blanco, por lo cual muchos archivos de texturas y de sonidos ya se adaptaron a esta condición.

Apéndice A – Documento de Diseño de Juego (GDD)

PainTint

Paintball Táctico y Estratégico

Versión 1.0

Autores:

Andrade, José

Suarez, Francisco

Fecha: 01/12/12

Sección I – Resumen Del Juego

1.1. Concepto del juego

El Videojuego será un simulador del deporte Paintball, el cual incluye modos de juego populares en los juegos de disparos en primera persona.

1.2. Género

- Disparos en primera persona.
- Deportes
- Simulador (algunos modos de juego no simulan ningún aspecto del juego real, como por ejemplo el modo dominación. Los movimientos del personaje serán realistas).
- Estrategia.

1.3. Audiencia Objetivo

Adolescentes en adelante.

1.4. Resumen del flujo de juego

Al iniciar un juego, el usuario escoge si quiere empezar una partida o unirse a ella, luego de que esta creada la partida y todos los participantes estén dentro de ella, se da inicio. Dentro del juego el objetivo principal es alcanzar a los jugadores del equipo contrario con esferas de pintura. Si un jugador es alcanzado debe retirarse de la ronda actual y esperar que la ronda finalice. Luego de tres rondas, el equipo que obtenga más victorias es el equipo ganador. Luego de esto se finaliza el encuentro y debe crearse un nuevo juego para participar de nuevo.

1.5. Apariencia y sensación – ¿Cuál es el aspecto básico y la sensación del juego? ¿Cuál es el aspecto visual?

El aspecto básico y la sensación del juego son realistas.

Sección II – Jugabilidad y Mecánicas

2.1. Jugabilidad

2.1.1. Modos de juego

"PainTint" tendrá diversos modos, los cuales son:

- Eliminación en equipos: La misión de este modo es remover del campo a todos los miembros del equipo contrario.
- Capturar la bandera: El equipo que logre robar 5 veces la bandera del centro del escenario será el equipo ganador.
- Modo de entrenamiento: El cual permitirá conocer los mapas y practicar ciertas habilidades.

2.1.2. Flujo del juego

Al empezar la partida el jugador aparecerá en una zona específica del escenario. Luego de esto podrá desplazarse dentro de este para localizar a posibles oponentes. Una vez localizado un objetivo u oponente, el jugador tiene que apuntar al objetivo y utilizar el marcador para disparar las esferas de pintura. Si se alcanzó al oponente con las esferas, se incrementa el marcador de puntuación, en caso contrario el jugador sigue intentándolo hasta marcar el oponente o ser alcanzado por una esfera. En este último caso al jugador queda eliminado de esa ronda. Las rondas terminan según el modo de juego que se esté utilizando. En el caso de eliminación por equipos, la partida termina cuando todos los jugadores contrarios sean removidos del escenario. En el caso del modo de "Capturar la bandera" la ronda finaliza al momento de que la bandera sea llevada hasta la zona inicial de uno de los equipos o alguno de los equipos se quede sin jugadores.

Una vez que la partida llegue a su fin, se mostrarán por pantalla las estadísticas del juego.

2.2. Mecánicas del juego

2.2.1. Física

El ambiente físico del juego será similar al mundo real, existirá un valor para la gravedad y la escala con la que se manejan los objetos. Sin embargo la presencia del aire no se tomará en cuenta para los cálculos del movimiento de las esferas.

2.2.2. Colisiones

Las colisiones serán de 4 tipos: jugador-escenario, jugador-jugador, jugador-esfera y escenario-esfera. El sistema de colisiones será implementado con cajas envolventes.

2.2.3. Movimiento

Los personajes podrán desplazarse libremente sobre un escenario y será posible caminar, correr, agacharse y saltar, pero como en la vida real, si un jugador se encuentra corriendo, este no podrá disparar a la vez que realiza una carrera a máxima velocidad.

2.2.4. Acciones

2.2.4.0. Botones

A continuación se presenta una tabla con los botones (Figura A.2.2.4.1.1 y Figura A.2.2.4.1.2) que se utilizan en el juego. Además de esto en la Sección 4 de este apéndice se incluye un diagrama mostrando la distribución de los mismos.

Botón Control	Función
Mover Stick izquierdo	Desplazarse en el mapa/Seleccionar opciones del menú
Presionar Stick izquierdo	Saltar
Mover Stick derecho	Rotar personaje
A	Seleccionar opción menú
B	Retroceder en el menú
Y	Cambio de equipo en el Lobby
Left Button	Correr
Left Trigger	Agacharse
Right Button	Disparar
Right Trigger	Disparar
Start	Pausar el juego/avanzar en los menús
Back	Retroceder en los menús

Fig. A.2.2.4.1.1

Botón Teclado y ratón	Función
Botones W,S,A,D	Desplazarse en el mapa
Flechas	Desplazamiento en el mapa/desplazamiento en menús
Shift mientras se avanza	Correr en el mapa
Mover ratón	Rotar Cámara

Enter	Seleccionar opción menú
Barra Espaciadora	Saltar
Control	Agacharse
Botón izquierdo y derecho ratón	Disparar
Escape	Pausar el juego/Retroceder en los menús

Fig. A.2.2.4.1.2

2.2.4.1. Chat

Cada participante podrá enviar mensajes de chat a los otros participantes.

2.2.5. Multijugador

El juego estará orientado a manejar múltiples jugadores, los cuales se conectarán utilizando el protocolo TCP/IP. La topología utilizada será Cliente-Servidor, donde un computador se designa como servidor y espera solicitudes de entrada de otros clientes, luego el servidor decide no aceptar más solicitudes y empieza la partida. En ambas versiones se utilizará las funciones de XNA para conectarse en red por área local.

2.2.6. Enfrentamientos

Los combates serán a través del uso de marcadores de pintura.

2.3. Flujo de pantallas

2.3.1. Diagrama de flujo de pantallas

En este diagrama de actividades (el cual se observa en la Figura 10.1.3.1) se muestra las posibles interacciones con todas las pantallas y estados del juego.

2.3.2. Descripción de las pantallas

2.3.2.0. Pantalla inicial

Esta pantalla permitirá conocer el método de entrada utilizado por el jugador. Si este es un control, podremos conocer el índice del mismo.

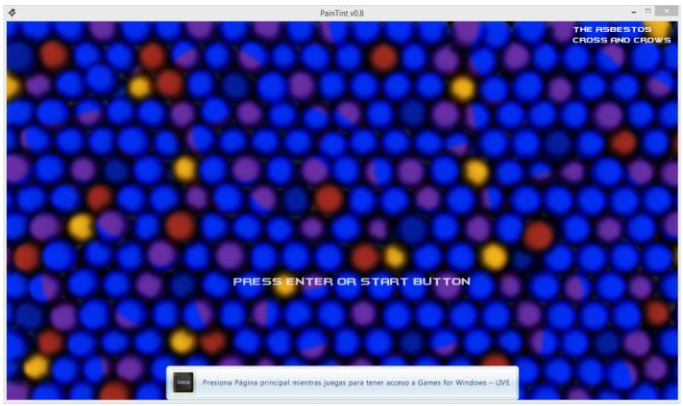


Fig. A.2.3.2.0 Pantalla inicial.

2.3.2.1. Menú Principal

Esta pantalla permitirá iniciar un nuevo juego, ver los créditos, crear un mapa, cambiar las opciones o salir de la aplicación.

Fig. A.2.3.2.1 Menú Principal.

2.3.2.2. Créditos

En esta pantalla se muestran los desarrolladores del videojuego.

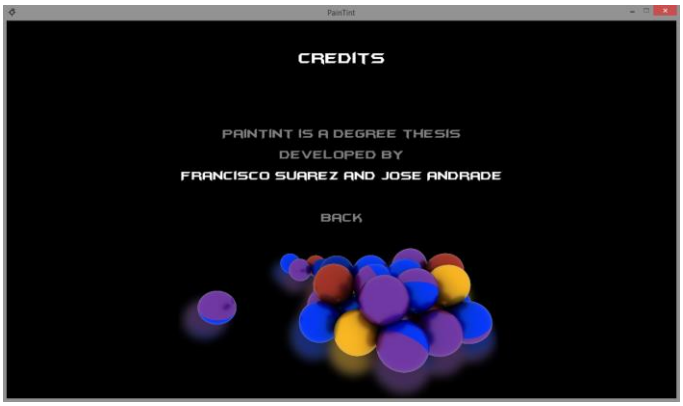


Fig. A.2.3.2.2 Créditos.

2.3.2.3. Crear mapa

En esta pantalla se puede crear un nuevo mapa, probarlo y guardarlo para su futura utilización.

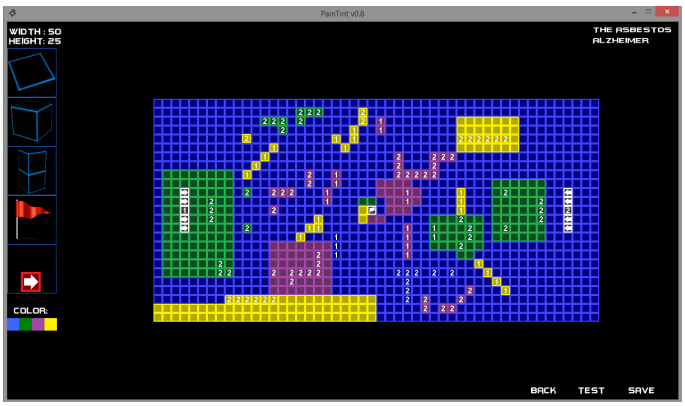


Fig. A.2.3.2.3 Crear mapa.

2.3.2.4. Opciones

Esta pantalla permite seleccionar ciertas opciones, entre ellas la selección del idioma.

Fig. A.2.3.2.4 Opciones.

2.3.2.1.1. Menú Nuevo juego

En esta pantalla se puede seleccionar entre crear una nueva partida o unirse a una partida anteriormente creada.

Fig. A.2.3.2.1.1 Menú nuevo juego.

2.3.2.1.2. Menú seleccionar partida

En esta pantalla se puede seleccionar a que partida unirse.

Fig. A.2.3.2.1.2 Menú seleccionar partida.

2.3.2.1.3. Sala de espera

En esta pantalla se puede seleccionar el mapa, el modo de juego y el equipo de cada jugador. También aquí los jugadores esperaran a que todos se unan a la partida para comenzar.

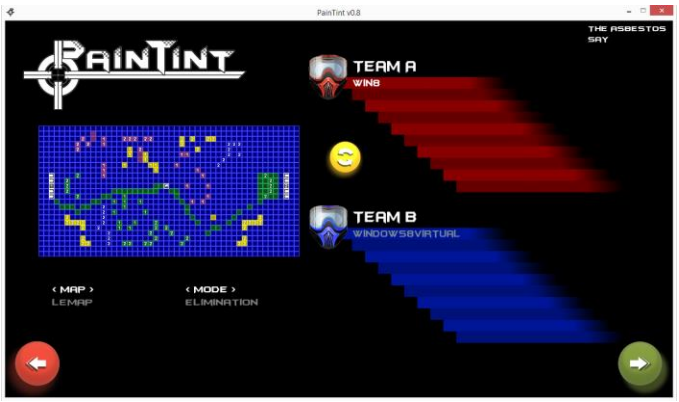


Fig. A.2.3.2.1.3 Sala de espera.

2.3.2.1.3.1 En juego

En esta pantalla se desarrolla la acción del videojuego. Se muestra el mapa de juego y los datos relevantes a la partida en curso.

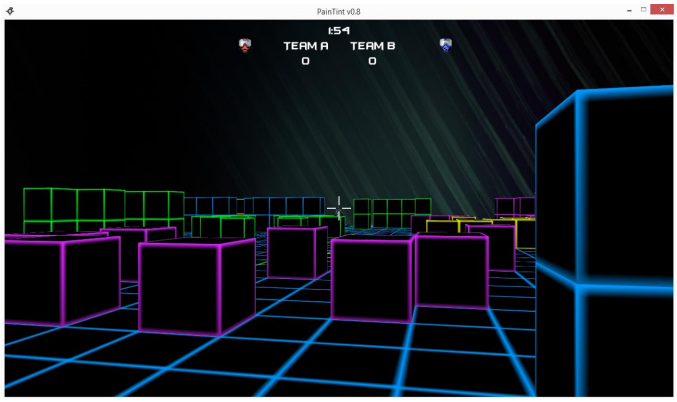


Fig. A.2.3.2.1.3.1 En juego.

2.3.2.1.3.2 Menú de pausa

Se muestra la opción de continuar la partida o salir.

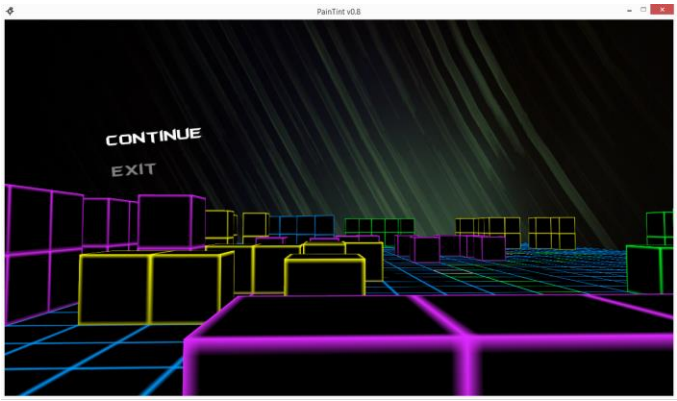


Fig. A.2.3.2.1.3.2 Menú de pausa.

Sección III – Interfaz

1.1. Sistemas visuales

1.1.1. Head-Up Display (HUD)

En el juego está presente un Head-Up Display, el cual es un conjunto de elementos que permiten visualizar información mientras la partida está en marcha, como la cantidad de jugadores activos, las rondas ganadas por cada equipo y el tiempo restante de la presente ronda.

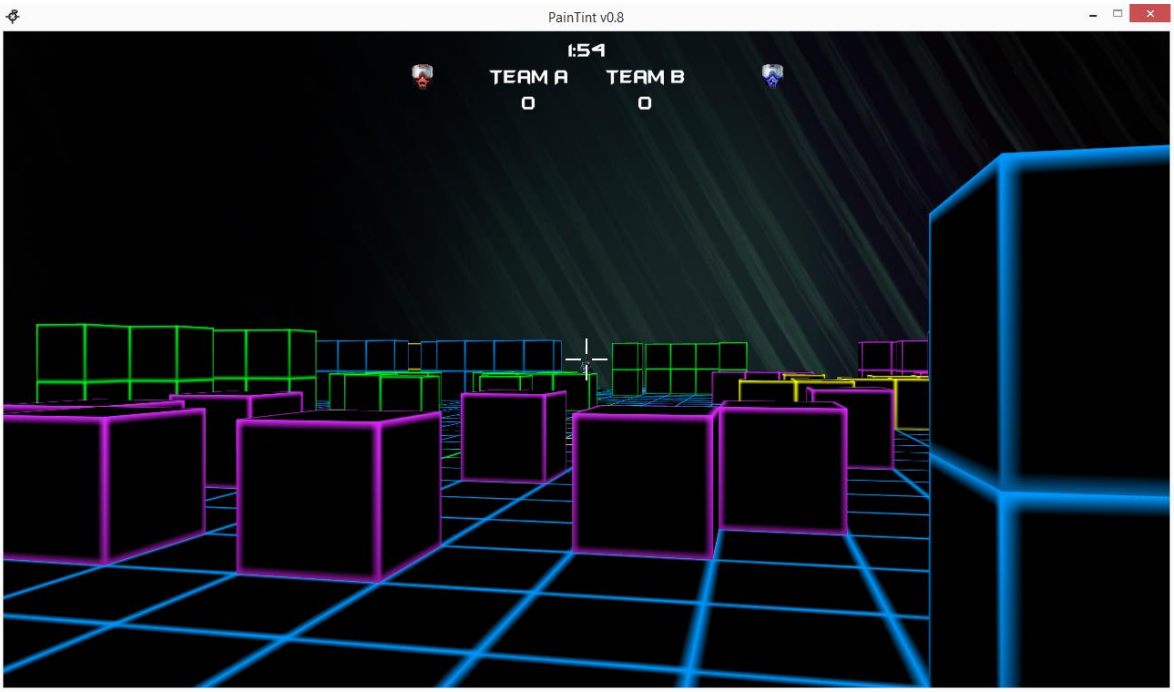


Fig.A.3.1.1.1 Head-Up Display de PainTint.

1.2. Sistemas de control

"PainTint" funciona con los siguientes sistemas de controles (mostrados en las Figuras A.3.2.1 y A.3.2.2)

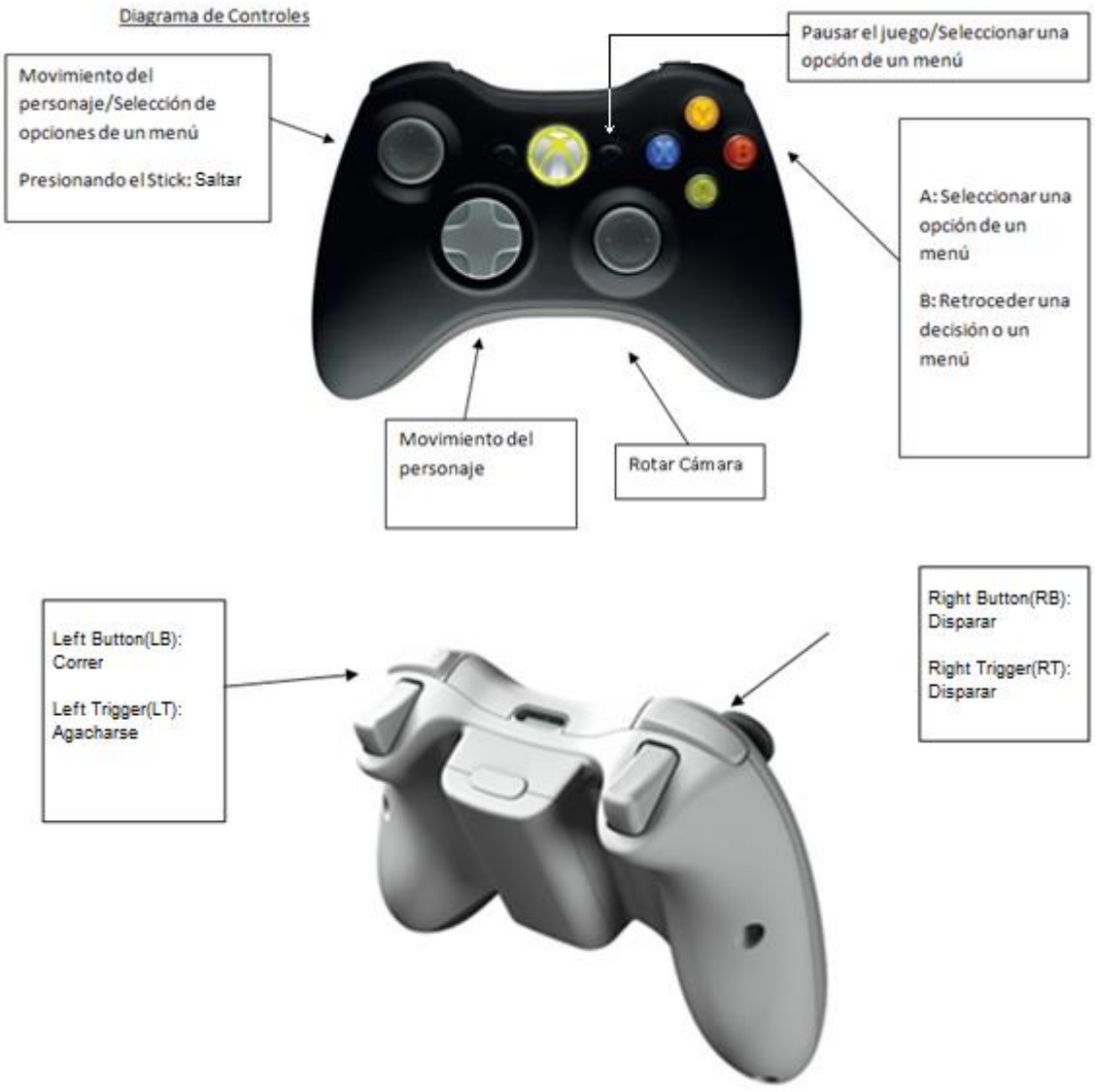


Fig. A.3.2.1 Descripción de funciones de los controles.

Fig. A.3.2.2 Descripción de funciones del teclado y ratón.

Sección IV – Especificaciones Técnicas

4.1 Hardware sugerido

Las características sugeridas que debe tener un computador para ejecutar "PainTint" son las siguientes:

- Procesador Intel Pentium Core 2 Duo de 2.6 Ghz
- 2 Gb de Memoria Ram
- Tarjeta de video NVidia GeForce 9800 GTX
- Sistema Operativo Windows XP o superior.
- .Net Framework 4.0
- Games for Windows (solo necesario bajo Windows 8)
- XNA 4.0 Redistributable

Además de esto, este juego puede ser ejecutado en cualquier consola Xbox 360, utilizando un gamepad, luego que el juego sea aprobado y publicado respectivamente en el Indie Game Store.

4.2 Software y Hardware de desarrollo

El juego se desarrolló con un computador con las siguientes características:

- Procesador Intel Pentium Dual Core de 2.7 Ghz
- 3 Gb de Memoria Ram
- Tarjeta de video NVidia GeForce 9800 GT

Además de esto se utilizaron las siguientes herramientas de software:

- Microsoft Windows XP, 7 y 8
- Microsoft Visual Studio 2010
- Microsoft Framework XNA 4.0
- .Net Framework 4
- Google Sketchup 8
- Autodesk 3D Studio Max 2012
- Blender 2.68
- Paint.Net
- NetBalancer
- Grabadora de Sonidos de Windows
- Free M4a to MP3 Converter
- VMWare Workstation 8

Sección V – Arte del Juego

Vista de un escenario creado con el editor de mapas

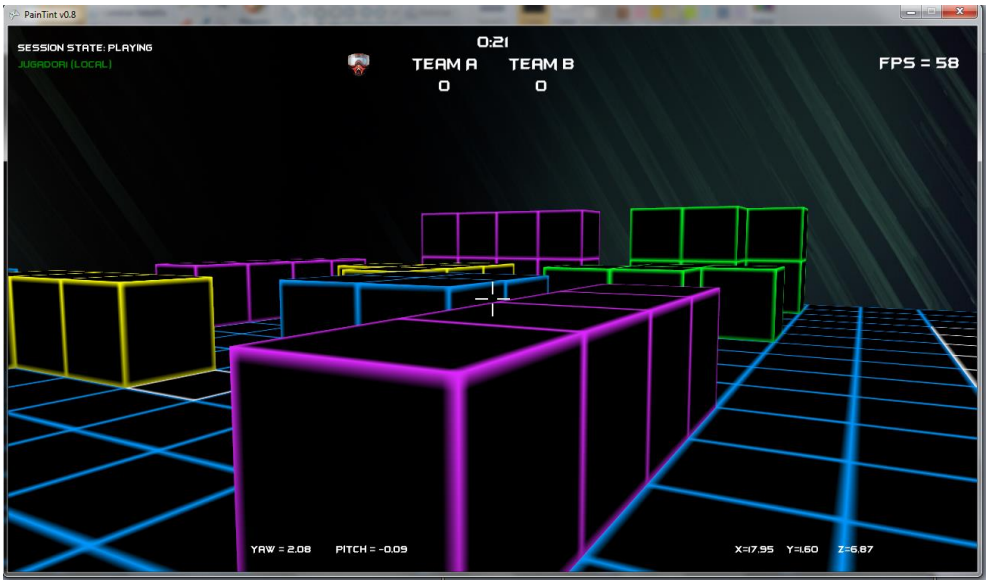


Fig. A.6.1 Vista previa "Eliminación Equipos".

Modelo utilizado para las pruebas del juego:

Fig. A.6.2 Modelo de pruebas.

Modelos modificados, se cambiaron las texturas, se eliminó el hacha y se escaló el modelo en el eje Y:

Fig. A.6.3 Modelo equipo azul.

Texturas modificadas:

Fig. A.6.4 Texturas del modelo.

Texturas de los mapas:

Fig. A.6.5 Texturas de los mapas.

Algunos elementos de los menús:

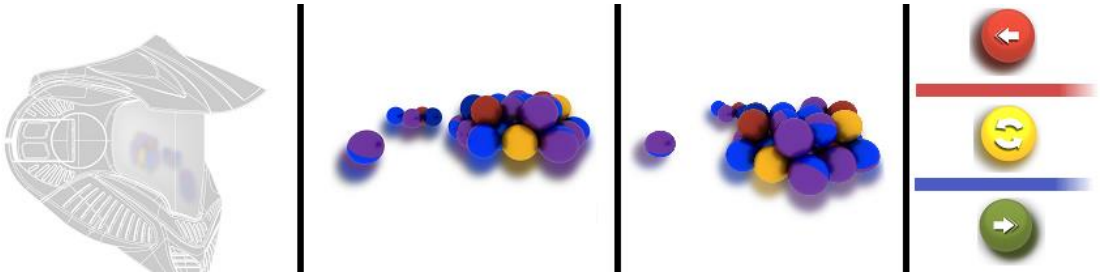


Fig. A.6.6 Elementos de los menús.

Logotipo del videojuego:

Fig. A.6.7 Logotipo.

Caretas:

Fig. A.6.8 Caretas de la sala de espera y el HUD.

Panel de puntuación:

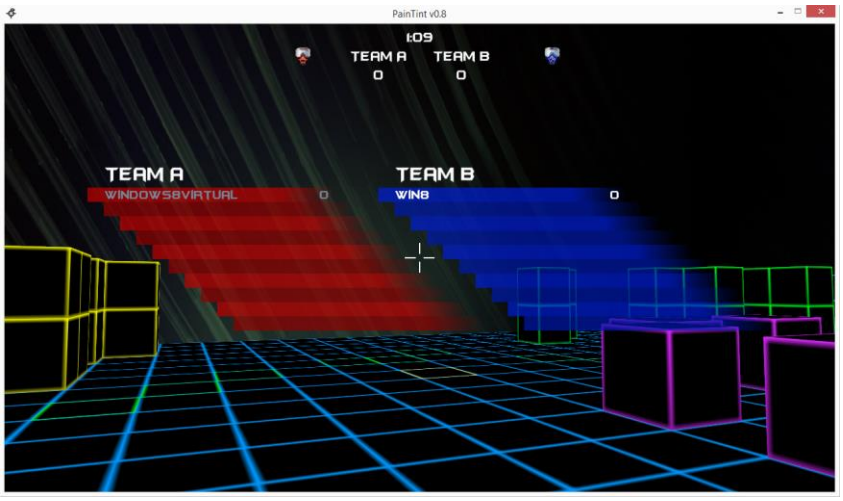


Fig. A.6.9 Tablas de puntuaciones de equipos.

Referencias

[1] Acevedo, D. (2009). *Diseño de una arquitectura para incorporar emociones en videojuegos*. Universidad Nacional Autónoma de México.

[2] Agile Manifesto (2001). *Manifiesto por el Desarrollo Ágil de Software*. Recuperado 21 de mayo, 2012, de http://agilemanifesto.org/iso/es/

[3] Amber, S. (2005). *The Elements of UML 2.0 Style*. Cambridge University Press

[4] Bethke, E. (2012). Game Development and Production. Wordware.

[5] Flynt, J. (2005), *Software Engineering for Game Developers*. Premier Press.

[6] Keith, C. (2010). *Agile Game Development with Scrum*. Addison Wesley.

[7] Microsoft (2009). *DirectX Graphics and Gaming*. Recuperado el 14 de junio, 2012, de http://msdn.microsoft.com/en-us/library/windows/desktop/ee663274(v=vs.85).aspx

[8] OMG Group (1997). *UML Resource Page.* Recuperado 9 de junio, 2012, de http://www.uml.org/

[9] Palacio, J. (2007). *Flexibilidad con Scrum*. [s.n].

[10] Schwaber,K. (2004). *Agile Project Management with Scrum* .Microsoft Press.

[11] Santos; Evangelista; Leal; Grootjans (2009), *Beginning XNA 3.0 Game Programming From Novice to Professional.* Apress.

[12] Evangelista, B. (2010). *XNAnimation Library*. Recuperado 21 de mayo, 2012, de http://xnanimation.codeplex.com/

[13] Rangel, A. (2010). Instrumento de evaluación de Videojuegos. Recuperado el 26 de octubre, 2012. https://docs.google.com/document/preview?id=10ztXF4TYPewwXU_26jgz0y1OLQfirac1_UWreyKMrog