*Universidad del Bío-Bío. Red de Bibliotecas - Chile*

**UNIVERSIDAD DEL BIO-BIO**

FACULTAD DE CIENCIAS EMPRESARIALES

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN Y

TECNOLOGÍA DE LA INFORMACIÓN

CAMPUS CHILLÁN

# "DISEÑO E IMPLEMENTACIÓN DE UN VIDEOJUEGO MULTIPLAYER ONLINE, UTILIZANDO EL MOTOR UNITY3D"

**JOSÉ ANDRÉS FUENTES RUBILAR**

**MEMORIA PARA OPTAR AL TÍTULO DE**

**INGENIERO CIVIL EN INFORMÁTICA**

**Chillán, Enero 2011**

Universidad del Bío-Bío. Red de Bibliotecas - Chile

**UNIVERSIDAD DEL BIO-BIO**

FACULTAD DE CIENCIAS EMPRESARIALES

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN Y

TECNOLOGÍA DE LA INFORMACIÓN

CAMPUS CHILLÁN

# "DISEÑO E IMPLEMENTACIÓN DE UN VIDEOJUEGO MULTIPLAYER ONLINE, UTILIZANDO EL MOTOR UNITY3D"

**JOSÉ ANDRÉS FUENTES RUBILAR**

| | |
|---|---|
| **PROFESOR GUIA** | **: SR. LUIS GAJARDO DÍAZ** |
| **PROFESOR INFORMANTE** | **: SR. MARIO GAETE PRADENAS** |
| **NOTA FINAL EXAMEN TITULO** | **: ___________________________** |

**MEMORIA PARA OPTAR AL TÍTULO DE**

**INGENIERO CIVIL EN INFORMÁTICA**

**Chillán, Enero 2011**

Universidad del Bío-Bío. Red de Bibliotecas - Chile


# *Agradecimientos*

Quisiera agradecer principalmente a Dios y a mi familia, por el gran apoyo brindado durante todos estos años. Además, también dar gracias a todos los que colaboraron en este proyecto, como son Rodrigo y Cristian de Jobbitgames, al profesor Miguel Pincheira por su aporte musical, y en especial al profesor Luis Gajardo, quien guió todo el desarrollo de éste trabajo. En fin, agradecer a mis amigos y compañeros de universidad, por su incondicional amistad y a todos los profesores que me educaron durante toda esta carrera.

***José Andrés Fuentes Rubilar***

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# *Dedicatoria*

*A mis padres, por su incansable lucha por darle lo mejor a sus hijos, a mi tía y en especial a mi abuelita que está en el cielo.*

***José Andrés Fuentes Rubilar***

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# RESUMEN

En la actualidad, los videojuegos tienen un rol muy importante en el rubro del entretenimiento a nivel mundial. De hecho, es un fenómeno social que traspasa todas las edades, sin distinción de géneros.

Sin embargo, en Chile, la industria de los videojuegos no ha alcanzado la madurez y envergadura de otros países del mundo, como es el caso de Estados Unidos, Japón y otros países desarrollados. Es más, existe un número muy reducido de empresas dedicadas al rubro, a pesar de las millonarias ventas que registra la industria alrededor del mundo, las que incluso han superado a las obtenidas por sectores tan importantes como lo es el cine. Una de las principales razones, de ésta situación, se debe al desconocimiento por parte de las empresas e instituciones de las tecnologías y procedimientos que son utilizados para producir un videojuego.

El presente proyecto aborda el tema del desarrollo de videojuegos utilizando tecnologías y herramientas especializadas, que permiten reducir tanto tiempo, así como también la complejidad que supone la producción de un videojuego. Todo esto, con el fin de mostrar la accesibilidad y el gran potencial que poseen herramientas como Unity3D, para las empresas emergentes de videojuegos como Jobbitgames, ubicada en la ciudad de Chillán.

En el desarrollo de éste trabajo, se utilizó la metodología de proceso de software iterativa incremental, con el fin de producir continuamente incrementos completamente funcionales. Además, se debió ajustar la etapa de diseño, producto de lo complejo y distinto que resulta el desarrollo de un videojuego, en comparación al software tradicional. Por otro lado, en la etapa de construcción del proyecto, se tuvo que implementar diferentes tipos de algoritmos, como por ejemplo, para dar inteligencia a los enemigos. Cabe mencionar, que todo esto fue posible gracias a la utilización e integración de diversas herramientas, como son, Unity3D, Google Sketchup, Maya, Blender, Gimp y Audacity, principalmente.

Una vez finalizado éste trabajo, se obtuvo como resultado un videojuego 3D respaldado con componentes y funcionalidades de nivel profesional, que entregan al usuario final una experiencia única.

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# ÍNDICE GENERAL

**CAPÍTULO I: ANTECEDENTES GENERALES....1**

1.1 INTRODUCCIÓN .... 2
1.2 DESCRIPCIÓN DEL PROBLEMA .... 2
1.3 OBJETIVO DEL PROYECTO .... 3
1.3.1 Objetivo General .... 3
1.3.2 Objetivos Específicos.... 3
1.4 METODOLOGÍA A UTILIZAR EN EL PROYECTO .... 4
1.5 DESCRIPCIÓN DE LA EMPRESA ASOCIADA AL PROYECTO .... 4

**CAPÍTULO II: EL MUNDO 3D .... 5**

2.1 SISTEMA DE COORDENADAS .... 6
2.2 TRANSFORMACIONES GEOMÉTRICAS.... 8
2.2.1 Translación .... 8
2.2.2 Rotación.... 9
2.2.3 Escalamiento .... 9
2.3 VECTORES.... 11
2.4 LA CÁMARA .... 12
2.5 RENDERIZACIÓN.... 14
2.6 POLÍGONOS, BORDES, VÉRTICES Y MALLAS .... 15
2.7 TEXTURAS, MATERIALES Y SHADERS .... 17
2.8 ILUMINACIÓN .... 18

**CAPÍTULO III: MOTORES DE VIDEOJUEGOS .... 19**

3.1 DEFINICIÓN .... 20
3.2 LA INDUSTRIA DE LOS VIDEOJUEGOS.... 21
3.3 HISTORIA.... 22
3.4 GÉNEROS Y ESTILOS DE VIDEOJUEGOS .... 25
3.5 PLATAFORMAS .... 27
3.6 ROLES DENTRO DEL DESARROLLO DE UN VIDEOJUEGO .... 28
3.6.1 Productor .... 28
3.6.2 Diseñador.... 28
3.6.3 Programador.... 29
3.6.4 Artistas .... 29
3.6.4.1 Visuales.... 29
3.6.4.2 Sonido.... 29

**CAPÍTULO IV: UNITY3D.... 30**

4.1 CARACTERÍSTICAS .... 31
4.2 LOS ASSETS .... 31

Universidad del Bío-Bío. Red de Bibliotecas - Chile

4.3 LAS ESCENAS ........ 32
4.4 LOS GAME OBJECTS ........ 32
4.5 LOS COMPONENTES ........ 33
4.6 SCRIPTS ........ 33
4.6.1 Operaciones Comunes ........ 34
4.6.2 Funciones Importantes ........ 36
4.7 LOS PREFABS ........ 36
4.8 LA INTERFAZ DE USUARIO ........ 37
4.8.1 La Ventana de Escena ........ 37
4.8.2 El Panel de Jerarquía ........ 38
4.8.3 El Inspector ........ 39
4.8.4 El Panel de Proyecto ........ 40
4.8.5 La Ventana de Juego ........ 41
4.9 HERRAMIENTAS EXTERNAS ........ 42
4.9.1 Modelado 3D ........ 42
4.9.2 Audio ........ 42

**CAPÍTULO V: DESARROLLO VIDEOJUEGO MULTIPLAYER ........ 43**

5.1 CONTROLANDO AL PERSONAJE ........ 44
5.2 UNA MIRADA 3D, LA CÁMARA ........ 50
5.3 LUZ, CÁMARA, ACCIÓN..., EL MUNDO 3D ........ 52
5.4 A TRAVÉS DE LA RED, EL MODO MULTIPLAYER ........ 55
5.5 ESTABLECER LA CONEXIÓN A LA BASE DE DATOS ........ 62
5.6 COMUNICACIÓN CHAT ........ 64
5.7 LA INTERFAZ DE USUARIO ........ 66
5.8 LA BATALLA FINAL, LOS ENEMIGOS ........ 71
5.9 PRUEBAS DE RENDIMIENTO ........ 78

**CAPÍTULO VI: CONCLUSIONES ........ 82**

6.1 CONCLUSIÓN ........ 83
6.2 RESULTADOS OBTENIDOS ........ 84
6.3 PROYECCIÓN A FUTURO ........ 84

**BIBLIOGRAFÍA ........ 85**

**ANEXOS ........ 87**

ANEXO A: HISTORIA DE ANCIENT GALE ........ 87
ANEXO B: JAVASCRIPT ........ 88
ANEXO C: FUNCIONES DE JAVASCRIPT ........ 90

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# ÍNDICE DE FIGURAS

*Figura 1, Sistema de coordenadas* .......... 6
*Figura 2, Sistema de mano izquierda y sistema de mano derecha* .......... 7
*Figura 3, "World Space" y "Local Space"* .......... 7
*Figura 4, Operaciones para realizar la translación* .......... 8
*Figura 5, Traslación* .......... 8
*Figura 6, Operaciones para realizar la rotación* .......... 9
*Figura 7, Rotación* .......... 9
*Figura 8, Operaciones para realizar el escalamiento* .......... 10
*Figura 9, Escalamiento* .......... 10
*Figura 10, Operaciones de transformación sucesivas sobre un objeto* .......... 11
*Figura 11, Suma de vectores* .......... 11
*Figura 12, Multiplicación de un escalar por un vector* .......... 12
*Figura 13, El Campo de visión FOV* .......... 12
*Figura 14, Punto de vista en primera y tercera persona* .......... 13
*Figura 15, Etapas del proceso de renderización* .......... 14
*Figura 16, Polígonos complejos representados en triángulos* .......... 15
*Figura 17, Polígonos Convexo y Cóncavo* .......... 16
*Figura 18, Esfera con distinta cantidad de polígonos* .......... 17
*Figura 19, Tipos de luces* .......... 18
*Figura 20, Esquema general de un motor de videojuego* .......... 20
*Figura 21, Gráfico de porcentajes de edad entre los jugadores* .......... 22
*Figura 22, Jerarquía de clases importantes de Unity3D* .......... 34
*Figura 23, Operaciones a través de la variable transform* .......... 35
*Figura 24, Interfaz de Usuario de Unity3D* .......... 37
*Figura 25, Herramientas de la ventana de escena* .......... 38
*Figura 26, Panel de Jerarquía* .......... 39
*Figura 27, El Inspector* .......... 40
*Figura 28, El Panel de Proyecto* .......... 41
*Figura 29, Modelo 3D personaje principal* .......... 44

Universidad del Bío-Bío. Red de Bibliotecas - Chile

*Figura 30, Estructura de componentes para el control del personaje* ............ 45
*Figura 31, Funciones para capturar la entrada del usuario*........................... 45
*Figura 32, Función para mover al personaje* ............................................. 46
*Figura 33, Habilidad de arrojar magia*...................................................... 47
*Figura 34, Obtener espada y abrir puerta*.................................................. 48
*Figura 35, La Vida del personaje*............................................................. 49
*Figura 36, Animaciones del personaje* ..................................................... 50
*Figura 37, Movimiento de la cámara*......................................................... 51
*Figura 38, Manipulación de la distancia y altura de la cámara*....................... 52
*Figura 39, Bosquejo del mapa de Ancient Gale* .......................................... 53
*Figura 40, Configuración de luces en Unity3D*............................................ 54
*Figura 41, Diseño del Mundo de Ancient Gale* ............................................ 55
*Figura 42, Modelo de interconexión entre clientes y servidores* ..................... 56
*Figura 43, El Script Conectar*.................................................................. 57
*Figura 44, Llamadas a Procedimientos Remotos, RPC* ................................. 60
*Figura 45, Jugador local y remoto* ........................................................... 61
*Figura 46, Tabla scores de la base de datos* .............................................. 62
*Figura 47, Funciones para enviar y solicitar los puntajes de los jugadores*...... 63
*Figura 48, El Script Chat*........................................................................ 65
*Figura 49, El Chat en Ancient Gale* .......................................................... 66
*Figura 50, Ventana de ingreso del nickname* ............................................. 67
*Figura 51, Ventana de selección del color del personaje*.............................. 68
*Figura 52, Salir del juego*....................................................................... 69
*Figura 53, La función OnGUI* .................................................................. 70
*Figura 54, El menú principal de Ancient Gale*............................................. 71
*Figura 55, Comportamiento de los Enemigos*............................................. 73
*Figura 56, El Script Dragon* .................................................................... 75
*Figura 57, el script EnemyStatus* ............................................................ 77
*Figura 58, Gráfico de rendimiento, caso jugador individual*........................... 79
*Figura 59, Gráfico de rendimiento, caso multiplayer*.................................... 80
*Figura 60, Definición de variables en JavaScript*......................................... 88
*Figura 61, Definición de una función en JavaScript*...................................... 89

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# Capítulo I

# ANTECEDENTES GENERALES

El siguiente capítulo tiene como fin introducir y comunicar al lector los objetivos del proyecto a desarrollar, así como también conocer sus principales características y funcionalidades.

En primer lugar, se comenzará con una introducción general, la cual presenta a grandes rasgos, las principales características del proyecto. Luego, se describe la problemática a abordar, en donde se menciona la situación actual y la alternativa de solución.

Además, se presentan los objetivos del proyecto, con el fin de comprender mejor lo que se quiere desarrollar. Por último, se describe la metodología de proceso de software seleccionada para construir el proyecto.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 1.1 Introducción

Hoy en día la industria de los videojuegos en Chile, aún no posee la madurez y envergadura de otros países del mundo, debido al gran riesgo que supone desarrollar un videojuego y al poco conocimiento por parte de las empresas e instituciones de herramientas que son comúnmente utilizadas para su creación, esto es, los motores de videojuego. Es más, son muy pocas las empresas chilenas dedicadas al rubro, encontrándose principalmente empresas como ACE Team, Atakama Labs, FrontDisk Games, AmnesiaGames, Jobbitgames, Baytex, Mazorca Studios, entre otras. De hecho, la mayoría se inclina por el desarrollo de juegos Flash, en lugar de videojuegos 3D como tal. Todo esto, a pesar de las millonarias ventas registradas por la industria, que incluso han superado a la del cine. Por ejemplo, en el año 2006 la industria de los videojuegos alcanzo ventas por sobre los $7.4 billones de dólares.

Es por esta razón que el proyecto a desarrollar, apuesta por abordar el tema de desarrollo de videojuegos utilizando herramientas especializadas para su diseño y creación, como lo es, por ejemplo, el motor Unity3D, el cual provee un conjunto de funcionalidades y componentes de software reutilizable.

Esto permitirá mostrar el gran potencial y atractivo que poseen estas herramientas en el desarrollo rápido de aplicaciones, ya que reducen costos, tiempo y complejidades en el proceso de construcción de un juego, todos estos factores son claves en la ya consolidada industria de los videojuegos.

## 1.2 Descripción del Problema

Como se mencionó anteriormente, la industria de los videojuegos posee una gran envergadura a nivel mundial, con una identidad propia y distinta a otros sectores industriales de desarrollo de software.

Sin embargo, en nuestro país muy poco se conoce del tema, esto se refleja principalmente en el reducido número de empresas que se dedican a desarrollar videojuegos. Las razones de éste poco interés por el desarrollo de este tipo de software pueden ser muchas, entre las cuales están: el desconocimiento de las herramientas y procedimientos que se utilizan en el desarrollo de un videojuego, a la gran cantidad de disciplinas que intervienen en su realización, dentro de los que se cuentan, programadores, diseñadores gráficos, sonidistas, etc. También, una de las razones, y quizás la principal, de por qué las empresas chilenas no ingresan a la

Universidad del Bío-Bío. Red de Bibliotecas - Chile



industria de los videojuegos es producto de la gran competencia con la que deben lidiar, esto es, las grandes empresas, que traspasan todas las fronteras y son reconocidas a nivel mundial en el desarrollo de videojuegos y las cuales llevan años en la competitiva industria. Un tema no menor, considerando que en algunas ocasiones el éxito de un videojuego radica en quién lo produjo.

Es por esto, que el proyecto a desarrollar intenta mostrar que existen herramientas accesibles, que no se alejan de la realidad de las empresas chilenas, y que ayudan de sobremanera en el desarrollo de un videojuego, tanto reduciendo costos, así como también tiempo.

## 1.3 Objetivo del Proyecto

A continuación se presentan los objetivos del proyecto, los cuales ayudarán a indicar sus principales características y que además proporcionarán una guía para el desarrollo del producto final.

### 1.3.1 Objetivo General

El objetivo general del proyecto a desarrollar es diseñar e implementar un videojuego multiplayer online, que utilice la arquitectura cliente/servidor y una conexión a una base de datos, mediante la utilización del motor Unity3D.

### 1.3.2 Objetivos Específicos

A continuación se presentan los objetivos específicos del proyecto a desarrollar, los cuales muestran más en detalle las características principales de éste:

- Desarrollar la base del videojuego, esto es, el mundo (3D) virtual en donde se desenvolverán los jugadores, el control del personaje principal, la cámara que mostrará al personaje y el modo multiplayer que permitirá interactuar a los jugadores a través de una red.
- Establecer una conexión a una base de datos, donde se guardarán los datos (nickname) y puntajes de los jugadores, que permita establecer un ranking.
- Implementar un sistema de comunicación tipo chat, que permita a los jugadores enviarse mensajes entre sí.
- Diseñar una interfaz de usuario, que permita a los jugadores tanto establecer o conectarse a un servidor, para comenzar una partida de juego, así como

Universidad del Bío-Bío. Red de Bibliotecas - Chile



también para mostrar el status del jugador, por ejemplo, su puntaje, su nivel de salud, etc.

- Diseñar e implementar un sistema de batallas tipo juego de rol, lo que implica la creación de enemigos dentro del juego, los cuales harán frente a los jugadores.

## 1.4 Metodología a Utilizar en el Proyecto

El modelo de proceso de software a utilizar en la realización de este proyecto es el iterativo incremental, el cual permitirá generar incrementos de software operacionales, los cuales entregarán una idea del avance y cumplimiento de los objetivos del proyecto.

Cabe mencionar que esta metodología solo se tomará como una referencia, debido a que el proceso de desarrollo de un videojuego es muy distinto al desarrollo tradicional de software, en donde el proceso de desarrollo es más flexible y un poco más informal. Sin embargo, igualmente se utilizarán aspectos de la metodología, con el objetivo de promover la ingeniería del software.

## 1.5 Descripción de la Empresa Asociada al Proyecto

En la construcción de éste proyecto, se contará con el apoyo de la empresa Jobbitgames, la cual se dedica al desarrollo de videojuegos en el campo de la educación y el entretenimiento, tanto para las plataformas de PC, así como también para las emergentes plataformas móviles de IPhone y IPad. Dentro de sus principales trabajos, está el videojuego denominado Uko en el Laboratorio Mágico, el cual tiene como objetivo enseñar las primeras palabras en inglés a los niños pequeños.

Ésta empresa se encuentra situada en la ciudad de Chillán, la cual fue fundada por Rodrigo Suárez y Cristian de la Fuente, ambos pertenecientes al área de diseño gráfico.

Ahora bien, en cuanto al aporte de la empresa, se destaca principalmente la animación de los modelos 3D y todo lo relacionado con las gráficas visuales del videojuego, como por ejemplo el logo, íconos de estado, textos, etc.

Cabe destacar que la colaboración y comunicación constante con la empresa Jobbitgames, será una instancia propicia para poner en práctica la integración con otras áreas de especialidad.

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# Capítulo II

# EL MUNDO 3D

El mundo de los videojuegos intenta recrear el espacio donde nos desenvolvemos, esto es, un mundo que posee tres dimensiones, un alto, un ancho y una profundidad.

Sin embargo, un videojuego, es representado dentro de una pantalla que posee solo dos de las dimensiones señaladas anteriormente (ancho y alto), por lo tanto, es necesario contar una serie de técnicas para simular la tercera dimensión faltante, esto es, la profundidad.

Es por esto, que el capítulo siguiente presentará las técnicas utilizadas para crear y simular un mundo 3D parecido al nuestro.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 2.1 Sistema de Coordenadas

Un concepto crucial dentro de cualquier entorno 3D es el sistema de coordenadas, el cual está compuesto por tres variables que comúnmente se denominan, X, Y, Z, las cuales sirven para representar cada una de las tres dimensiones: ancho, alto y profundidad, y que nos ayudan a referenciarnos dentro de un espacio 3D [3]. Es por esto que es muy importante conocer cuál es el sistema de coordenadas con que trabaja un entorno 3D particular.

Por lo general, cada dirección o dimensión dentro de un sistema de coordenadas está representada por un eje. El ancho es representado comúnmente por el eje X, el alto por el eje Y, y la profundidad por el eje Z. En la siguiente figura se puede apreciar de mejor manera como están dispuestos cada uno de los ejes con respecto a la visión que posee una persona del espacio 3D [3]:

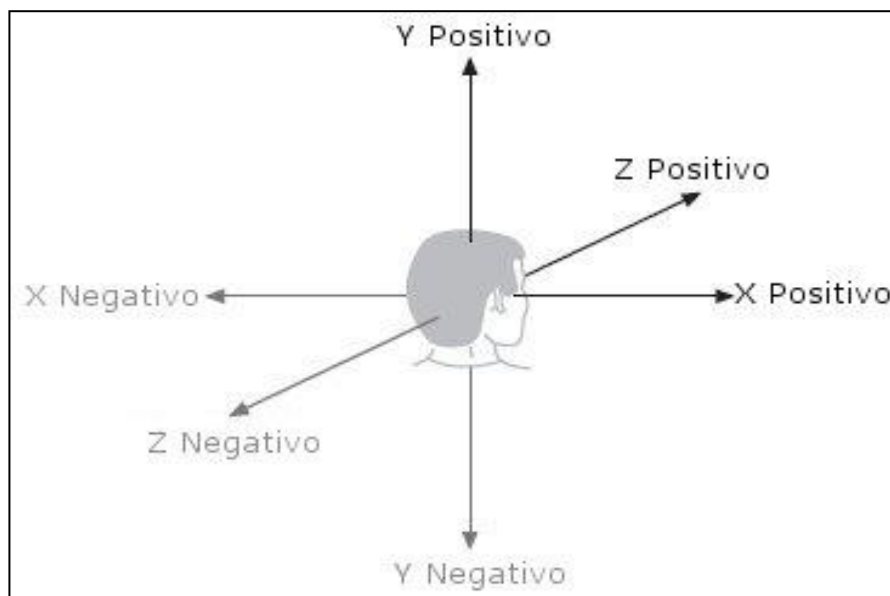



*Figura 1, Sistema de coordenadas*

Se puede ver en la Figura 1, que cada uno de los ejes coordenados posee un lado positivo y un lado negativo, y que también existe un punto de intersección de todos ellos, el cual se conoce como el punto cero u origen (0, 0, 0).

Ahora bien, existen diversas formas de representar las diferentes dimensiones, dentro de las que se pueden mencionar por ejemplo, el sistema de mano izquierda y el de mano derecha. Ambos, poseen diferencias sutiles, pero que hay que considerar. La Figura 2 muestra ambos tipos de sistemas y en donde se pueden apreciar sus principales distinciones [3]:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



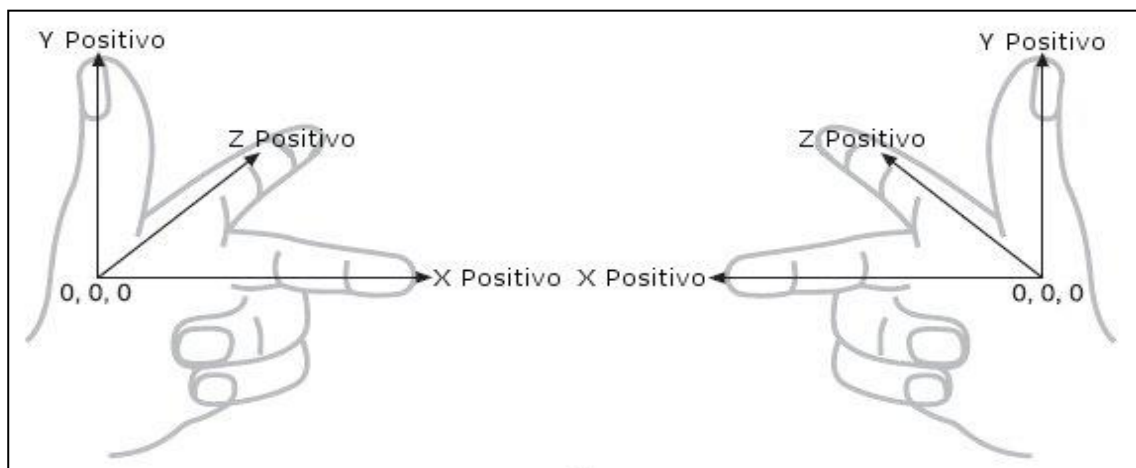



*Figura 2, Sistema de mano izquierda y sistema de mano derecha*

Cabe mencionar, que el motor Unity3D, utiliza el sistema de mano izquierda como sistema de referencia dentro de su espacio o mundo 3D.

Otra de las consideraciones que se deben tener en cuenta al trabajar en un entorno 3D, es que existen dos tipos de espacios dentro del entorno, uno es el denominado "Space World", el cual consiste en el espacio global, que es común para todos los objetos dentro del espacio 3D y el cual posee un punto de origen único. El otro, es el denominado "Local Space", el cual pertenece a un objeto particular dentro del espacio 3D, y que posee un punto cero u origen propio, que es comúnmente el centro geométrico del objeto [4]. Así, cada objeto puede ser manipulado con respecto al "Space World" o a su "Local Space", según se requiera. La Figura 3 ilustra la definición de "World Space" y "Local Space":

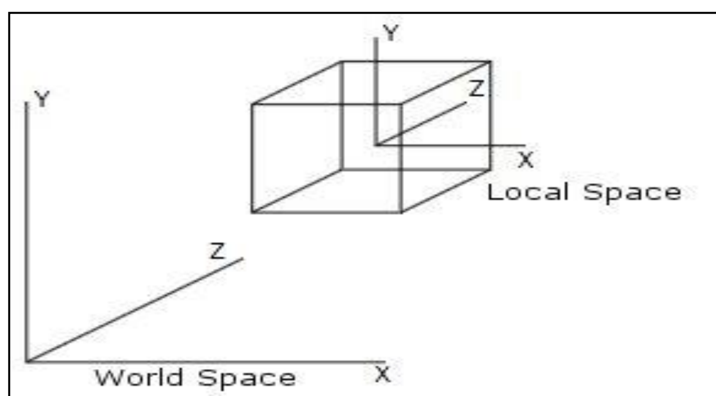



*Figura 3, "World Space" y "Local Space"*

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 2.2 Transformaciones Geométricas

En cada entorno 3D, y más específicamente en Unity3D, el que nos interesa, todo objeto en el espacio posee un concepto denominado transformación geométrica. Éste concepto, indica que todo objeto posee una posición dentro del espacio 3D, una rotación y una escala, cada uno de ellos definido por las variables X, Y, Z. Ahora bien, cada una de estas propiedades (posición, rotación y escala), se traducen en operaciones matemáticas que operan sobre los objetos y que son esenciales a la hora de comenzar a trabajar en un espacio 3D. A continuación se describirán cada una de estas operaciones.

### 2.2.1 Translación

Es la más simple de las transformaciones que se puede realizar a un objeto y la cual consiste en mover o trasladar de un punto a otro un objeto dentro de un espacio 3D [3]. Esta operación se realiza mediante adición de un vector a la posición del objeto. Por ejemplo, se tiene un punto P(0, 0, 0), el cual corresponde a la posición actual del objeto. Ahora, si se aplica la operación de traslación al objeto y éste se quiere trasladar a un punto distinto, se añade las unidades necesarias a cada componente de la posición del objeto, esto es, se quiere trasladar el objeto al punto (1, 3, 2), entonces el nuevo punto o posición del objeto queda como sigue, P`(1, 3, 2). La Figura 4 muestra como sería mover un punto P(X, Y, Z) a otro P`(X`, Y`, Z`) una distancia d en cada eje coordenado:

$$X` = X + d_x \qquad Y` = Y + d_y \qquad Z` = Z + d_z$$

*Figura 4, Operaciones para realizar la translación*

La Figura 5 ilustra cómo sería la operación de translación aplicada a un objeto dentro de un espacio 3D [3]:

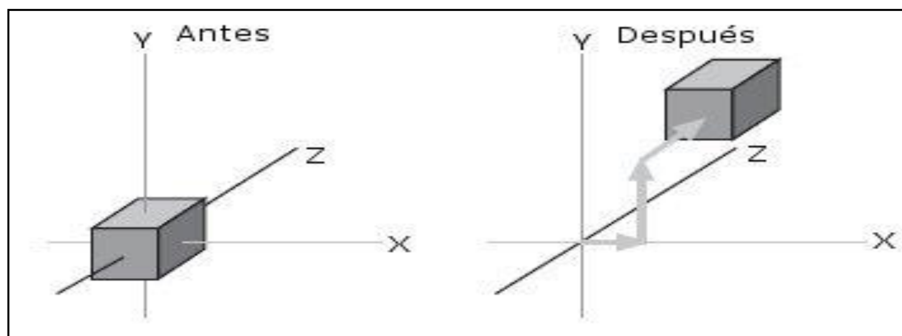



*Figura 5, Traslación*

Universidad del Bío-Bío. Red de Bibliotecas - Chile



### 2.2.2 Rotación

La operación de rotación es una de las más complejas dentro de las que se pueden realizar a un objeto en un espacio 3D. Ésta consiste en rotar alrededor de un eje coordenado un objeto en un ángulo dado [14]. La Figura 6 muestra las fórmulas matemáticas para rotar un punto P(X, Y, Z) a otro P`(X`, Y`, Z`) en un ángulo θ, para cada uno de los ejes coordenados:

En el eje X:
$X' = X$; $Y' = Y * \cos\theta - Z * \text{sen}\,\theta$; $Z' = Y * \text{sen}\,\theta + Z * \cos\theta$

En el eje Y:
$X' = X * \cos\theta + Z * \text{sen}\,\theta$; $Y' = Y$; $Z' = -X * \text{sen}\,\theta + Z * \cos\theta$

En el eje Z:
$X' = X * \cos\theta - Y * \text{sen}\,\theta$; $Y' = X * \text{sen}\,\theta + Y * \cos\theta$; $Z' = Z$

*Figura 6, Operaciones para realizar la rotación*

La Figura 7 muestra más gráficamente cómo sería rotar un objeto alrededor de su eje Z [3]:

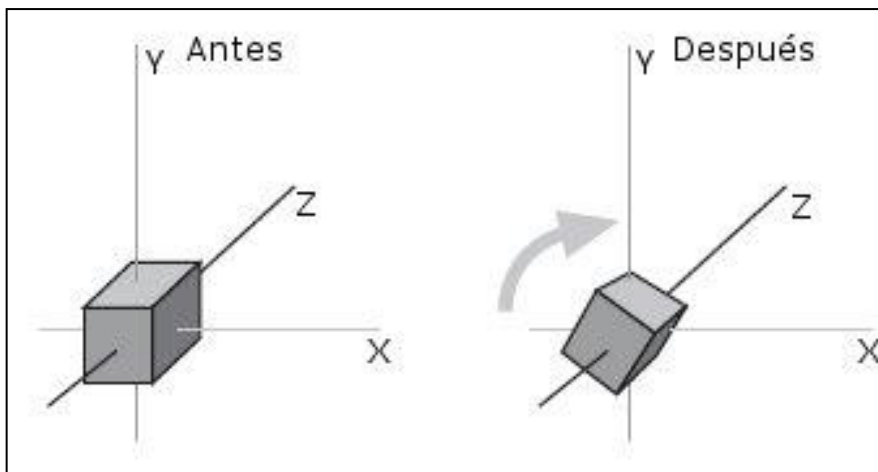



*Figura 7, Rotación*

### 2.2.3 Escalamiento

La última de las operaciones geométricas que actúan sobre un objeto es la de escalamiento. Consiste en escalar una coordenada por un factor constante, lo que se traduce en aumentar el tamaño de un objeto (si el factor es mayor que 1) o disminuir su tamaño (si el factor es menor que 1 y mayor que 0) [3]. Si el factor de escala es igual a 1, el objeto no cambia. Cabe mencionar que un objeto se puede escalar de forma distinta para cada componente X, Y, Z, por lo que se traduce en un escalamiento no uniforme, produciéndose así una deformación del objeto. La Figura 8 muestra como

Universidad del Bío-Bío. Red de Bibliotecas - Chile



sería escalar un punto P(X, Y, Z) a otro P`(X`, Y`, Z`) por un factor s, en cada uno de los ejes coordenados:

$$X` = X * s_x \qquad Y` = Y * s_y \qquad Z` = Z * s_z$$

*Figura 8, Operaciones para realizar el escalamiento*

La Figura 9 ilustra cómo sería la operación de escalamiento aplicada a un objeto dentro de un espacio 3D [3]:

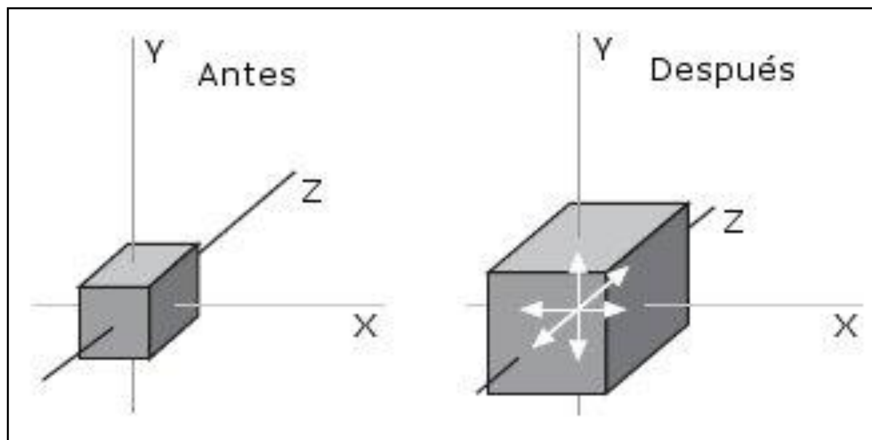



*Figura 9, Escalamiento*

Es importante destacar que todas estas operaciones de transformación, sobre los objetos dentro de un espacio 3D, se realizan sobre cada uno de los puntos o vértices que conforman el objeto. Además, cada una de estas operaciones se lleva a cabo a través de una matriz de coordenadas homogéneas, en donde se van acumulando cada una de las transformaciones realizadas a un objeto, ya sea una traslación, una rotación o un escalamiento.

Cabe mencionar, además, el beneficio que entrega el formato de coordenadas homogéneas a la hora de dibujar una escena 3D, ya que permite aplicar las diferentes transformaciones geométricas de forma combinada, reduciendo así el número de operaciones matemáticas, y por ende mejorar el rendimiento de la aplicación 3D, debido a que cada vértice de un objeto se multiplica una sola vez por ésta matriz, en lugar de realizar un cálculo por cada transformación geométrica que se quiera realizar. La Figura 10 muestra una sucesión de operaciones de transformación sobre un objeto dentro de un espacio 3D [3]:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



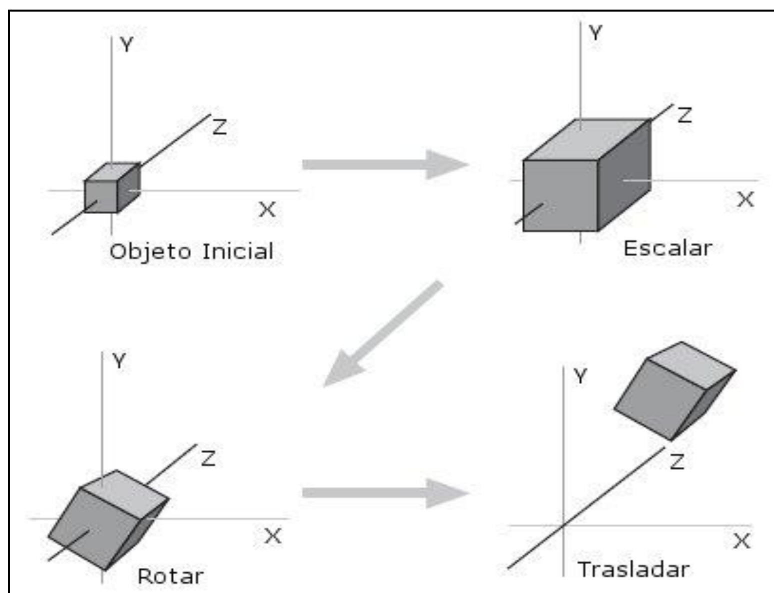



*Figura 10, Operaciones de transformación sucesivas sobre un objeto*

## 2.3 Vectores

Formalmente, un Vector se define como "todo segmento de recta dirigido en el espacio" [2], el cual posee una magnitud, una dirección y el cual se denota, en un contexto 3D, como una tupla de tres componentes (X, Y, Z). La magnitud de un vector es la longitud o tamaño del vector, mientras que la dirección viene dada por la orientación en el espacio de la recta que lo contiene [2]. Los vectores son muy importantes en cualquier entorno 3D, ya que pueden ser utilizados para calcular distancias entre objetos dentro del espacio, para calcular ángulos relativos y también la dirección de los objetos, entre otras cosas [4].

Ahora bien, matemáticamente un vector puede ser sumado con otro, para así formar un nuevo vector. El resultado de esta suma, se traduce en un nuevo vector, el cual posee una dirección o magnitud distinta. La Figura 11 muestra la operatoria para sumar el vector U con el vector V [1]:

$$U + V = (U_x + V_x, U_y + V_y, U_z + V_z)$$

*Figura 11, Suma de vectores*

También es posible multiplicar un vector por una constante o escalar, el cual cambia su magnitud sin modificar su dirección. La Figura 12 muestra la operación matemática para multiplicar un escalar s por un vector V [1]:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



$$s * V = (s * V_x, s * V_y, s * V_z)$$

*Figura 12, Multiplicación de un escalar por un vector*

## 2.4 La Cámara

La cámara es un concepto esencial dentro de cualquier videojuego, ya que actúa como la visión que tiene el jugador, capturando y desplegando por pantalla los elementos y objetos del mundo 3D. Ésta puede ser posicionada en cualquier punto dentro del espacio 3D, pero por lo general tiene como objetivo acompañar al personaje principal dentro del juego. Por lo tanto, toda cámara dentro de un entorno 3D posee una posición, orientación y un campo de visión. El campo de visión (FOV) es un término asociado a la cámara, el cual corresponde al ángulo de cobertura o de visión que posee ésta [7]. Como ejemplo del campo de visión, se puede mencionar el efecto denominado Zoom, en donde un incremento del FOV (ángulo de visión), se puede apreciar como si los objetos se alejaran de la cámara, mientras que un descenso del FOV hace que los objetos a distancia se acerquen a la cámara. La Figura 13 ilustra el FOV, desde el punto de vista de la cámara:

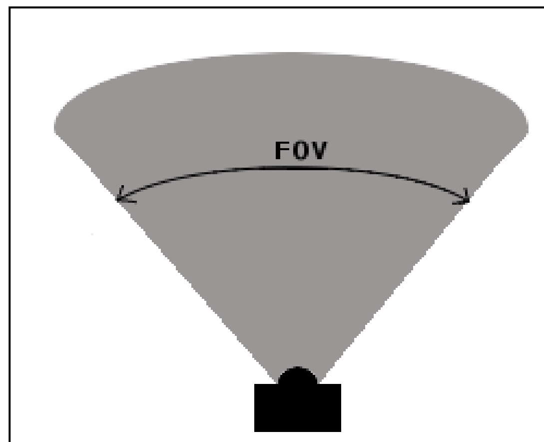



*Figura 13, El Campo de visión FOV*

Otro concepto que se puede desprender de la cámara es el de punto de vista (POV). Éste concepto consiste básicamente en la posición que posee la cámara con respecto al personaje principal de un videojuego. Principalmente, existen dos puntos de vistas, el de primera persona y el de tercera persona. El de primera persona consiste en que la cámara esta posicionada en los "ojos" o "cabeza" del personaje principal, simulando como si el jugador estuviera dentro del juego mirando desde sus ojos el entorno. En cambio, el punto de vista de tercera persona consiste en posicionar la cámara a una distancia tal, que se puede apreciar tanto al personaje principal, como también el entorno, a diferencia del punto de vista de primera persona, en el cual no

Universidad del Bío-Bío. Red de Bibliotecas - Chile



se ve el "cuerpo" del personaje [7]. La Figura 14, muestra gráficamente la diferencia entre el punto de vista de primera y el de tercera persona:

*Figura 14, Punto de vista en primera y tercera persona*

Como se aprecia en la Figura 14, en el punto de vista de primera persona, lo único que se aprecia del personaje principal es el arma que lleva en su mano, siendo esto muy común en los juegos con este tipo de vista. A diferencia del tipo de tercera persona, donde el personaje es visualizado completamente por la cámara.

Otro de los conceptos inherentes a la cámara es el de proyección. La proyección corresponde a una transformación de la vista de la cámara, donde se pueden encontrar, principalmente, dos tipos: proyección paralela y perspectiva. En cada uno de estos dos casos, un plano conocido como plano de vista, es posicionado dentro del mundo virtual [14]. Ahora, la proyección paralela proyecta puntos 3D en el plano de vista, a través líneas paralelas en una dirección fija. Cuando las líneas de la proyección son perpendiculares al plano de vista, la proyección paralela se llama ortográfica [14]. El volumen de vista proporcionado por la proyección ortográfica es equivalente al de un paralelogramo. La principal característica de la proyección ortográfica es que no se percibe la lejanía entre objetos que están dentro de una escena 3D, todos se ven a la misma escala, aunque estén a una distancia notable de la cámara. Por otro lado, se tiene la proyección en perspectiva, en la cual todas las líneas de proyección convergen en el denominado punto de vista (posición del mirador), formando un volumen de vista en forma de pirámide. Este tipo de proyección es más realista que las paralelas, ya que es posible apreciar la lejanía entre objetos, es decir, objetos más cercanos al punto de vista, son más grandes que los objetos a distancia, simulando así el modo de proyección del ojo humano [14].

Universidad del Bío-Bío. Red de Bibliotecas - Chile



En la mayoría de los videojuegos modernos se puede apreciar varios efectos aplicados a la cámara, dentro de los que se encuentran efectos de luz, "Motion Blurs" (distorsión de la vista, simulando un movimiento brusco de la cámara), "lens flares" (efecto que simula la refracción de la luz en un lente de cámara, por ejemplo al enfocar el sol), etc. Además, es muy común encontrar múltiples cámaras dentro de un mismo juego, permitiendo al jugador visualizar diferentes lugares al mismo tiempo.

## 2.5 Renderización

Como se mencionó en el punto anterior, la cámara captura y muestra al jugador los elementos y objetos del mundo 3D, mediante la realización de un proceso denominado renderización (o rendering, en inglés). Básicamente, la renderización consiste en el proceso de convertir los modelos matemáticos 3D en una imagen 2D que se muestra por pantalla [3]. Sin duda, este proceso es muy complejo, ya que se deben considerar múltiples factores a la hora de renderizar un objeto, como por ejemplo: su geometría, su posición, las transformaciones geométricas realizadas a éste, sus texturas, la luz, etc. [14]. Todos éstos influyen en la apariencia del objeto cuando es mostrado por pantalla. A continuación, en la Figura 15, se muestran las tres etapas que deben realizarse para llevar a cabo el proceso de renderización, el cual es conocido como "Rendering Pipeline" [8]:

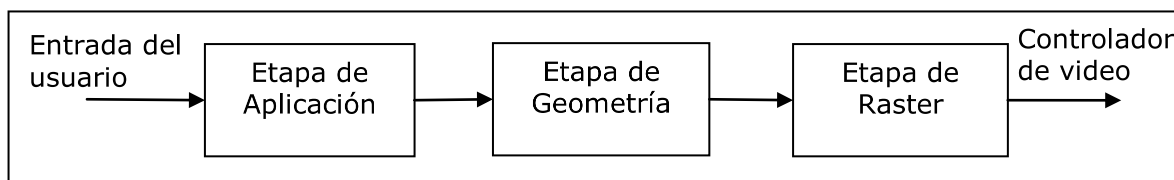



*Figura 15, Etapas del proceso de renderización*

Como se puede apreciar en la Figura 15, la primera etapa del proceso de renderización, consiste en la etapa de aplicación, la cual se encarga de leer la entrada del usuario y la geometría del mundo virtual. Ésta etapa, entrega como salida primitivas geométricas (vértices, polígonos y mallas), las cuales sirven como entrada a la etapa de geometría. En la etapa de geometría, se llevan a cabo una serie de operaciones, dentro de las que se encuentran, la realización de las transformaciones geométricas de los modelos, el cálculo de luz de la escena 3D y el tipo de proyección de la cámara (perspectiva o paralela). También, en ésta etapa, se realiza la operación denominada "Clipping", la cual permite establecer qué elementos serán  renderizados o mostrados por pantalla, en base a sí éstos están dentro del campo de visión de la cámara. Para ello, se debe realizar una transformación de coordenadas del mundo

Universidad del Bío-Bío. Red de Bibliotecas - Chile



virtual (3D), a coordenadas de pantalla (2D), obteniendo así una especie de mapeo de coordenadas. La etapa de geometría, produce como salida, polígonos en dos dimensiones, los cuales son tomados como entrada por la etapa de raster. En resumen, en ésta última etapa, se lleva a cabo la conversión de la información obtenida en la etapa de geometría, en forma de pixel, la cual es traspasada al controlador de video, para que la dibuje por pantalla. Aquí, también se realizan las operaciones necesarias para establecer los colores y texturas que poseerán los pixeles, correspondiéndose así, con la realidad del mundo 3D.

## 2.6 Polígonos, Bordes, Vértices y Mallas

Toda figura 3D es construida en base a la interconexión de figuras 2D conocidas como polígonos [4]. Cada uno de estos polígonos se forma mediante puntos o vértices en el espacio, los cuales se unen por medio de líneas rectas, también conocidos como bordes. Al unir tres vértices, se forma el polígono cerrado (una cara) más simple de todos, el triángulo. En la actualidad, el hardware de las tarjetas gráficas es diseñado para manipular y desplegar por pantalla, millones y millones de triángulos por segundo [3]. Es por esta razón, que Unity3D, al momento de importar un modelo 3D, convierte todos los polígonos en triángulos, con el objetivo de aprovechar al máximo las capacidades de cálculo de las tarjetas gráficas. Esto es posible, gracias a que cualquier tipo de polígono, no importando que tan complejo sea, puede ser representado por una colección de triángulos. La Figura 16 ilustra cómo se convierten polígonos complejos en un conjunto de triángulos:

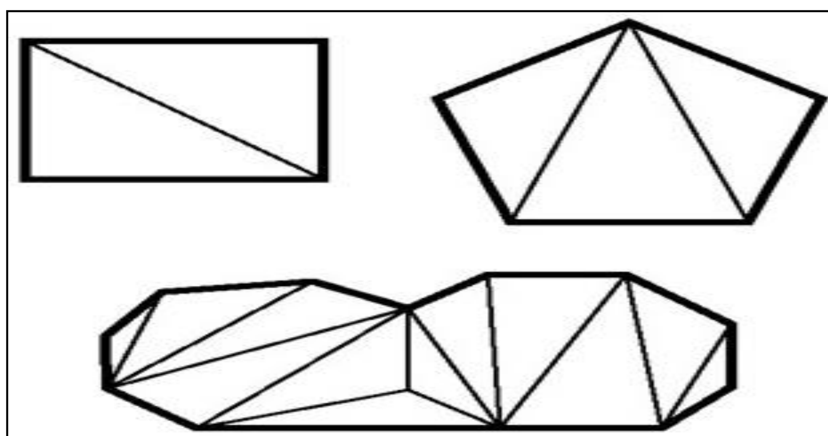


*Figura 16, Polígonos complejos representados en triángulos*

Ahora bien, existen 2 tipos de polígonos: Convexos y Cóncavos. Para definir más claramente cuál es un polígono convexo y uno cóncavo se presenta la Figura 17:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



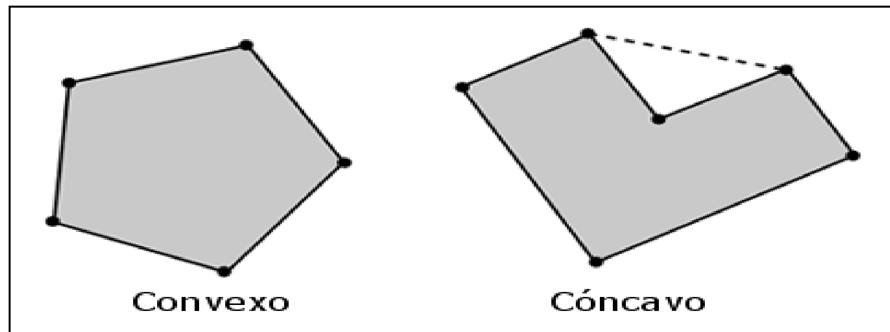



*Figura 17, Polígonos Convexo y Cóncavo*

Como se puede apreciar en la Figura 17, la principal diferencia entre ambos tipos de polígonos es que, el cóncavo posee vértices y bordes que forman una especie de agujero dentro de la Figura [1]. Esta diferencia es importante al considerar que las figuras cóncavas son más complejas a la hora de detectar colisiones (intersección de dos objetos dentro del espacio 3D), es por esto que gran parte de los motores 3D trabajan con figuras convexas [7].

A través de la conexión de múltiples polígonos, las herramientas de modelado 3D permiten crear figuras complejas denominadas mallas. Ahora bien, al pasar de los años, se ha podido percibir el incremento de polígonos en las figuras 3D, pasando, por ejemplo, desde el recordado Super Mario 64 (1996, Nintendo 64), hasta modelos ultra detallados como los de God of War III (2010, PlayStation 3). Así, en todo proyecto de desarrollo de videojuego, es crucial entender la importancia del número de polígonos que manejará el videojuego. Esto, debido a que un elevado número de polígonos, producirá una mayor carga de procesamiento a la hora de mostrar las figuras 3D en la pantalla. Así, se explica de porqué los modelos de hace unas décadas eran bastante más simples que los actuales; pero que gracias al gran avance de las tecnologías y especialmente de la potencia del hardware, hoy en día podemos apreciar modelos ultra realistas [4]. Para ilustrar de mejor forma, la Figura 18 presenta la diferencia que existe entre una misma figura 3D, con distinto número de polígonos.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



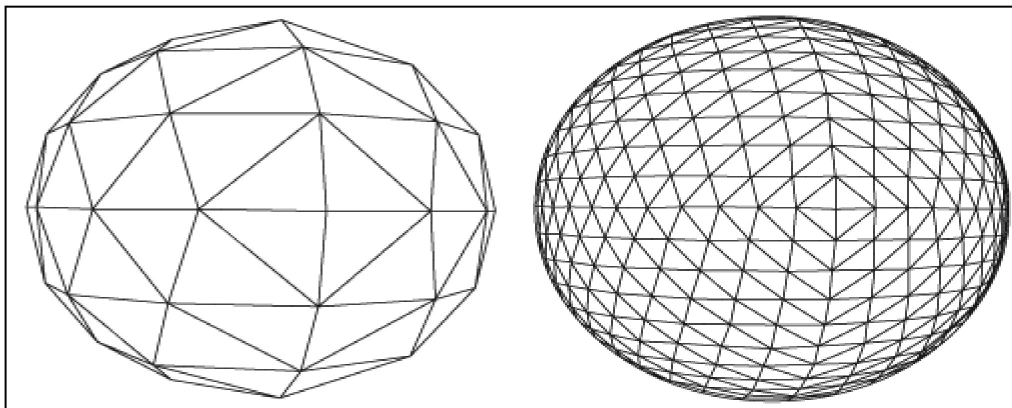


*Figura 18, Esfera con distinta cantidad de polígonos*

Como se puede apreciar en la Figura 18, la esfera de la izquierda posee un menor número de polígonos, y por ende es menos costosa para un computador mostrarla por pantalla, en comparación con la de la derecha, la cual posee un mayor nivel de detalle y más redondez, pero que requiere más cálculos para ser visualizada.

## 2.7 Texturas, Materiales y Shaders

Los videojuegos en general, desde sus inicios, han sido intensivos en cuanto al aspecto visual, en comparación con otros tipos de software. Es por ello, que el concepto de texturas es muy importante dentro de cualquier videojuego, producto que le dan el color, estilo y vida a éste. Básicamente, una textura corresponde a una imagen 2D, la cual es utilizada para "pintar" (utilizando la técnica conocida como mapeo de textura [14]) una figura geométrica 3D, dándole una apariencia más detallada y realista, sin la necesidad de contar con una figura 3D con una geometría muy compleja. Por ejemplo, una textura puede ser utilizada para pintar el suelo o terreno de un videojuego, para crear el cielo, dar color a las montañas, arboles, casas, etc. Específicamente, la técnica de mapeo de textura utiliza puntos dentro de la textura denominados texel, permitiendo así que la imagen posea su propio sistema de coordenadas. Luego, se mapea la textura asignando a cada uno de los vértices de una figura 3D, la coordenada de textura del correspondiente texel [14].

Por otro lado, existe un concepto muy relacionado con las texturas, y que es muy común en todas las aplicaciones 3D, esto es, los materiales. Un material posee un conjunto de características que permiten establecer la apariencia visual a un modelo o figura 3D. Cabe mencionar que un material puede estar compuesto por un simple color o por un conjunto de texturas [4]. Ahora bien, la apariencia que entregue un material a un modelo 3D, depende del shader que se utilice. Un shader define el estilo de

Universidad del Bío-Bío. Red de Bibliotecas - Chile



renderización de las imágenes. Por ejemplo, con un shader de reflexión, los materiales podrían reflejar los objetos que están a su alrededor, produciendo un efecto parecido al de un espejo o como el agua.

## 2.8 Iluminación

La luz, provee la fuente para iluminar los objetos dentro de un mundo virtual y entregar más realismo a la escena 3D. Las luces poseen diversas características, entre las que se pueden mencionar: un color, una posición, dirección y otras que dependen del tipo específico de luz [14]. Típicamente, en la mayoría de las aplicaciones 3D, existen 4 tipos de luces: la luz ambiente, la luz direccional, los puntos de luz y los focos de luz. La Figura 19, muestra los distintos tipos de luces que pueden existir dentro de un entorno 3D y particularmente, dentro de Unity3D:

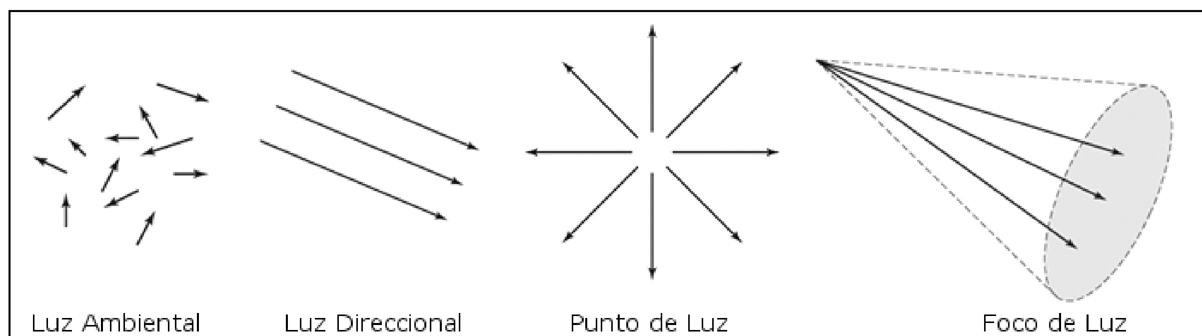



*Figura 19, Tipos de luces*

Como se puede apreciar en la Figura 19, la luz ambiente se caracteriza por ser uniforme en todas las direcciones y la cual produce sombras muy suaves. La luz direccional, al contrario, posee una dirección fija y la cual no posee una localización o posición específica. Por lo general, la luz direccional se utiliza para simular fuentes de luz como la del sol, además se caracteriza por crear sombras muy fuertes. Los puntos de luz, poseen una posición (punto) dentro del espacio 3D, desde donde emite en todas direcciones rayos de luz. Se caracterizan por disminuir su intensidad a medida que se alejan desde su fuente de origen, produciendo un efecto de atenuación. Similarmente, la luz tipo foco, también posee un punto de origen, desde donde emite rayos de luz y los cuales se direccionan de tal modo que forman una especie de cono (ver Figura 19). Por ende, este tipo de luz posee propiedades para definir la altura y radio de este cono, las cuales permiten especificar el alcance y área de influencia de la luz. Esta luz posee la característica de poder atenuarse o aumentar desde el centro del cono hacia fuera.

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# Capítulo III

# MOTORES DE VIDEOJUEGOS

A lo largo de la historia, el ser humano ha tenido siempre la curiosidad de descubrir y crear mundos distintos, que salen de su imaginación y que rozan la ficción. Prueba de ello han sido la gran cantidad de leyendas y cuentos mitológicos que existen en las diferentes culturas, que narran historias y aventuras de sus protagonistas. Similarmente, un videojuego sumerge al jugador en un mundo fantástico, que posee sus propios protagonistas e historia. Sin embargo, un videojuego, a diferencia de una leyenda o cuento mitológico, es intensivo en el uso de gráficos, aspectos visuales, sonidos, etc. Por ello, es un proceso complejo, costoso y que requiere mucho tiempo para su desarrollo. Es por esta razón, que la industria de los videojuegos ha buscado soluciones a este problema, desarrollando software especializado, que facilite la construcción de un videojuego. ¿La solución?, Los motores de videojuegos.

El siguiente capítulo tiene por objetivo abordar el tema de los motores de videojuegos, en donde se partirá con una definición que explique en qué consiste este tipo de software. Además, para comprender mejor éste concepto, es necesario también indagar un momento en la industria de los videojuegos y la historia de los motores de juegos. Adicionalmente, se hablará de los estilos de videojuegos que existen, los tipos de plataformas y de los roles que integran el desarrollo de un videojuego.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 3.1 Definición

Un motor de videojuego, en estricto rigor, es un conjunto de componentes de software reutilizable (librerías, interfaces, servicios, kit de desarrollos, etc.), que ayuda a los desarrolladores a abstraerse de los detalles de tareas y procesos que comúnmente se realizan en la construcción de un videojuego, como son, el renderizado, el manejo de la física y el control de dispositivos de entrada (teclado, mouse, joystick, etc.). Esto permite a los artistas, diseñadores, programadores, y a todo el grupo de trabajo, enfocarse solamente en la lógica del videojuego, del modelado y diseño de personajes (y demás objetos), del aspecto visual, etc., es decir, en los detalles que hacen al videojuego único [13]. Ciertamente, estamos en presencia de un software complejo, y como prueba de ello, la Figura 20 presenta un esquema general de la pila tecnológica que posee un motor de videojuego [12]:

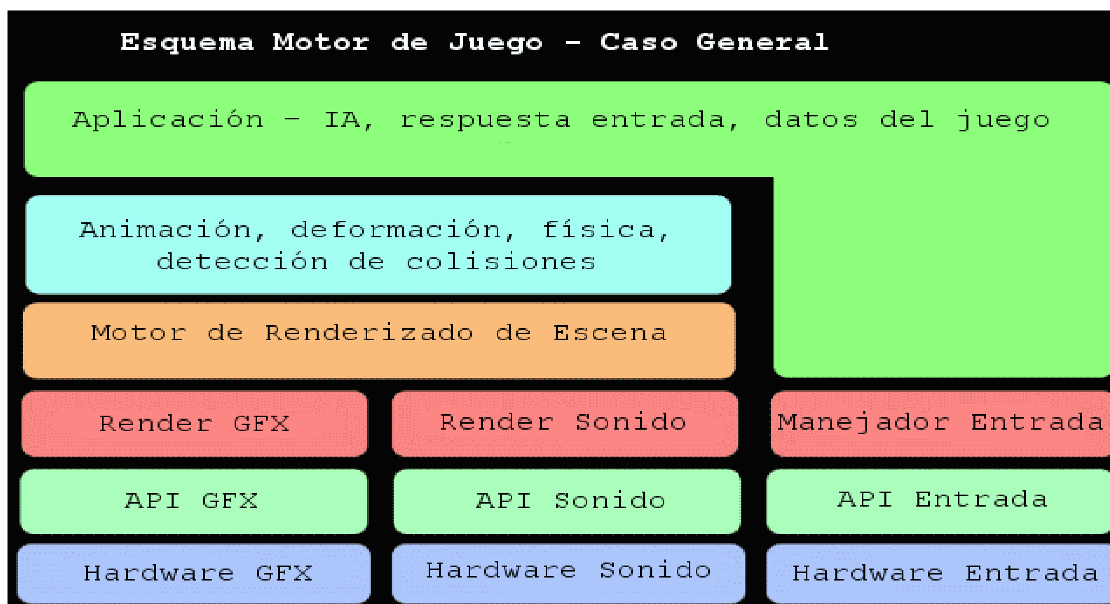



*Figura 20, Esquema general de un motor de videojuego*

Al apreciar la Figura 20, el nivel más bajo de la pila tecnológica está compuesto por el hardware, esto es, tarjetas gráficas, de sonido y dispositivos de entrada. Luego, un nivel más arriba, está las API (Application Programming Interface, *Interfaz de Programación de Aplicaciones*), que proveen el acceso al hardware del sistema. En el caso de las librerías gráficas (API GFX), existen principalmente dos posibilidades, Direct3D (para Windows) y OpenGL (multiplataforma). Para el sonido (API Sonido), podrían ser DirectSound o OpenAL y en el manejo de la entrada (API Entrada) DirectInput. Ahora bien, Render GFX es el responsable dibujar la escena (crear una imagen 2D a partir de los modelos y objetos 3D) que será mostrada por pantalla.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



Similarmente, Render Sonido, se ocupa de reproducir los sonidos, integrando diversos efectos, como por ejemplo el sonido 3D (la intensidad del sonido varía, dependiendo de la distancia y posición que se esté de él). El Manejador de Entrada captura los eventos provenientes desde el teclado, mouse u otros dispositivos de entrada, como joystick o gamepad, y los convierte a un formato que el motor de videojuego pueda entender. Por otro lado, el motor de renderizado utiliza métodos de bajo nivel para renderizar la escena por pantalla y reproducir los sonidos por el dispositivo de salida correspondiente. Las siguientes capas, son más de alto nivel y corresponden a un sin número de funciones que cooperan entre sí. Aquí se incluyen, funciones de animación, de detección de colisiones, de física (para calcular la gravedad o la masa de un personaje, por ejemplo), entre otras, dependiendo del motor. Finalmente, se halla el nivel de aplicación, que es en donde se encuentran los detalles específicos del videojuego, como son, los datos y la lógica, la inteligencia artificial de los personajes (controlados por el computador), la respuesta a la entrada de los usuarios, etc.

En definitiva, el motor de juegos se transforma en el núcleo de software de un videojuego.

## 3.2 La Industria de los Videojuegos

La industria de los videojuegos se caracteriza por ser un sector económico algo diferente en comparación a la tradicional industria de desarrollo de software. De hecho, ésta se asemeja más a la industria del cine (Hollywood), debido a que envuelve a productores, artistas, desarrolladores, distribuidores y una amplia gama de otras disciplinas [3]. Es más, ésta posee un perfil un poco más informal y la cual cuenta con sus propias celebridades, como por ejemplo Shigeru Miyamoto, reconocido productor y creador de videojuegos muy populares, tales como "Super Mario Bros.", "The Legend of Zelda" y "Donkey Kong".

Sin duda, es una industria muy joven, pero ha avanzado a pasos agigantados durante las últimas décadas. Ha pasado de ventas por $2.6 Billones dólares en el año 1996, a $7.4 Billones dólares en 2006. Por ejemplo, el año 2009 se vendieron aproximadamente 273 millones de juegos [5], cifra que ha crecido año tras año, y que de hecho ha incluso superado en ventas a la industria del cine [9]. Hoy en día, la industria de los videojuegos traspasa a todas las edades, culturas y géneros; se estima que el promedio de edad de los jugadores (también denominados "gamers") es de 34

Universidad del Bío-Bío. Red de Bibliotecas - Chile



años [5]. La Figura 21 presenta un gráfico que muestra el porcentaje de jugadores clasificados por edad:

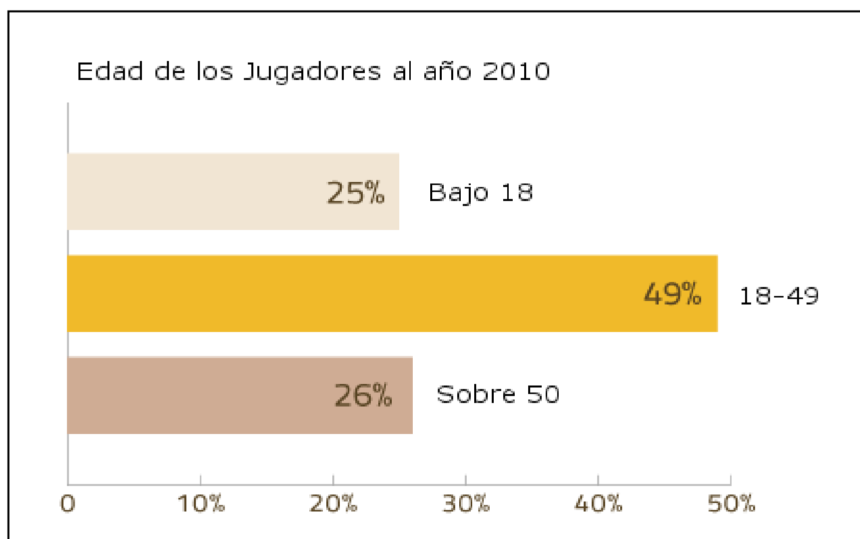



*Figura 21, Gráfico de porcentajes de edad entre los jugadores*

La Figura 21 demuestra que los videojuegos no son sólo un pasatiempo de los niños, sino que en una gran mayoría los practican personas adultas. Es más, no solo los practican los hombres, sino también una gran cantidad de mujeres. Cifras indican que el 40% de los jugadores son mujeres [5].

## 3.3 Historia

En un principio, allá por la década de los 80, generalmente un videojuego era desarrollado por un equipo de trabajo muy reducido, incluso por un solo programador y en un tiempo de unos pocos meses. Sin embargo, con el avance de la tecnología y el aumento en la complejidad de los videojuegos, el tiempo de producción y los equipos de desarrollos se extendieron, ya no solo lo integraban programadores, sino también diseñadores gráficos, sonidistas, entre otros. No obstante, en una época que se podría considerar como la edad de piedra del desarrollo de videojuegos, la mayoría de las compañías y equipos de trabajo construían desde cero cada videojuego, derrochando recursos y tiempo en cada nueva producción. Pero, con la invención de los motores de videojuegos, cambió radicalmente la forma de desarrollar videojuegos, ahora la reutilización de componentes de software es un hecho, se ahorra tiempo y costos en la realización de tareas que son comunes y habituales en el desarrollo de un videojuego. Es más, muchas empresas dejaron de producir videojuegos con el fin de especializarse en la construcción de motores de juego, para luego venderlo a otras empresas del rubro.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



Sin duda, hasta el día de hoy, han sido cientos los motores de videojuegos que han salido a la luz, de entre ellos, algunos de software libre, de código abierto y otros varios de tipo comercial. Ahora bien, entre los más relevantes que se podría mencionar están [6]:

- **Id Tech**: no fue realmente un verdadero motor de juego 3D, pero que en base a sprites (imágenes 2D, utilizadas para crear los videojuegos de antaño) simulaba perfectamente un efecto 3D. Fue lanzado en el año 1993 y destacaron los juegos Doom (I y II), Hexen, Chex Quest, Heretic y Strife.
- **Build:** al igual que el anterior, simulaba un mundo 3D en base a sprites. Además incorporó algunas mejoras, como incluir tags especiales dentro del juego, permitiendo a los desarrolles crear puntos que teletransporten al jugador a diferentes lugares. También para simular la caída a un agujero profundo y así volver al jugador a un sitio específico. Fue lanzado en el año 1993 y destacaron, principalmente los juegos Duke Nukem 3D, Blood, Extreme Paintbrawl, entre otros.
- **XnGine:** fue uno de los primeros motores de videojuegos 3D, haciendo uso de gráficos de alta resolución, de hecho, fue posible crear vastos mundos 3D. Su fecha de salida fue en el año 1995 y sobresalieron los juegos Draggerfall, Redguard y Battlespire.
- **Quake:** fue un motor de videojuego 3D, que introdujo nuevas técnicas que los hacían sobresalir entre lo demás, dentro de las que se cuentan el Z-buffering (método para determinar qué partes de una escena 3D son visibles, para así solo renderizar lo necesario), permitiendo reducir la demanda de procesador. Además incorporaba la utilización de iluminación 3D. Vio la luz en el año 1996 y destacaron los juegos Quake, Hexen II, Laser Arena, entre otros.
- **GoldSrc:** se caracterizó por dar soporte a las librerías gráficas más populares, esto es, OpenGL y Direct3D. Fue lanzado en el año 1998 y se utilizó para crear juegos como Half Life, Counter Strike, James Bond 007, entre otros.
- **Unreal:** fue uno de los más populares y el principal competidor del motor Quake II (versión mejorada de la anterior). Además, rompió los esquemas al no solo ser un motor de videojuegos en primera persona, sino que también sirvió como base para varios títulos de rol. Fue lanzado en 1998 y destacaron los juegos Unreal, Unreal Tournament, Harry Potter y Rune, principalmente.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



- **Quake III:** una versión mejorada de sus predecesores, dentro de lo que se puede mencionar un mayor énfasis en el uso de sombras, así como también la introducción de shaders, superficies curvas, color de 32 bits (más de 16 millones de colores, además del uso del canal alfa o transparencia) y el uso avanzado de redes. Así como AMD competía con Intel y Nvidia con ATI, en el mundo de los motores de juego, hacían lo propio Quake III con Unreal. Su fecha de salida fue en el año 1999 y destacaron Quake III Arena, Call of Duty, Medal of Honor y varios más.

- **Torque:** dentro de sus principales características fue incluir un editor de terrenos, de interfaz gráfica de usuario, el control del nivel de detalle (capacidad de reducir el número de polígonos de los objetos 3D) y también capacidades para juegos multiplayer en línea. Salió en el 2001 y destacaron juegos como Penny Arcade Adventures, Tribes 2 y como no mencionar a Uko, videojuego educativo 3D desarrollado en la ciudad de Chillán, por la empresa Jobbitgames.

- **Gamebryo:** fue diseñado para ser multi-plataforma, pudiendo integrarse con herramientas de modelado 3D (por ejemplo, 3DS Max y Maya), soporte para detección dinámica de colisiones, sistema de colisiones, audio 3D, etc. Se lanzó en el año 2003 y destacaron juegos como Oblivion, Fallout 3, Prince of Persia 3D, entre otros.

- **Unreal 3:** es la versión más avanzada del motor Unreal, con soporte para PC y para otras plataformas como Xbox 360 y PlayStation 3. Se caracteriza por poseer y hacer uso de gran parte de las funcionalidades y tecnologías utilizadas hoy en día por los videojuegos modernos, esto es, avanzados motores de física, complejos sistemas de animación de esqueletos, sistemas de partículas, entre muchas otras. Fue lanzado en el año 2007 y destacaron juegos como Gears of War 1 y 2, Bioshock 1 y 2, Mass Effect 1 y 2, etc.

En definitiva, existe una amplia gama de posibilidades, para todos los gustos y tipos, pero que al fin y al cabo todos con un objetivo en común, facilitar la compleja tarea de desarrollar un videojuego.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 3.4 Géneros y Estilos de Videojuegos

A la hora de crear videojuegos, la primera regla para realizar un diseño creativo es que no hay reglas y límites que restrinjan la imaginación de los desarrolladores [3]. Esto se debe a que en la industria y desarrollo de videojuegos, predomina la creatividad y las ideas innovadoras. Sin embargo, existen una serie de géneros y estilos de videojuegos, que es recomendable conocer y analizar, ya que de ésta manera, sirven como una guía al momento de generar las ideas y el estilo del videojuego a desarrollar. Un género engloba a una lista de videojuegos, que poseen características y similitudes que son comunes entre ellos.

Ahora bien, a lo largo del tiempo, ha sido innumerable la cantidad de videojuegos que han salido a la luz, cada uno con su propia identidad y estilo, pero que sin duda pueden ser clasificados y ajustados a géneros bien definidos. A continuación se mencionan los géneros y estilos de videojuegos más comunes y reconocidos dentro de la industria [3]:

- **Juegos de Acción:** como su nombre lo indica, se caracterizan por ser intensivos en combate e interacción entre los jugadores y los oponentes del videojuego. Éste género, puede tomar diversas formas, dentro de las que destaca el estilo de primera persona (FPS, First-Person Shooter), el cual se caracteriza por poner al jugador en los ojos del personaje que controla, permitiendo así sumergirlo más intensamente en la atmosfera del videojuego (como si el jugador estuviese dentro del juego). También, se puede encontrar dentro de este género, el estilo de tercera persona, el cual consiste en que la vista del jugador esta posicionada a una distancia específica del personaje que controla y no desde los ojos de éste, como en el estilo anterior. Dentro de este género se pueden encontrar juegos como Doom, Quake, Halo, God of War, entre otros.

- **Juegos de Aventura:** consisten básicamente en juegos de exploración y búsqueda de una meta u objetivo. Por lo general, el jugador debe superar diversos retos para avanzar en el juego, a través, por ejemplo, de la búsqueda de objetos e ítems especiales, resolver puzles y derrotar enemigos concretos. Uno de los exponentes más destacados de este género es la saga The Legend of Zelda.

- **Juegos de Rol (RPG, Role-Playing Games):** este tipo de género se caracteriza por estar basados en mundos de fantasía inmensos, visualmente muy atractivos y llenos de detalles. Además, poseen historias muy complejas y en donde cada

Universidad del Bío-Bío. Red de Bibliotecas - Chile



personaje dentro del juego, incluso los enemigos, poseen una identidad propia. Aquí, el jugador, por lo general, es responsable de desarrollar las habilidades y experiencia del o los personajes que controla, permitiéndole así personalizar a éstos de manera distinta a otro jugador del mismo videojuego. Sin duda, en este género destaca principalmente la saga Final Fantasy, pero también existen otros como Chrono Trigger, Chrono Cross, entre otros.

- **Juegos de Puzle:** consisten básicamente en resolver puzles, careciendo, por lo general de una historia central y enemigos de por medio. En este género es muy común el uso de iconos en la pantalla, los cuales deben ser manipulados y controlados por el jugador de tal forma que permita resolver un problema planteado. Aquí, como ejemplo y referente para todo tipo de juegos de puzle esta el conocido Tetris.

- **Juegos de Simulación:** aquí el objetivo principal es reproducir mediante un videojuego, una situación del mundo real. El factor clave dentro de este tipo de género es, la precisión, fidelidad y realismo con que se simula el mundo real. Es por esto que se coloca especial énfasis en la apariencia visual, sonidos y física del juego, con el fin de sumergir al jugador dentro de la atmosfera del videojuego de tal forma que se sienta, por ejemplo, que esta maniobrando un avión o un automóvil de verdad. Dentro de este género destaca The Sims, SimCity, Gran Turismo, Nintendogs, etc.

- **Juegos de Deportes:** pese a que pueden ser considerados como una especie de juegos de simulación, los juegos de deportes son clasificados dentro de su propio género. Como ya se puede sospechar, se trata de videojuegos que simulan los deportes tradicionales del mundo real. Aquí, se pueden encontrar juegos de fútbol, beisbol, basquetbol, de tenis, de boxeo, entre muchos otros. Dentro de este género destacan, juegos como Pro Evolution Soccer, Virtua Tennis, Wii Sports, etc.

- **Juegos de Estrategia:** se centran en que el jugador debe controlar, estratégicamente, a múltiples unidades (ejércitos o tropas de personajes) dentro del juego, con el fin de derrotar a sus oponentes (destruirlos y apoderarse de sus bases) y así lograr la victoria. Aquí destacan los juegos como StartCraft, WarCraft, Command and Conquer, entre otros.

Claramente, hoy en día, existen videojuegos que caen o se ajustan a varios de los géneros y estilos mencionados aquí. Esto debido a que algunos juegos incorporan,

Universidad del Bío-Bío. Red de Bibliotecas - Chile



dentro de sus opciones, diversos modos de juego con el fin de ofrecer una mayor variedad y flexibilidad al jugador. Así también, para crear o implantar un nuevo estilo o género de videojuego. Otra consideración importante, es que, en cada uno de los géneros mencionados anteriormente, podrían ser incluidos modos de juegos que permiten interactuar a varios jugadores (juegos Multiplayer o de múltiples jugadores) en el mismo juego, e incluso, desde lugares remotos o de diferentes plataformas (juegos online o en línea), pudiendo así ser considerados como otros géneros de videojuego.

### 3.5 Plataformas

Al igual que en el desarrollo de software tradicional, en la industria de los videojuegos existen diversas plataformas en las cuales se puede desarrollar. En general, se puede encontrar principalmente dos tipos de plataformas en la que se puede distribuir un videojuego: en un computador personal (PC, Personal Computer) o en una consola de videojuego (incluyendo las versiones portátiles) [3]. Aunque, no se puede dejar de mencionar las plataformas móviles, como son los teléfonos celulares, los cuales están teniendo un gran auge hoy en día.

Ahora bien, en el caso de los computadores personales, existe una subdivisión de plataformas, dentro de las que están: Windows, Macintosh y Linux. A la hora de desarrollar en cada una de estas plataformas, existen diversas opciones y herramientas a utilizar, unas más específicas, como por ejemplo Direct3D, que es solamente para Windows y otras multiplataforma y de código abierto, como OpenGL [3]. En el caso de las consolas de videojuegos, existen igualmente diversas plataformas en las cuales desarrollar, una son, las consolas domesticas: Nintendo Wii, Xbox 360 y PlayStation 3, y la otra, las portátiles: Nintendo DS y PlayStation Portable. La selección de la plataforma más adecuada para un desarrollo, dependerá de las necesidades y recursos que posea el equipo de trabajo, ya que por ejemplo, para realizar un desarrollo de un videojuego para una consola, las herramientas de desarrollo son más privativas y solo se obtienen a través de la compra de licencias que poseen un elevado costo económico [3], a diferencia de las herramientas para computadores personales, que son más asequibles para los grupos de desarrollos más pequeños. Otro factor a considerar, son las tecnologías y diversidad de opciones disponibles en cada plataforma, por ejemplo, en el caso de Nintendo Wii y PlayStation 3 (con su dispositivo Move), permiten hacer uso de tecnologías de detección de

Universidad del Bío-Bío. Red de Bibliotecas - Chile



movimiento, que son más atractivas para el jugador final, en cambio con la plataforma de computador, se puede hacer uso de internet para llegar a más usuarios.

## 3.6 Roles Dentro del Desarrollo de un Videojuego

Como se mencionó anteriormente, la era en que los videojuegos eran desarrollados por una sola persona, ha quedado en el pasado. En la actualidad, los equipos de desarrollos de videojuegos, están compuestos por una gran cantidad de personas y diversas disciplinas, las cuales deben cooperan entre sí. Por ende, en el desarrollo de videojuegos, al igual que en el desarrollo tradicional de software, es muy común encontrar diversos roles dentro del equipo de trabajo, con el fin de coordinar y organizar las diferentes tareas a realizar. A continuación se describirán los roles más comunes e importantes dentro de cualquier equipo de desarrollo de videojuegos:

### 3.6.1 Productor

El productor es esencialmente el líder del equipo de desarrollo [3]. Se encarga de todo lo relacionado con la administración del proyecto, destacando tareas como la planificación, la gestión del presupuesto y de los gastos, la coordinación de diferentes secciones del equipo de trabajo, resolver problemas, negociar contratos y licencias, ser la interfaz entre el mundo externo y el equipo de desarrollo, entre otras [3]. Por lo general, los productores no conocen cada uno de los detalles de la producción de un videojuego, pero sí, todo lo que respecta al equipo de desarrollo, esto es, las tareas encomendadas y el avance del proyecto.

### 3.6.2 Diseñador

El diseñador es el encargado de hacer que el videojuego sea divertido y atractivo para los jugadores. Dentro de sus tareas se encuentran decidir cuál será la temática y las reglas del juego, diseñar el mapa y los niveles, describir la historia, entre otras [3]. Por lo general, un diseñador debe escribir un documento de diseño, el cual contiene todas las descripciones de los elementos que poseerá el videojuego, y cómo éstos se unen para formar un todo. Así, este documento sirve como base y guía para el trabajo de los demás integrantes del equipo de desarrollo. Por lo tanto, un diseñador necesita ser un buen comunicador, ya que debe transmitir correctamente las ideas y todos los detalles que el videojuego tendrá, a los otros miembros del equipo [3]. Ahora bien, a diferencia del productor, un diseñador si debe conocer los aspectos técnicos del desarrollo de un videojuego, con el fin de comprender las herramientas y así saber qué es lo que se puede y no se puede hacer.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



### 3.6.3 Programador

Los programadores de videojuegos son los encargados de escribir el código y los programas que permiten transforman las ideas, el arte, el sonido y la música, en un juego completamente funcional [3]. Controlan la relación causa-efecto de los eventos que se producen dentro del videojuego, además, convierten las entradas del usuario en experiencias visuales y auditivas [3]. A menudo, los programadores se especializan en subsistemas del juego, tales como la de gráficos, de red, de inteligencia artificial, de audio, o en herramientas específicas, como editores para el juego.

### 3.6.4 Artistas

Son los encargados de dar el toque artístico y atractivo al videojuego. En esta categoría, podemos encontrar dos roles importantes, los artistas visuales y los de audio.

#### 3.6.4.1 Visuales

Los artistas visuales poseen la responsabilidad de crear el contenido gráfico y visual de un videojuego. Dentro de las tareas de un artista visual, se pueden distinguir tres principalmente: el modelado, la animación y la creación de texturas [3]. En la etapa de modelado se diseñan y construyen los modelos 3D de los personajes, enemigos y todos los objetos que el videojuego tendrá. En la etapa de animación, se toman los modelos de la etapa anterior, con el fin de darles movimiento. Luego, en la etapa de texturizado, a través de la creación de texturas, se pintan y se da color a los objetos 3D modelados. En definitiva, los artistas visuales, son los encargados de crear la atmósfera y el estilo visual del videojuego.

#### 3.6.4.2 Sonido

Los artistas de audio son los encargados de componer la música y los efectos de sonido del videojuego [3]. Se caracterizan por trabajar estrechamente con los diseñadores, con el fin de determinar cuáles efectos de sonido son los más adecuados y se adaptan mejor al concepto del videojuego. En fin, los artistas de audio deben ser capaces de transmitir emociones al jugador a través la música y sonidos.

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# Capítulo IV

# UNITY3D

---

La Tecnología es una herramienta que nos ayuda a realizar cosas impresionantes, con mayor rapidez y facilidad que usando métodos tradicionales.

Es por esto, que el siguiente capítulo tiene como objetivo describir las principales características de la herramienta a utilizar en el desarrollo de este proyecto, esto es, Unity3D. Cabe destacar, que ésta herramienta posee varios conceptos, que es recomendable conocer y manejar antes de comenzar a desarrollar un proyecto, ya que éstos definen la forma de trabajo de Unity3D.

Una vez finalizado éste capítulo, se estará en condiciones de comenzar con el desarrollo del videojuego.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 4.1 Características

El motor Unity3D se puede definir como una herramienta para la creación de videojuegos de distintos estilos, el cual está disponible en dos versiones, una gratuita (para fines más bien educativos y sin lucro) y otra de pago (para fines comerciales), tanto para las plataformas de Windows (utilizada en el proyecto) y de Mac, aunque también para IPhone (incluyendo el IPad), Wii y Android, pero las cuales requieren de una licencia adicional.

Ahora bien, Unity3D permite transformar la producción de un videojuego en un proceso simple [4], debido a que provee un lienzo en blanco y un conjunto de procedimientos coherentes que permiten que la imaginación sea el límite de la creatividad de los desarrolladores.

Por otro lado, la mayoría de los videojuegos de consolas y de PC modernos, son construidos bajo el lenguaje de programación C++, debido que es actualmente el más eficiente de todos en términos de velocidad [4]. Unity3D no es la excepción, ya que utiliza la librería Mono de código abierto C++. Adicionalmente, Unity3D también toma las ventajas de otras librerías de software para su funcionamiento, como son el motor de física de NVidia PhysX, OpenGL y DirectX para el renderizado 3D y OpenAl para el manejo del audio. Eso sí, hay que considerar que todas estas librerías están dentro de Unity3D, por lo tanto, a la hora de construir el proyecto, el equipo de desarrollo se abstrae de todos estos detalles y no debe preocuparse de entender cómo funciona cada uno de ellos, simplemente los debe utilizar y ocuparse de diseñar lo particular y diferente de su videojuego.

Unity3D se compone de los siguientes elementos: Assets, escenas, Game Objects y Prefabs. A continuación, se introduce al lector cada uno de estos conceptos, con el fin de entender su funcionamiento, además se abordará la interfaz de usuario de Unity3D y las herramientas externas necesarias para el desarrollo de un proyecto de videojuego.

## 4.2 Los Assets

Los "*Assets*" (o activos, traducido al español) en Unity3D, se refieren a todos los componentes básicos para la construcción de un proyecto. Estos van principalmente, desde los gráficos en forma de archivos de imagen (.png, .jpg, .psd, etc.), hasta los modelos 3D y los archivos de sonido [4]. En resumen, son todos los recursos que se utilizan para formar, tanto visualmente, como también lógicamente el videojuego.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



Unity3D organiza, en forma de jerarquía de carpetas, cada uno de los Assets del proyecto, permitiendo al usuario acceder rápidamente a cada uno de ellos a través del llamado panel de proyecto (tema que se tratará más detalladamente en las siguientes secciones). Además, provee un conjunto de funcionalidades y operaciones que permiten gestionar y manipular los Assets, dentro de las que se encuentran, importar crear y eliminar. Cabe destacar que, es muy recomendable utilizar cada una de estas funcionalidades a la hora de administrar los Assets del proyecto, debido a que Unity3D maneja un archivo de metadatos, con el cual lleva un registro de todos los Assets, por lo que si se elimina uno, utilizando, por ejemplo, el explorador de Windows, podría producir errores o incluso corromper el proyecto.

## 4.3 Las Escenas

El concepto de escena en Unity3D, corresponde a un nivel o escenario individual dentro del juego, específicamente a un mundo virtual 3D completo, con sus propias características y objetos. Un videojuego puede estar compuesto por múltiples niveles, el cual puede corresponder por ejemplo, a una isla donde el jugador debe recorrer para buscar algún objeto, o incluso un menú dentro del videojuego. Cada una de las escenas, en Unity3D, son formadas y construidas en base a los Assets disponibles en el proyecto.

## 4.4 Los Game Objects

Los Game Objects son el concepto más importante dentro de Unity3D, debido a que cada uno de los objetos dentro del juego es un Game Object, de hecho, cada Assets, al ser incorporado o insertado dentro de una escena, pasa a convertirse en un Game Object. Sin embargo, un Game Object por sí solo no realiza ninguna acción, es decir, necesitan propiedades especiales antes de que puedan convertirse por ejemplo, en un personaje, en un auto o en un efecto especial [10]. Los Game Objects son contenedores, los cuales son dotados de diferentes propiedades y comportamientos que los hacen diferenciarse de los demás. A estas propiedades y comportamientos se le denominan componente, los cuales serán cubiertos en detalle en la siguiente sección.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 4.5 Los Componentes

Un Componente corresponde a una pieza funcional, que forma parte de un Game Object, el cual permite definir diferentes aspectos y propiedades, como por ejemplo, crear un comportamiento particular, definir una apariencia, hacer interactuar un Game Object con otros, etc. Es por esto, que el concepto de Componente ésta estrechamente relacionado con el de Game Object, debido a que cada Game Object dentro de una escena, posee al menos un Componente, el cual corresponde al denominado "*Transform*" [4]. El Componente "*Transform*", es uno de los más importantes dentro de Unity3D, ya que define la posición, rotación y escala de un Game Object dentro de una escena [4].

Cabe mencionar, que un Game Object puede estar compuesto por diversos Componentes, cada uno con sus propias características y funcionalidades. Además, Unity3D provee diversos Componentes predefinidos, listos para ser utilizados y manipulados por los desarrolladores, como por ejemplo, Componentes para el manejo y manipulación de la física de un Game Object, para crear fuentes emisoras y receptoras de audio, para crear efectos especiales de partículas como el fuego o una explosión, entre otros.

Como se mencionó anteriormente, los Componentes sirven para crear y definir un comportamiento o modo de funcionamiento particular de un Game Object. Esto se logra a través de los denominados Scripts, los cuales serán definidos en la siguiente sección.

## 4.6 Scripts

Los Scripts son una parte esencial en el proceso de producción de un videojuego, en especial en Unity3D, ya que son los componentes que entregan la funcionalidad y comportamiento a los Game Objects dentro de una escena [4]. A través de ellos, se puede controlar cualquier objeto y componente, así como también modificar sus características y propiedades, e incluso, eliminar y crear elementos en la escena. En definitiva, proveen la parte lógica del videojuego. Ahora bien, los Scripts en Unity3D son implementados a través de lenguajes de programación orientados a objetos, existiendo tres opciones de donde elegir [4]: C#, Boo (una variación del lenguaje Python) y JavaScript (versión adaptada para Unity3D), siendo este último (del cual está basada toda la documentación de Unity3D), el utilizado en el desarrollo del proyecto, aunque pueden ser usados los tres en conjunto. Cabe destacar que, Unity3D,

Universidad del Bío-Bío. Red de Bibliotecas - Chile



provee una aplicación para escribir y editar Scripts, esto es, UniSciTE para Windows y Unitron para Macintosh. Cada Script dentro de Unity3D corresponde a un comportamiento (Behavior en inglés), el cual puede poseer un conjunto de atributos y funciones que interactúan entre sí para crear una funcionalidad específica.

Un atributo consiste en una variable, que representa un objeto (Game Object) o componente dentro del juego, el cual posee un tipo o clase. Unity3D provee una amplia diversidad de clases, las cuales se utilizan para representar la gran cantidad de objetos y componentes que éste provee. Además, se pueden crear clases propias, de hecho, todo nuevo Script define un nuevo tipo o clase, el cual puede ser instanciado o referenciado por otro Script. Para más detalles sobre cómo implementar un script, específicamente en JavaScript, ver Anexo B.

### 4.6.1 Operaciones Comunes

Como se mencionó anteriormente, cada Script en Unity3D define un comportamiento, es más, al crear un Script utilizando JavaScript, éste deriva automáticamente de la clase base para todos los Script, esto es, `MonoBehaviour` [11]. De esta clase se heredan diversos atributos y funciones que permitirán a los desarrolladores manipular objetos de una escena y también para acceder a todas las funcionalidades que Unity3D ofrece. De hecho, Unity3D posee una amplia API, que permite abstraerse de los detalles de implementación del motor y la cual está compuesta por un sin número de clases que representan cada uno de los objetos y componentes que Unity3D posee. A continuación, se presenta la jerarquía de las clases más importantes de Unity3D [11]:

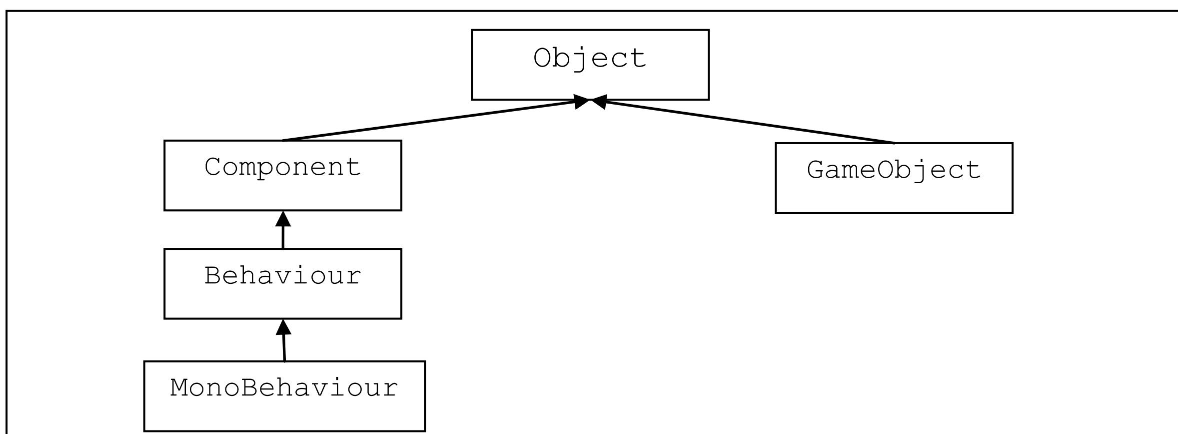



*Figura 22, Jerarquía de clases importantes de Unity3D*

Universidad del Bío-Bío. Red de Bibliotecas - Chile



Como se puede observar en la Figura 22, la base de todas las clases de Unity3D es `Object`. También, que las clases `GameObject` y `Component` heredan de ella. La clase `GameObject` representa la base para todas las entidades u objetos que se encuentren dentro de una escena. Por otro lado, la clase `Component` sirve como base de todos los componentes que sean unidos o añadidos a un Game Object (recordar que los Game Object son contenedores de componentes). Es por esto que `Behaviour` y `MonoBehaviour` (clase base para todos los Script) derivan de `Component`, ya que estos se añaden a un Game Object.

Ahora bien, al crear un Script, es posible utilizar una serie de atributos y funciones que son heredados de las clases anteriormente mencionadas y las cuales entregan la base para las operaciones que son comunes en todo videojuego, esto es, manipular la posición, rotación y escala de un objeto. Para ello, se puede hacer uso de la variable `transform` (objeto de la clase `Transform`), que representa la posición, rotación y escala del objeto (Game Object), al cual está unido el Script. Cabe recordar que, todo Game Object en Unity3D posee el componente `Transform`, por lo que esta variable nunca será nula, es decir, que no referencie a ningún componente, ya que algunas de las variables heredadas, como `rigidbody` (de la clase `RigidBody`), son nulas, en caso de que el Game Object no posea ese componente. A modo de ejemplo, a continuación se presenta algunas de las operaciones que se pueden realizar a un objeto a través de la variable `transform` [11]:

```
1 transform.Rotate(0, 5, 0); //Rotar 5 grados un objeto en torno al eje
2 Y
3 transform.Translate(2, 0, 0);//Trasladar 2 unidades un objeto a
4 través del eje X
5 transform.localScale.z = 2; // Escalar al doble un objeto en el eje Z
```

*Figura 23, Operaciones a través de la variable transform*

La Figura 23, muestra las operaciones básicas de transformación a un objeto, a través de las funciones de la clase `Transform`, como son `Rotate` (para rotar un objeto), `Translate` (para trasladar), entre otras. Notar también que, es posible acceder a los miembros de la variable `transform`, como es el caso de `localScale` (línea 5), que representa la escala del objeto a través de un vector de tres elementos(x, y, z), el cual pertenece a la clase `Vector3`.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



### 4.6.2 Funciones Importantes

Al crear un Script, se pueden implementar nuevas funciones que realicen tareas específicas de acorde a las necesidades y características del videojuego. Sin embargo, existen una serie de funciones (heredadas), que son de gran ayuda a la hora de construir un Script. Todas estas funciones, poseen la característica de que son llamadas o invocadas automáticamente por el motor, debido a un evento producido, sin necesidad de intervención del programador. Solamente, el programador debe preocuparse de sobrescribir y darle la funcionalidad que se requiera a éstas. Más específicamente, Unity3D provee una gran cantidad de funciones, cada una con distintos fines. Dentro de las más importantes se encuentra la función `Update`, la cual se invoca automáticamente por cada frame, permitiendo así realizar acciones cíclicas y que necesiten ser actualizadas periódicamente, como por ejemplo la entrada del usuario, una animación, una medida de tiempo, etc. También, se encuentra la función `OnCollisionEnter`, la cual es invocada automáticamente cuando el objeto colisiona con otro dentro de una escena. En fin, un sin número de funciones que ayudan de sobre manera a los desarrolladores a la hora de crear los comportamientos para el videojuego. Para más detalles sobre las funciones que provee Unity3D, ver Anexo C.

## 4.7 Los Prefabs

Los Prefabs se pueden definir como un tipo de Asset reutilizable, son como una plantilla, la cual puede ser instanciada o creada dentro de una escena en múltiples ocasiones [10]. De hecho los Prefabs están pensados para almacenar la configuración de Game Objects complejos, que poseen una gran cantidad de componentes y propiedades. Cuando se añade un Prefabs a una escena, se crea una instancia de éste, un clon, el cual queda conectado con el Prefab original. Así, no importando cuantas instancias de un Prefab existan dentro de una escena, si se modifica el Prefab original, estos cambios se verán reflejados en cada una de las instancias de éste. Sin embargo, una instancia de un Prefabs, puede perder la conexión con el original [10], debido a un cambio que se le realice a la instancia, como puede ser, añadiéndole o quitando un componente, por lo que la instancia será un objeto individual y no dependerá del Prefab original.

En Unity3D, los Prefabs se crean dentro del panel de proyecto, donde se encuentran los Assets. Al momento de crearlos, éstos son como una plantilla vacía o en blanco, la cual no contiene ningún Game Object, por lo que no puede ser instanciada [10]. Para poder instanciar un Prefabs, éste debe ser "rellenado" con un

Universidad del Bío-Bío. Red de Bibliotecas - Chile



Game Object creado dentro de una escena. Esto se logra mediante la selección del Game Object dentro del panel de jerarquía, para luego arrastrarlo encima del Prefab, el cual almacena toda la información y configuración del Game Object. Notar que el Game Object seleccionado dentro de la escena queda automáticamente conectado con el nuevo Prefab, convirtiéndose así en una instancia de éste.

## 4.8 La Interfaz de Usuario

Otro de los aspectos importantes de Unity3D es su interfaz de usuario, ya que a través de ésta se realiza todo el trabajo de implementación y desarrollo del videojuego. La interfaz de Unity3D, está compuesta por diversos paneles, los cuales realizan tareas y funcionalidades concretas. A continuación, se presenta una captura de la interfaz completa de Unity3D, conteniendo sus paneles más importantes [10]:

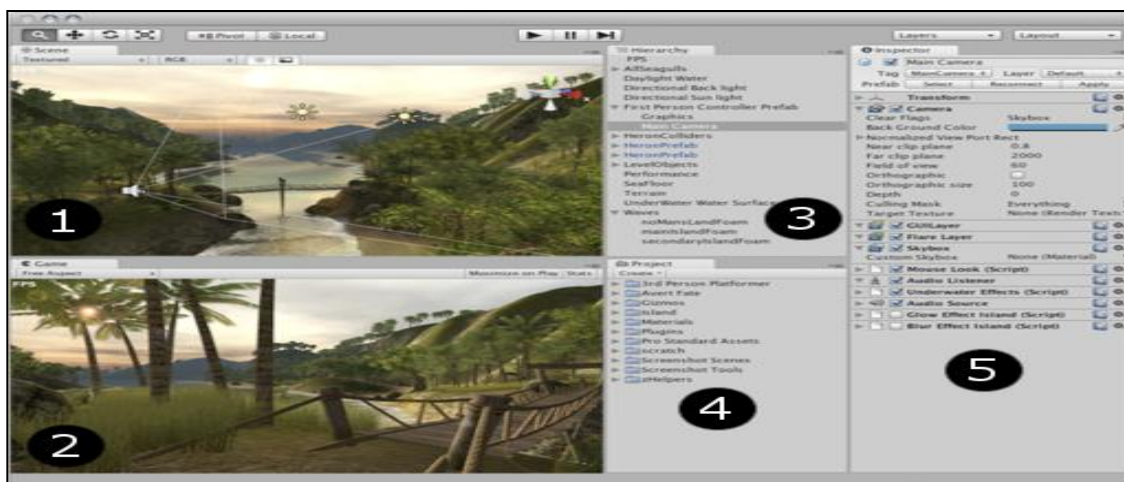



*Figura 24, Interfaz de Usuario de Unity3D*

Como se puede apreciar en la Figura 24, la interfaz de Unity3D está compuesta principalmente por 5 secciones, los cuales corresponden a: la ventana de escena (1), la ventana de juego (2), el panel de jerarquía (3), el panel de proyecto (4) y el panel del inspector. En las siguientes secciones se explicarán cada uno de estos paneles.

### 4.8.1 La Ventana de Escena

La ventana de escena es donde se construye completamente el videojuego, corresponde al área en la cual se visualiza por completo el mundo 3D, donde se posicionan los objetos. Esta ventana ofrece una vista en perspectiva completamente 3D, la cual puede ser transformada a una vista ortográfica [4]. En esta ventana es donde se “arrastran” los Assets del proyecto, los cuales se convierten en Game Objects

Universidad del Bío-Bío. Red de Bibliotecas - Chile



dentro de la escena. Además, la ventana de escena cuenta con cuatro herramientas [4]:

*Figura 25, Herramientas de la ventana de escena*

Estas cuatro herramientas, como se aprecia en la Figura 25, consisten en la herramienta de mano (accesible a través de la tecla Q, del teclado), la cual permite navegar a través de la escena por medio de una vista panorámica. Otra herramienta es la de traslación (tecla W), la cual permite trasladar o cambiar de posición a través de los ejes X, Y, Z, un objeto seleccionado. Igualmente, está la herramienta de rotación (tecla E), la cual permite rotar, en los ejes X, Y, Z, objetos dentro de la escena y por último, la herramienta de escala, que ajusta visualmente el tamaño o escala de un objeto (ejes X, Y, Z, nuevamente). También, existe un modo de navegación muy útil dentro de la ventana de escena, el cual corresponde a un modo estilo primera persona. Esto se logra manteniendo presionado el botón derecho del mouse y utilizando las teclas W, A, S, D, del teclado para navegar a través de toda la escena 3D.

### 4.8.2 El Panel de Jerarquía

El panel de jerarquía contiene cada uno de los Game Object dispuestos dentro de la ventana de escena [10]. Algunos de estos objetos son instancias directas de un Asset como por ejemplo un modelo 3D, y otros son instancias de un Prefab. Dentro de este panel se pueden seleccionar cada uno de los objetos dentro de la jerarquía, esto incluye, a los "padres" e "hijos". Un padre consiste en un Game Object que contiene a otros Game Object (hijos) dentro de su jerarquía, los cuales heredaran los movimientos y rotación de su padre, es decir, si el padre se traslada o cambia de posición, sus hijos también lo harán [10]. Este concepto es recursivo, ya que los hijos también pueden ser padres de otros Game Object. A continuación se presenta una imagen que muestra el panel de jerarquía:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



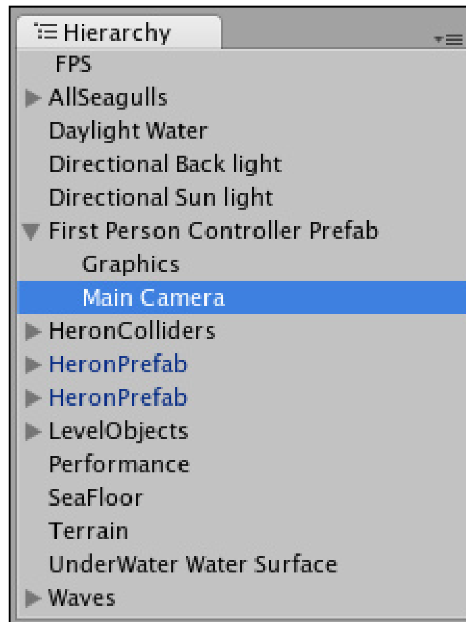



*Figura 26, Panel de Jerarquía*

Como se puede apreciar en la Figura 26, existen unos iconos (flechas) antecediendo el nombre de algunos Game Object, los cuales, al seleccionarlos, permiten expandir la jerarquía de los objetos que poseen hijos. Además, se puede observar que los nombres de algunos objetos están con letra azul. Esto indica que esos objetos corresponden a una instancia de un prefab.

### 4.8.3 El Inspector

El inspector muestra información detallada acerca de un Game Object particular (seleccionado desde la ventana de escena o del panel de jerarquía), incluyendo cada uno de los componentes y sus propiedades correspondientes [10]. Este panel, además, permite modificar los componentes y propiedades de un Game Object, incluyendo los miembros públicos de un Script. La Figura 27, muestra una captura del panel de inspección [10]:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



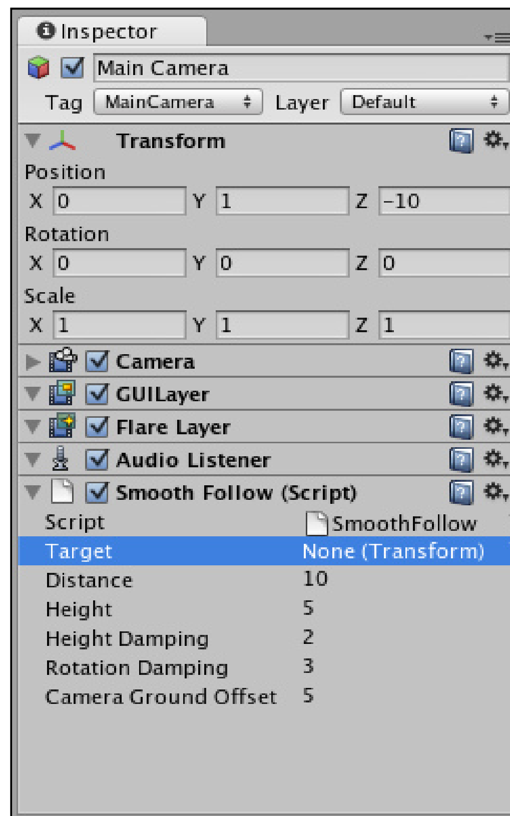



*Figura 27, El Inspector*

Como se puede apreciar en la Figura 27, desde el inspector se puede modificar las propiedades de transformación de un Game Object (posición, rotación y escala). Además, es posible cambiar el nombre del objeto, asignarle un tag especial para facilitar su búsqueda dentro de todos los Game Object de una escena, cambiar las propiedades de sus componentes, entre otras opciones. Como se mencionó anteriormente, se puede modificar un Script añadido al Game Object desde este panel, en particular, al observar la Figura 27, existe una variable llamada "Target", dentro del Script "SmoothFollow", la cual posee un valor nulo (no asignado). Bueno, este valor puede ser asignado arrastrando un Game Object desde el panel de jerarquía o incluso un prefab desde el panel de proyecto. Así, al utilizar esta variable dentro del Script, se estará referenciando al objeto asignado y el cual podrá ser modificado desde allí.

### 4.8.4 El Panel de Proyecto

El panel de proyecto contiene cada uno de los Asset del proyecto, permitiendo a los desarrolladores acceder directamente a ellos, sin la necesidad de usar el explorador, en el caso de Windows o el finder, en el caso de Macintosh [4]. Desde este panel nacen y se almacenan (dentro de carpetas) la mayoría de los Game Object y

Universidad del Bío-Bío. Red de Bibliotecas - Chile



componentes de una escena (modelos 3D, imágenes, sonidos, animaciones, prefabs, Scripts, etc.), e incluso las propias escenas. Cabe mencionar que, para importar recursos como modelos 3D, imágenes, sonidos, etc., a un proyecto de Unity3D, solamente se necesita arrastrarlos desde el explorador o finder, hasta el panel de proyecto, los cuales son automáticamente importados por el motor. También es posible utilizar las opciones que ofrece Unity3D, a través de un menú contextual, presionando el botón derecho del mouse. La Figura 28 ilustra al panel de proyecto:

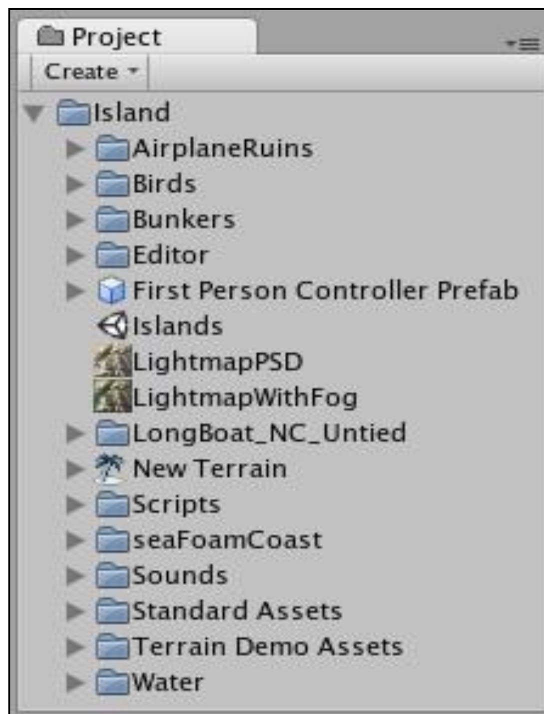



*Figura 28, El Panel de Proyecto*

Nótese que en la Figura 28, existe una opción denominada "Create", la cual permite crear Asset especiales, que solamente se pueden crear con Unity3D. Dentro de ellos están los prefabs, los Script, los materiales, estilos visuales para la GUI, etc.

### 4.8.5 La Ventana de Juego

La ventana de juego, básicamente representa la vista de la escena cuando el videojuego se está ejecutando, es decir, como se vería éste en su publicación final [10]. Esta sirve para probar de forma rápida y sencilla (a través del botón play, dispuesto en la barra superior), el juego en funcionamiento. Además, permite visualizar el juego en distintas resoluciones de pantallas, ya sea pantalla ancha (16:9), cuadrada (4:3), etc. Otra característica importante de la ventana de juego es que, permite visualizar a través del panel de jerarquía y el inspector las propiedades y

Universidad del Bío-Bío. Red de Bibliotecas - Chile



comportamientos de los Game Object en tiempo de ejecución, es más, es posible modificar las propiedades de éstos (temporalmente, ya que vuelven su estado normal cuando se detiene el juego), facilitando las pruebas y experimentos sobre los Game Object [10].

## 4.9 Herramientas Externas

Unity3D provee un conjunto de diversas herramientas y funcionalidades que permiten hacer más fácil el desarrollo de un proyecto de videojuego. Sin Embargo, también es necesario manejar y utilizar otras herramientas externas al motor, que permiten crear de forma más especializada objetos como modelos 3D, imágenes y sonidos.

### 4.9.1 Modelado 3D

Aquí existen diversas opciones, soportadas o compatibles con Unity3D, desde donde se puede elegir, destacando principalmente herramientas como Maya, 3D Studio Max, Cinema 4D, Cheetah 3D y Blender [4]. Todas ellas, excepto Blender, requieren de costosas licencias para su utilización. En general, pueden ser utilizadas herramientas que puedan exportar modelos en los formatos .FBX, .dae, .3ds, .dxf, y .obj, archivos que pueden ser leídos por Unity3D. Al momento de importar los modelos 3D, Unity3D permite modificar algunas características de éstos, dentro de las que destacan el factor de escala, ya que algunos modelos, al agregarlos a una escena, su tamaño, en algunas ocasiones, es demasiado grande o muy pequeño, por lo que es necesario ajustarlo mediante esta opción. Cabe mencionar también que, cuando se importa un modelo, éste queda automáticamente como un prefab.

### 4.9.2 Audio

En el caso de importar recursos de audio, Unity3D soporta dos tipos de archivos de audio [10]: descomprimido y Ogg Vorbis. Así, si se importan otros tipos de archivos, el motor automáticamente los convierte a uno de estos formatos [10]. Ahora bien, como herramienta externa puede ser utilizada la aplicación libre Audacity, para editar y exportar archivos de audio. Dentro de los formatos que se importan, están principalmente los archivos .WAV (útil para efectos de sonido cortos), .MP3 (principalmente para música de fondo), .OGG (también para la música de fondo) y los .AIFF (para efectos de sonido corto).

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# Capítulo V

# DESARROLLO VIDEOJUEGO MULTIPLAYER

En este capítulo se abordará el tema crucial de este proyecto, el cual corresponde a la implementación del videojuego multiplayer online, denominado "Ancient Gale", el cual se enmarca en una historia de fantasía, en donde el personaje principal deberá cumplir con la misión de derrotar a un temible dragón.

Por ende, el objetivo de este capítulo es mostrar las técnicas y procedimientos que se utilizaron en el desarrollo del proyecto, siguiendo la metodología iterativa incremental. Dentro de los contenidos de este capítulo se encuentran:

1. La implementación del control del personaje principal.
2. La implementación de la cámara.
3. El diseño del mundo 3D.
4. Implementación del modo multiplayer.
5. La conexión con una base de datos.
6. La implementación de la comunicación chat.
7. El diseño de la interfaz de usuario.
8. La incorporación de enemigos.

Cabe mencionar que cada una de estas funcionalidades se dividió en tres incrementos o etapas, en donde el primer incremento correspondió a los puntos 1, 2 y 3. Para el segundo, los puntos 4 y 5, y por último el tercer incremento se desarrollaron los puntos 6, 7 y 8. Además, también se realizaron las pruebas de rendimiento, con el fin de determinar las especificaciones mínimas para poder ejecutar el videojuego.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 5.1 Controlando al Personaje

Uno de los aspectos fundamentales en todo videojuego es el control del personaje principal, ya que corresponde a la principal fuente de entrada e interacción del usuario final con el videojuego.

Inicialmente, se debe contar con una figura o modelo 3D, el cual será controlado a través de la entrada de usuario, esto es, el teclado, el mouse o un joystick. La Figura 29, ilustra al modelo 3D correspondiente al personaje principal de Ancient Gale, el cual ha sido tomado de otro videojuego (The Legend of Zelda), debido a lo complejo que significa modelar una figura 3D.

*Figura 29, Modelo 3D personaje principal*

Ahora bien, para implementar el control de un personaje en Unity3D, se debe utilizar un componente denominado `CharacterController`, el cual provee las funciones necesarias para proporcionar a una figura 3D, movimiento y detección de colisiones dentro de una escena [10]. Sin embargo, este componente no funciona por sí solo, por lo que se debe escribir un Script que utilice sus funcionalidades y cree el comportamiento necesario para controlar al personaje. En primer lugar, se debe definir el estilo del control que poseerá el videojuego, y además las acciones y movimientos que el personaje podrá realizar. Específicamente, en Ancient Gale el control del personaje será al estilo de tercera persona, pudiendo moverse en todas direcciones,

Universidad del Bío-Bío. Red de Bibliotecas - Chile



saltar, atacar y realizar otras acciones, como tomar objetos y abrir puertas. En cuanto a la implementación del control, la estructura de componentes que posee el Game Object del personaje se presenta en la Figura 30:

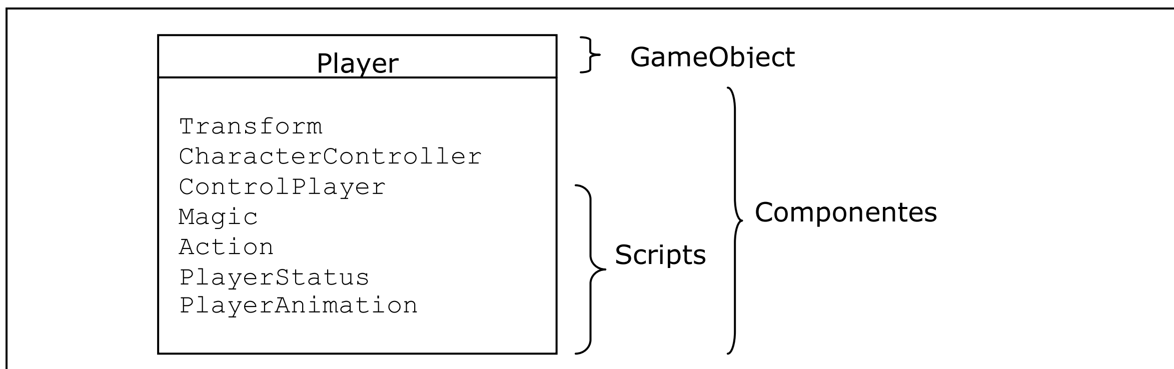



*Figura 30, Estructura de componentes para el control del personaje*

Como se puede apreciar en la Figura 30, el objeto Player está compuesto por varios componentes. El componente `Transform`, como se mencionó en capítulos anteriores, representa la posición, rotación y escala del Game Object en una escena. El componente `ControlPlayer`, el cual corresponde a un Script, es el encargado de capturar la entrada del usuario y dar movimiento al Player, a través de las funciones que provee el componente `CharacterController`. El componente `Magic`, proporciona al Player la habilidad de arrojar magia a sus enemigos. `Action`, por su parte, permite realizar acciones al Player como abrir puertas y tomar ítems. El Script `PlayerStatus`, es el encargado de administrar la vida del personaje, restableciéndolo dentro de la escena si éste muere (pierde toda su vida) producto de los daños que le produjeron los enemigos.

A continuación se presentan las funciones más importantes de cada uno de los Scripts que controlan las acciones del Player. Primeramente, se tiene a `ControlPlayer`, el que principalmente captura la entrada del usuario a través de las funciones que provee clase `Input`:

```
1 Input.GetAxis("Vertical");
2 Input.GetAxis("Mouse X");
3 Input.GetButton("Jump"));
```

*Figura 31, Funciones para capturar la entrada del usuario*

La función `GetAxis`, retorna un valor entre -1 y 1, para la entrada del teclado o de un joystick, devolviendo 0 cuando el usuario no realiza ninguna acción. El `String`

Universidad del Bío-Bío. Red de Bibliotecas - Chile



que es pasado como parámetro, corresponde al nombre de la tecla o botón que es mapeado por Unity3D. Por ejemplo, "`Vertical`" se relaciona con el movimiento hacia adelante o hacia atrás (teclas w, s, arriba, abajo del teclado y el movimiento vertical de un joystick), representando una especie de eje, en este caso vertical. Lo mismo sucede con "`Mouse X`", el cual se utiliza para mapear el movimiento horizontal del mouse. Esta función se utiliza principalmente para realizar movimientos dentro de una escena. La función `GetButton`, por su parte devuelve verdadero o falso dependiendo si la tecla, mapeada por el `String` que se le pasa como parámetro, es presionada por el usuario. Esta función es útil para realizar acciones como saltar, atacar, etc. Cabe mencionar que Unity3D, permite configurar completamente la entrada del usuario, pudiendo los desarrolladores mapear a gusto las teclas y botones a utilizar en el juego. Ahora bien, una vez capturada la acción del usuario, se debe realizar el movimiento del personaje a través de la función `Move` que provee la clase `CharacterController`:

```
1  moveDirection = new Vector3(Input.GetAxis("Horinzontal"), 0,
2                             Input.GetAxis("Vertical"));
3  moveDirection = transform.TransformDirection(moveDirection);
4  moveDirection *= speed;
5  if(Input.GetButton("Jump")){
6     moveDirection.y = jumpSpeed;
7  }
8  moveDirection.y -= gravity * Time.deltaTime;
9 var controller:CharacterController=GetComponent(CharacterController);
10 controller.Move(moveDirection * Time.deltaTime);
```

*Figura 32, Función para mover al personaje*

Al observar el código de la Figura 32, en primer lugar se define el vector `moveDirection`, el cual representa el movimiento dentro del espacio 3D, a través de sus tres componentes x, y, z. El componente x, representa el movimiento horizontal de izquierda o derecha. Por otro lado, el componente y, representa el movimiento vertical (por ejemplo, de salto), mientras que el componente z, representa el movimiento hacia adelante o hacia atrás. El tipo de movimiento dependerá del signo de los valores de cada componente, el cual viene dado por el valor que retorna la función `GetAxis`. Luego, se debe convertir el vector de movimiento desde el espacio de coordenadas local (Local Space) del personaje al espacio de coordenadas del mundo (World Space), mediante la función `TransformDirection` (línea 3) de la clase `Transform`. En seguida, se multiplica el vector de movimiento por un escalar, con el fin de definir la velocidad (`speed`, línea 4) del movimiento. Para realizar la acción de salto, se debe asignar la velocidad de salto (`jumpSpeed`, línea 5) al componente y del vector de movimiento. Por

Universidad del Bío-Bío. Red de Bibliotecas - Chile



último, se aplica el movimiento al personaje a través de la función Move (línea 10), correspondiente a la clase CharacterController, la cual realiza las translaciones necesarias al personaje en base al vector de movimiento, el cual es multiplicado por un valor delta (deltaTime), con el fin de definir el movimiento en unidades de distancia por unidad de tiempo (por ejemplo, metros por segundo) [11]. Cabe mencionar que esta función, solamente realiza el movimiento si el personaje no colisiona con otro objeto, es decir si el personaje choca o colisiona con una pared, no podrá avanzar. Además, como se aprecia en la línea 8 del código de la Figura 32, es necesario aplicar gravedad al personaje para que cuando éste salte, no permanezca flotando en el aire. Simular la gravedad, se logra mediante la substracción continúa del componente 'y' del vector de movimiento por un valor definido previamente (también en unidades de distancia por tiempo).

Con respecto al Script Magic, éste tiene como objetivo proporcionar al personaje la habilidad de arrojar magia a sus enemigos. La Figura 33, muestra el código de las funciones más importantes que utiliza éste componente:

```
1 var magic = Instantiate(magicPrefab, transform.position,
2                 transform.rotation);
3 magic.rigidbody.AddForce(transform.forward * speed);
```

*Figura 33, Habilidad de arrojar magia*

Como se aprecia en la Figura 33, en primer lugar, se instancia el objeto *magicPrefab* (una referencia a un Prefab) dentro de la escena, a través de la función Instantiate (heredada de la clase Object), la cual crea un clon del objeto original, con una posición y rotación específica. Notar que cuando se define "transform.position", se está refiriendo a la posición del personaje, ya que la variable transform representa el componente Transform del Game Object al que pertenece el Script. La segunda sentencia, tiene como fin aplicar movimiento al objeto recién creado, a través de la función AddForce del componente rigidbody (de la clase RigidBody). Observar que a ésta función se le pasa como parámetro un vector al cual se le aplica una velocidad. La sentencia transform.forward representa el vector que se dirige hacia adelante del personaje, por lo que el objeto instanciado se moverá en esa dirección cuando el usuario presione el botón relacionado con la acción de arrojar magia.

Por otro lado, el Script Action se encarga de realizar acciones relacionadas con la lógica del juego, esto es, permitir que cuando el personaje cumpla con un objetivo

Universidad del Bío-Bío. Red de Bibliotecas - Chile



específico, pueda realizar otras acciones. Por ejemplo, cuando el personaje encuentre la espada que está escondida en la aldea, podrá abrir la puerta con la cual se accede al bosque. A continuación, se muestra las funciones más importantes de este Script:

```
1   function OnTriggerEnter(other : Collider){
2      if(other.gameObject.tag == "Espada"){
3            getSword = true;
4            ...
5      }
6      if(other.gameObject == "Reja" && getSword && !isOpen){
7            isOpen = true;
8            other.gameObject.animation.Play("doorAnim");
9      }
10 }
```

*Figura 34, Obtener espada y abrir puerta*

El código de la Figura 34, consiste en la función (resumida) que permite al jugador obtener la espada y abrir la puerta o reja del bosque. Esto realiza mediante la función `OnEnterTrigger`, la cual se invoca automáticamente cuando el personaje ingresa (o colisiona) a una zona o área específica, perteneciente a otro objeto. La variable `other`, que recibe como parámetro esta función, corresponde al componente `collider` del objeto con que se colisionó. A través de `collider`, se puede acceder a cada una de las variables y componentes del otro objeto (aunque algunas de solo lectura). Esto queda en evidencia cuando se utiliza la variable `tag` de `other`, con el fin de verificar si el objeto con que se colisionó es, en este caso, la "`Espada`". Lo mismo sucede con la "`Reja`", pero además se consulta si antes había obtenido la espada (`getSword`), por lo que de ser así, se abre la puerta a través de la función `Play` del componente `animation` perteneciente al Game Object con que se colisionó. La función `Play`, ejecuta la animación que se especifica por parámetro (previamente creada e incorporada en el modelo 3D).

Por su parte, el Script `PlayerStatus` se encarga de administrar la vida del personaje, aplicando daños a éste cuando es golpeado por un enemigo. Además, restablece al personaje cuando muere. En resumen, las funciones más importantes de éste Script se presentan a continuación:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
1  function applyDamage(damage : int){
2     ...
3     life -= damage;
4     if(life <= 0){
5           life = 0;
6           isDead = true;
7     }
8 }
9
10 function respawnPlayer(){
11    ...
12    var indice = Random.Range(0, spawn.puntos.length);
13    isDead = false;
14    life = maxLife;
15    transform.position = spawn.puntos[indice].position;
16    transform.rotation = spawn.puntos[indice].rotation;
17 }
```

*Figura 35, La Vida del personaje*

Como se observa en la Figura 35, la función `applyDamage` se utiliza para restarle vida (`life`) al personaje, estableciendo a éste como muerto (`isDead = true`) si la pierde por completo. Cuando el jugador muere, se llama a la función `respawnPlayer`, la cual traslada al personaje a una posición al azar (dentro de un conjunto de puntos), restableciendo además su estado (`isDead = false`) y vida (`life = maxLife`).

Por último, el Script `PlayerAnimation`, tiene como objetivo reproducir las animaciones del personaje (incorporadas en el modelo 3D), según la acción que realice, esto es, cuando corre, arroja magia, toma un objeto, entre otras. Por lo tanto, este Script también debe capturar la entrada del usuario y utilizar los Script mencionados anteriormente, con el objeto de conocer el estado del personaje. A continuación se presenta un trozo de código que ilustra las acciones que realiza `PlayerAnimation`:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
1  function Update(){
2      ...
3      if(Input.GetAxis("Vertical") > 0.1){
4            animation.CrossFade("run");
5      }
6      ...
7      if(playerControl.isJump()){
8            animation.CrossFade("jump");
9      }
10     ...
11 }
```

*Figura 36, Animaciones del personaje*

Al mirar el código de la Figura 36, cada una de las animaciones del personaje se ejecuta o reproduce dentro de la función `Update`, la cual es llamada cíclicamente por Unity3D, por lo que permite actualizar y conocer el estado del personaje constantemente. Como se mencionó anteriormente, algunas de las animaciones dependen de la acción que realice el usuario, por lo que se debe controlar la entrada. Esto queda de manifiesto cuando se consulta, a través de la función `GetAxis`, si se está presionando la tecla para ir hacia adelante (valor de retorno mayor que 0). De ser así, se reproduce la animación "`run`" *(*correr*)*, a través de la función `CrossFade` del componente `animation` (perteneciente a la Clase `Animation`) adjunto al personaje. La función `CrossFade` reproduce la animación especificada como parámetro, pero se diferencia de la función `Play`, en que realiza un cambio más suave de una animación a otra, por lo que se visualiza un movimiento más natural. Nótese que, también este Script consulta a otros (`ControlPlayer`*,* específicamente), si el personaje está realizando una acción, como por ejemplo saltar (`playerControl.isJump`, en la línea 7).

## 5.2 Una Mirada 3D, La Cámara

La implementación de la cámara está muy relacionada con el control del personaje, ya que ésta debe seguir todos los movimientos y acciones que éste realice. En general, Unity3D provee los Script necesarios para manejar la cámara, por lo cual no es necesario implementar desde el principio su funcionalidad, solamente se debe agregar los comportamientos específicos que tendrá en el videojuego a desarrollar. Ahora bien, el primer paso para incorporar una cámara dentro de una escena es crear un objeto `Camera`, el cual es provisto por Unity3D. Luego, se debe añadir a la cámara creada, los Scripts necesarios para crear el comportamiento que se requiera.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



Específicamente, en Ancient Gale, como se señaló anteriormente, la cámara posee el estilo de tercera persona, la cual enfoca a una distancia y altura específica al personaje, pudiendo acercar y alejar la cámara en base a la entrada del usuario. Ahora bien, lis Scripts utilizados para manejar la cámara son dos, uno es `SmoothFollow` y `ControlPlayer` (componente del personaje). El Script `SmoothFollow` se encarga de rotar y trasladar la posición de la cámara en base a los movimientos del personaje. Esto se logra mediante una referencia al componente `transform` del personaje, el cual se denomina `target`. Además, se utilizan una serie de funciones matemáticas (de la clase `Mathf`), para realizar una interpolación (obtener valores intermedios, entre un mínimo y un máximo) de los atributos de la cámara, esto es, su distancia y altura con respecto al personaje. A continuación, se muestra un trozo de código de las funciones más importantes del Script `SmoothFollow`:

```
1   currentRotationAngle = Mathf.LerpAngle(currentRotationAngle,
2            wantedRotationAngle, rotationDamping * Time.deltaTime);
3   currentHeight = Mathf.Lerp(currentHeight, wantedHeight,
4                         heightDamping * Time.deltaTime);
5   currentRotation = Quaternion.Euler(0, currentRotationAngle, 0);
6   transform.position = target.position;
7   transform.position -= currentRotation * Vector3.forward * distance;
8   transform.position.y = currentHeight;
9   transform.LookAt(target);
```

*Figura 37, Movimiento de la cámara*

En primer lugar, se calcula la interpolación de los ángulos de rotación de la cámara (`currentRotationAngle`) con respecto al de personaje (`wantedRotationAngle`), a través de la función `LerpAngle` (toma valores en grados). Lo mismo sucede con la altura, donde se calcula la interpolación entre la altura actual de la cámara (`currentHeight`) con respecto a la del personaje (`wantedHeight`), pero usando esta vez la función `Lerp` (toma valores flotantes). Luego, se transforma la rotación calculada a un `Quaternion` (forma en que representa Unity3D todas las rotaciones), mediante la función `Euler` de la clase `Quaternion`. En seguida, se posiciona la cámara a la distancia y altura calculada anteriormente (líneas 6 a la 8), siempre con respecto al personaje. Finalmente, se utiliza la función `LookAt` del componente `transform`, la cual rota la cámara (en torno al eje Y), de tal forma que el vector que representa su vista hacia adelante (`forward`), queda "mirando" hacia el personaje (`target`).

Universidad del Bío-Bío. Red de Bibliotecas - Chile



Por otro lado, como se mencionó anteriormente, el Script `ControlPlayer` utiliza los atributos de distancia y altura de la cámara, con el fin de modificarlos en base a la entrada del usuario. La Figura 38 ilustra el código que manipula los atributos en cuestión:

```
1 if(Input.GetAxis("Mouse Y") > 0.1){
2     camaraTransform.gameObject.GetComponent(SmoothFollow).height +=
3           Time.deltaTime * 5.0;
4     camaraTransform.gameObject.GetComponent(SmoothFollow).distance +=
5           Time.deltaTime * 2.0;
6 }
```

*Figura 38, Manipulación de la distancia y altura de la cámara*

Como se aprecia en el código de la Figura 38, en primer lugar se captura la entrada del usuario, específicamente si el mouse se movió hacia arriba, por lo que se aumenta la altura (`height`) y distancia (`distance`) de la cámara (por un valor en unidades de distancia y tiempo). Nótese, que estos atributos son accedidos mediante una referencia al componente `transform` de la cámara, el cual permite acceder a la función `GetComponent` de la clase `GameObject`, la cual permite obtener un componente añadido a un objeto, en este caso, el Script `SmoothFollow`.

## 5.3 Luz, Cámara, Acción..., el Mundo 3D

En la siguiente sección se aborda el diseño y la construcción del mundo de fantasía de Ancient Gale, el cual está compuesto de 3 sectores. El primero consta de una aldea, en donde comienza la aventura. Aquí, el jugador tendrá como misión encontrar la espada legendaria, oculta en algún lugar dentro de la aldea, con la cual tendrá que hacer frente a sus enemigos. Una vez obtenido este objeto, podrá avanzar a la segunda etapa, correspondiente al gran bosque, en donde tendrá que combatir con enemigos y hallar el fuego sagrado, con el cual obtendrá la habilidad de arrojar magia. Una vez logrado este objetivo, podrá entrar al castillo donde se encuentra el dragón y en donde deberá combatir con él (más detalles en Anexo A). A continuación, en la Figura 39, se presenta un bosquejo del mapa que representa el mundo de Ancient Gale:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



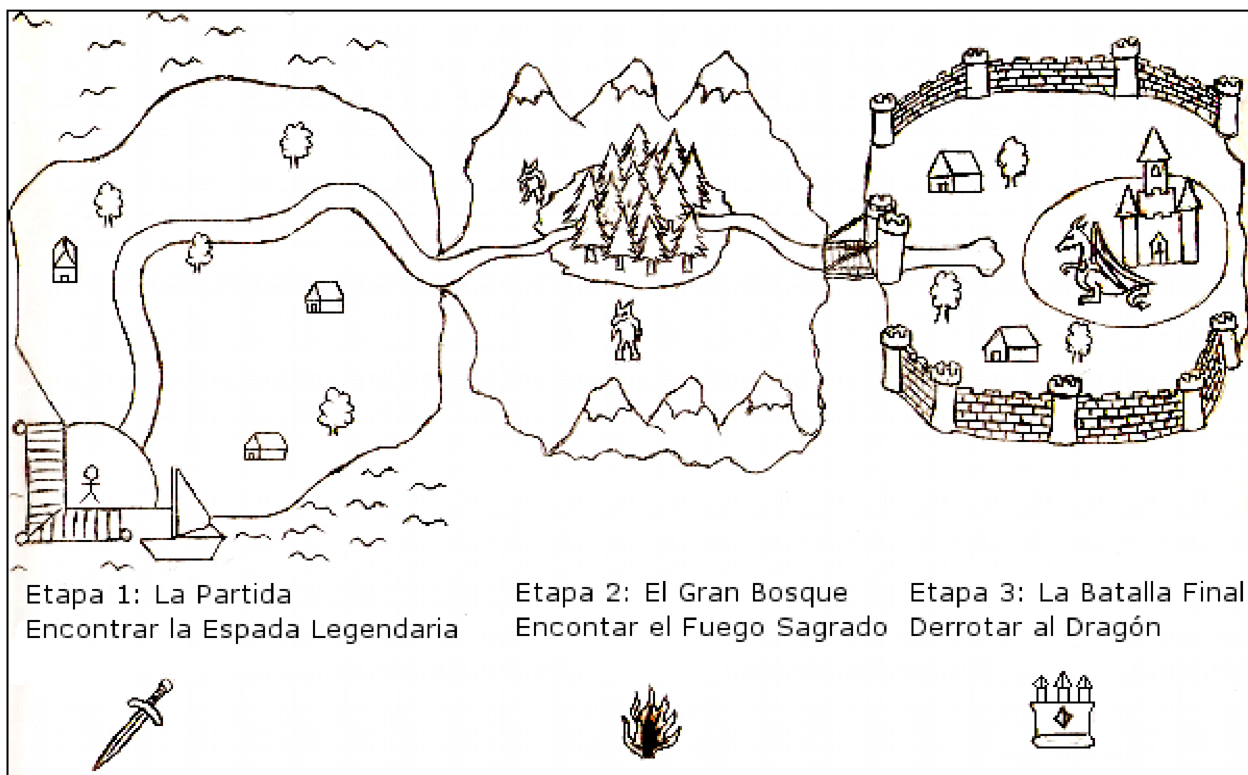



*Figura 39, Bosquejo del mapa de Ancient Gale*

Ahora bien, el primer paso para modelar el mundo 3D, es crear un terreno o Terrain. Unity3D incorpora un completo motor de terrenos, el cual provee todas las herramientas necesarias para diseñar y construir vastos y complejos terrenos. Un terreno consiste en un plano que posee diversas características, entre la cuales están principalmente, un largo, ancho, un alto (altura máxima del terreno), todas éstas medidas en metros. Ahora bien, el motor de terrenos de Unity3D provee herramientas para esculpir montañas y cuencas, pintar terrenos mediante pinceles que utilizan texturas definidas por los diseñadores, añadir eficientemente árboles y vegetación al mundo 3D mediantes modelos y figuras planas. En fin, un completo set de herramientas que facilitan el modelado de cualquier terreno, solamente limitado por la imaginación de los diseñadores.

Una vez diseñado el mundo 3D, se puede comenzar a agregar adornos y efectos especiales, tales como casas, luces, el cielo, etc. En el caso de los modelos 3D, éstos deben ser importados desde herramientas externas de modelado. Particularmente, en la realización de este proyecto, se utilizaron principalmente modelos provenientes desde Google SketchUp (modelos .fbx), Maya y 3D Studio Max. Para el caso de las luces, Unity3D provee los Game Objects y componentes necesarios para crearlas. Solo vasta definir de qué tipo es la luz (direccional, un punto o foco), su color, intensidad y

Universidad del Bío-Bío. Red de Bibliotecas - Chile



rango, principalmente. La Figura 40 muestra las propiedades que se pueden definir a una luz en Unity3D:

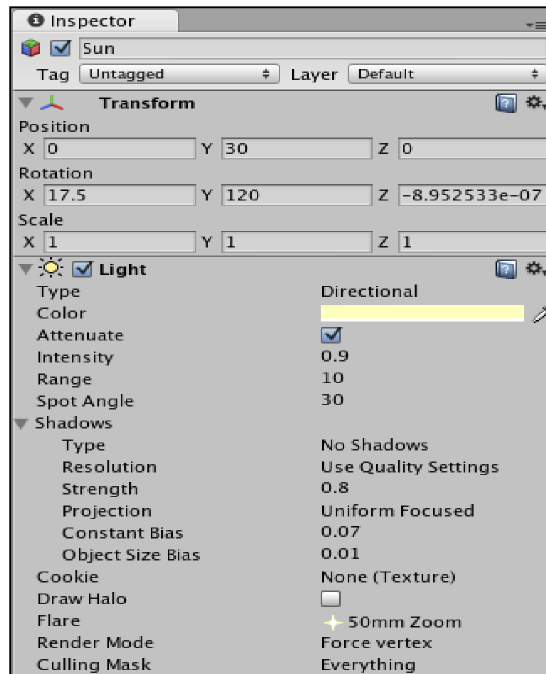



*Figura 40, Configuración de luces en Unity3D*

Esta configuración corresponde a una luz tipo direccional, utilizada para simular el sol que alumbra la escena completa. Nótese, además que también es posible agregar el efecto de la luz cuando se refleja en el lente de la cámara (Flare). En cuanto a los efectos visuales, como el fuego o el humo, Unity3D provee un sistema de partícula. También viene dado como un componente, por lo que solo se debe modificar sus propiedades específicas para crear el efecto deseado. Dentro de las propiedades que se pueden definir están el color, el tamaño de la partícula, la textura, el movimiento (animación), la cantidad de partículas, la velocidad, entre muchos otros atributos. Para ilustrar más gráficamente el diseño final del mundo de Ancient Gale, se presenta en la Figura 41, algunas imágenes de sus principales niveles:

Universidad del Bío-Bío. Red de Bibliotecas - Chile

Figura 41, Diseño del Mundo de Ancient Gale

## 5.4 A través de la Red, el Modo Multiplayer

En la siguiente sección se presenta la implementación de una de las funcionalidades más importantes del proyecto, esto es, el modo multiplayer (o multi-jugador). Este modo de juego consiste en que varios jugadores, por medio de una red de computadores, puedan jugar simultáneamente en la misma escena. La Figura 42 muestra el modo de interconexión entre los jugadores, basado en el modelo Cliente/Servidor:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



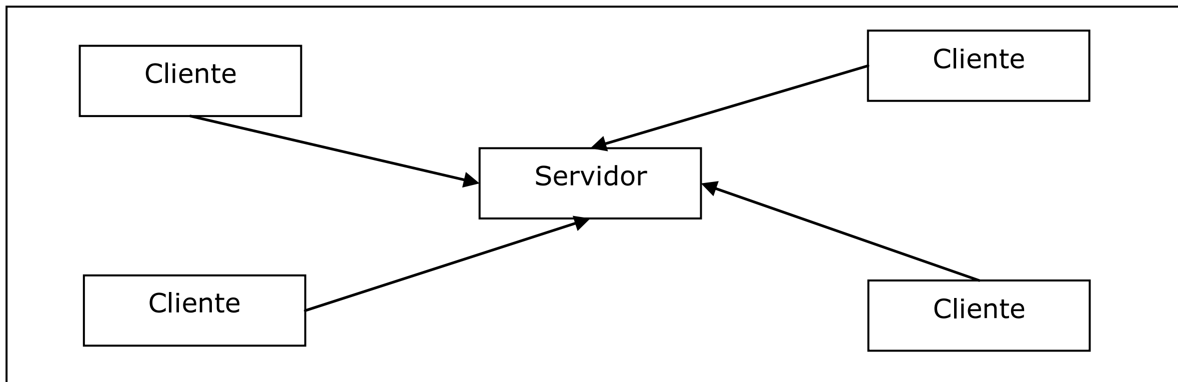



*Figura 42, Modelo de interconexión entre clientes y servidores*

Como se puede observar en la Figura 42, el modo de interconexión de Ancient Gale consiste en que un jugador se establece como servidor, al cual se podrán conectar uno o más clientes (otros jugadores). Cabe mencionar que, cualquier jugador puede hacer las veces de servidor o de cliente. Ahora bien, Unity3D incorpora soporte para desarrollar y construir videojuegos en línea, por medio de componentes y funcionalidades que permiten, a un alto nivel, crear conexiones entre clientes y servidores.

En cuanto a la implementación del modo multiplayer, se debe comenzar por la creación y establecimiento de las conexiones entre los diferentes jugadores. Para ello, se debe hacer uso de la clase `Network`, la cual provee un conjunto de funciones para tal propósito. Específicamente, el script `Conectar`, perteneciente a la cámara, es el encargado de crear y conectar a los servidores y clientes. A continuación se presentan las principales funciones de este script:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
1  if(GUILayout.Button("Conectar", GUILayout.Width(115),
2          GUILayout.Height(25))){
3     ...
4     Network.Connect(IP, puerto);
5  }
6  ...
7  if(GUILayout.Button("Crear Servidor", GUILayout.Width(115),
8          GUILayout.Height(25))){
9     ...
10    Network.InitializeServer(numPlayer, puerto);
11 }
12 ...
13 if(Network.isClient || Network.isServer){
14    ipaddress = Network.player.ipAddress;
15    port = Network.player.port.ToString();
16    if(Network.isClient){
17          GUI.DrawTexture(Rect(0, 0, 450, 110), faceTexture);
18          GUI.Label(Rect(130,0, 500, 20), "Nickname: "+nickName);
19          GUI.Label(Rect(130,12, 500, 20), "Status: Cliente");
20          GUI.Label(Rect(130,24, 500, 20), "IP:"+ipaddress+":"+port);
21    }else{
22          if(Network.isServer){
23                GUI.DrawTexture(Rect(0, 0, 450, 110), faceTexture);
24                GUI.Label(Rect(130,0, 500, 20), "Nickname:
25                            "+nickName);
26                GUI.Label(Rect(130,12, 500, 20), "Status: Servidor");
27                GUI.Label(Rect(130,24, 500, 20), "Conectados:
28                            "+Network.connections.length);
29                GUI.Label(Rect(130,36, 500, 20), "IP:
30                            "+ipaddress+":"+port);
31          }
32    }
33    if(Input.GetKeyDown(KeyCode.Escape) && !isEscape){
34          isEscape = !isEscape;
35    }
36    if(isEscape){
37          Screen.showCursor = true;
38          GUILayout.Window(7,Rect(Screen.width*0.35,
39                      Screen.height*0.45, 240, 100), windowEscape,
40                      "¿Salir del Juego?");
41    }
42 }
```

*Figura 43, El Script Conectar*

Al observar el código de la Figura 43, se tiene que existen dos acciones relevantes (representadas en botones) con respecto al manejo de las conexiones entre jugadores, las que corresponden a "`Conectar`" y "`Crear Servidor`". La acción "`Crear Servidor`" (línea 7), inicializa o crea un servidor, mediante la función estática `InitializeServer` de la clase `Network`. Ésta función recibe como parámetro el número máximo de jugadores o clientes (`numPlayer`) que se pueden conectar al servidor y el puerto que escuchará las conexiones de los clientes. Por otro lado, la acción

Universidad del Bío-Bío. Red de Bibliotecas - Chile



"Conectar" (línea 1), conecta un cliente a un servidor mediante la función estática Connect, de la clase Network, la cual recibe como parámetro la dirección IP y el puerto del servidor al cual se desea conectar. Adicionalmente, en la línea 33, se tiene la acción "Desconectar", la cual cierra cualquier conexión establecida, ya sea como cliente o como servidor. Para ello, cuando el jugador presiona la tecla escape, se despliega por pantalla una pequeña ventana (línea 38), que entrega la opción de desconectarse del juego. Notar que, cuando se invoca la función Window (de la clase GUILayout), se le pasa como parámetro "windowEscape", que cual corresponde al nombre de la función que crea la ventana. Ésta función (windowEscape), básicamente hace uso de la función estática Disconnect (de la clase Network), para desconectar al jugador del juego. Cabe mencionar que, si se desconecta un servidor, todos los clientes que posean una conexión con éste, también se desconectarán. Notar, que la clase Network permite identificar si un jugador es un cliente o un servidor, a través de las variables de clase isServer (línea 22) o isClient (línea 16), así como también obtener la dirección IP, el puerto del jugador (línea 14 y 15), ambas por medio de la variable player (de la clase NetworkPlayer) y la cantidad de conexiones en el servidor (Network.connections.length, línea 28), todo esto con el fin de conocer el estado de la conexión, tanto en el servidor como en los clientes.

Una vez establecidas las conexiones, se debe comenzar a intercambiar datos entre los diferentes jugadores. Para ello, se debe utilizar el componente NetworkView, el cual permite a un Game Object u otro componente, comunicar datos a través de una red [10]. Por lo tanto, cada objeto o componente que necesite enviar y recibir datos en una red, deberá poseer el componente NetworkView. Ahora bien, considerando que el ancho de banda de una red es relativamente limitado, no es necesario que todos los objetos y componentes dentro de una escena comuniquen datos en la red, solamente los más relevantes como por ejemplo el personaje principal, los enemigos y algunos efectos visuales. De hecho, no se envían completamente cada uno de estos Game Object, sino solo algunas de sus propiedades como son su posición, su rotación, un estado particular, en fin datos simples, como valores numéricos, vectores (x, y, z), cadenas de caracteres, etc.

Dentro de las propiedades del componente NetworkView están "State Synchronization" y "Observed". La primera propiedad indica el tipo de sincronización que se llevará a cabo a través de la red, pudiendo seleccionar entre tres posibles estados: Reliable Delta Compressed, Unreliable y Off [10]. Reliable Delta

Universidad del Bío-Bío. Red de Bibliotecas - Chile



`Compressed` consiste en enviar datos a la red, solamente cuando se ha producido un cambio en el estado anterior de los datos, por ejemplo cuando el personaje cambia de posición (se traslada). En cambio `Unreliable`, envía datos por la red sin considerar si éstos han cambiado desde el estado anterior. En el caso de `Off`, no se realiza ninguna de las acciones mencionadas anteriormente, es decir, el `NetworkView` no sincroniza automáticamente, a través de la red, el estado del Game Object al cual pertenece. Por otro lado, la propiedad `Observed` consiste en el componente de un Game Object que será sincronizado por el `NetworkView`, que por lo general es `Transform`. Si bien, `NetworkView` permite automatizar, entre las diferentes conexiones, la sincronización de la transformación de un objeto (posición, rotación y escala), en ocasiones se necesita sincronizar otros estados o variables que `NetworkView` no sincroniza automáticamente, por ejemplo mostrar objetos que el personaje ha obtenido, la animación actual que está ejecutando el personaje, entre otros. Para ello, se debe hacer uso de una característica especial que ofrece el componente `NetworkView`, la cual se denomina `RPC` (Remote Procedure Calls, o Llamadas a Procedimientos Remotos, en español), que permite invocar funciones remotamente desde un jugador a otro, por medio de la red. A continuación, en la Figura 44, se presenta una de las tantas sincronizaciones que se debieron implementar en Ancient Gale, la cual corresponde a la animación de ataque del personaje principal:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
1  function Update(){
2     ...
3     if(getSword && Input.GetButton ("Fire1")
4                 && (lastTime+attackTimer)<Time.time){
5           var anim = GetComponentInChildren(Animation);
6           lastTime = Time.time;
7           anim.CrossFade("attack");
8           networkView.RPC("SetAnimation", RPCMode.Others, "attack");
9           isAttack = true;
10    }
11    ...
12 }
13
14 @RPC
15 function SetAnimation(anim : String){
16    var animacion = GetComponentInChildren(Animation);
17    if(anim == "jump"){
18          animacion.CrossFade(anim, 0.05);
19    }else{
20          animacion.CrossFade(anim);
21    }
22 }
```

*Figura 44, Llamadas a Procedimientos Remotos, RPC*

Como se puede apreciar en el código de la Figura 44, en primer lugar se verifica dentro de la función `Update` (del Script `ControlPlayer`) si se está realizando la acción de ataque, esto es, si el jugador obtuvo la espada (`getSword`), además si presionó el botón de ataque y si el tiempo de ataque ya se cumplió (para que el personaje no ataque continuamente). De cumplirse todas estas sentencias, el jugador podrá atacar, por lo que se obtiene el componente `Animation` (ubicado en uno de los hijos Game Object actual, específicamente el modelo 3D del personaje), a través de la función `GetComponentInChildren`. Una vez obtenido el componente (`anim`, línea 5), se ejecuta localmente (en la máquina del jugador) la animación de ataque. Es por esto que luego se debe realizar una llamada a un procedimiento remoto para ejecutar esta animación en la máquina de los otros jugadores (en la copia del jugador local que reside en los demás jugadores de la red). Esta llamada se realiza a través de la función `RPC` (línea 8) del componente `networkView`, añadido al Game Object que pertenece el Script, pasando como parámetro el nombre de la función que se desea invocar ("`SetAnimation`"), el modo de llamada y los parámetros de la función remota (el nombre de la animación, "`attack`"). El modo de llamada indica a quiénes se envía la llamada del procedimiento remoto, pudiendo enviar a todos los jugadores (incluido el jugador local), solamente a los otros jugadores (excluyendo al jugador local) o únicamente al servidor. En el caso de la Figura 44, se envía la llamada a los otros

Universidad del Bío-Bío. Red de Bibliotecas - Chile



jugadores (RPCMode.Others). Una vez realizada la invocación del procedimiento remoto, éste se ejecuta en cada una de las máquinas de los otros jugadores. Básicamente, la función SetAnimation reproduce la animación indicada como parámetro, en la copia (residente en la máquina remota) del personaje que envió la llamada. Cabe destacar que, para que una función sea reconocida por Unity como un procedimiento remoto, se debe colocar el prefijo '@RPC', como se muestra en la línea 14 del código de la Figura 44.

Otra de las consideraciones que se debió tomar en cuenta a la hora de implementar el modo multiplayer, es que en algunos casos, ciertos bloques de código, solamente deben ser ejecutados por el jugador local y no por las copias de los jugadores remotos. Por ejemplo, cuando el jugador local arroja magia, solamente él debe ejecutar esta acción y no los demás jugadores dentro de la escena. Para ello, se debe hacer uso del atributo isMine (booleano) de la clase NetworkView, la cual indica si el componente NetworkView pertenece a quien está ejecutando el código. El atributo isMine devuelve un valor verdadero cuando el que está ejecutando el código es el creador del componente NetworkView (el jugador local), de lo contrario entrega falso cuando una copia de un jugador remoto (presente en la escena del jugador local) está ejecutando el código. A continuación, en la Figura 45, se presenta un trozo de código que muestra la funcionalidad de este atributo:

```
1 function Update () {
2     if(networkView.isMine){
3           //realiza acción porque es el jugador local
4           ...
5     }else{
6           //no realiza acción ya que es un jugador remoto
7     }
8 }
```

*Figura 45, Jugador local y remoto*

Como se muestra en el código de la Figura 45, la funcionalidad del atributo isMine, es útil para decidir si una acción se debe realizar en el jugador local o en uno remoto. En general, este tipo de decisiones deben tomarse cuando está involucrada la entrada del usuario, por lo que casi siempre este código se ejecuta en el jugador local (un jugador local no debería controlar a uno remoto).

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 5.5 Establecer la Conexión a la Base de Datos

En la siguiente sección se presenta la implementación de la funcionalidad que permite almacenar en una base de datos los puntajes obtenidos por los jugadores dentro del juego. Para ello, se deben utilizar tecnologías externas a Unity3D, como son MySql (como motor de base de datos) y PHP (como herramienta para acceder a la base de datos). Ahora bien, cada jugador dentro del videojuego, podrá obtener puntaje al luchar contra los enemigos y también al encontrar ítems especiales (tesoros). Éste puntaje será almacenado automáticamente (junto con el nickname del jugador) en una base de datos cuando el jugador salga del juego, con el fin de crear un ranking con los mejores 10 puntajes. En cuanto a la implementación, básicamente la base de datos está compuesta por una tabla, denominada *score*, la cual se presenta a continuación, en la Figura 46:

| scores |
| --- |
| id<br>nickname<br>score |

*Figura 46, Tabla scores de la base de datos*

Cabe mencionar que la tabla *score* está alojada en un servidor central, a cual todos los jugadores se conectan automáticamente. Por otro lado, también en éste servidor central, existen tres archivos escritos en el lenguaje PHP, los cuales cumplen la misión de establecer la conexión y realizar las consultas a la base de datos. Además, se encargan de enviar a los jugadores el ranking con los mejores 10 puntajes. En cuanto al trabajo que se realiza dentro de Unity3D, se encuentra el script *Conectar* como el responsable de comunicarse con los archivos PHP alojados en el servidor central. En resumen, éste script posee 3 funciones, las cuales se encargan de enviar y solicitar los datos al servidor central. A continuación, en la Figura 47, se presenta un resumen de cada una de éstas funciones:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
1   function getNicknames(){
2      puntajes = "Cargando Puntajes...";
3      hs_get = WWW(highscoreUrl);
4      yield hs_get;
5      if(hs_get.error){
6            puntajes = "Se produjo un error al obtener los puntajes:
7                  "+hs_get.error;
8      }else{
9            puntajes = "\tNickname"+"\n\n";
10           puntajes += hs_get.data;
11     }
12   }
13
14 function getPuntajes(){
15     score = "";
16     hs_get = WWW(highscoreUrl2);
17     yield hs_get;
18     if(hs_get.error){
19           score = "Se produjo un error al obtener los puntajes: "
20                 +hs_get.error;
21     }else{
22           score = "Puntaje"+"\n\n";
23           score += hs_get.data;
24     }
25 }
26
27 function guardarPuntaje(nickname, score){
28     var hash = Md5.Md5Sum(nickname + score +secretKey);
29     var highscore_url = addScoreUrl + "nickname="
30           +WWW.EscapeURL(nickname)+"&score="+score+"&hash="+hash;
31     hs_post = WWW(highscore_url);
32     yield hs_post;
33     if(hs_post){
34           print("Se produjo un error al guardar los puntajes:"
35                 +hs_post.error);
36     }
37 }
```

*Figura 47, Funciones para enviar y solicitar los puntajes de los jugadores*

Como se puede apreciar en el código de la Figura 47, en primer lugar se tiene a la función `getNicknames`, la cual es la encargada de solicitar los nicknames de los 10 mejores puntajes al servidor central, a través de la función `WWW` (línea 3). Ésta función, recibe como parámetro una dirección url (*uniform resource locators,* o localizador de recursos uniforme en español), la cual se utiliza para enviar la petición al archivo PHP correspondiente, alojado en el servidor central. La función `WWW` toma la url y comienza a descargar automáticamente, desde la dirección indicada, la respuesta del servidor, por lo cual se debe esperar, mediante la instrucción `yield`, que el flujo completo de datos se obtenga. Luego que se obtiene una respuesta, se debe verificar si se produjo un error durante la descarga de la petición, a través del atributo `error`, de lo contrario

Universidad del Bío-Bío. Red de Bibliotecas - Chile



se obtienen los datos (`data`) que entrega el servidor (en forma de string), los cuales serán mostrados por pantalla a través de la variable *puntajes*. De la misma manera opera la función `getPuntajes` (línea 14), con la diferencia que ésta obtiene los puntajes de los 10 mejores jugadores. Notar que la petición del nickname y el puntaje de un jugador se realizan por separado. Esto con el fin de dar un formato más ordenado a los datos obtenidos. Por otro lado, se tiene la función `guardarPuntaje` (línea 27), la cual recibe como parámetros el nickname y el puntaje (`score`) del jugador que se desea guardar en la base de datos. Nótese que la primera instrucción de ésta función es una llamada a la función `Md5Sum` del script `Md5` (provisto por la comunidad de Unity3D), la cual se encarga de encriptar los datos (`nickname` *y* `score`) a través de una clave secreta (`secretKey`), con el fin de dar más seguridad a los datos que serán enviados. Cabe mencionar que ésta clave secreta también está presente en el servidor central, por lo que cuando se recibe la petición en éste, se verifica (también a través del método de encriptación MD5) la veracidad e integridad de los datos. Luego, al igual que los otros procedimientos, se envía la petición (Post), a través de la función `WWW`, pasándole como parámetro la url que direcciona al archivo PHP correspondiente, el cual almacena los datos enviados en la base de datos.

Finalmente, cabe mencionar que cualquier usuario puede visualizar el ranking con los mejores 10 puntajes, esto a través del menú principal de Ancient Gale.

## 5.6 Comunicación Chat

Otra de las funcionalidades definidas para implementar en este proyecto fue la comunicación tipo chat, con el fin de complementar el modo multiplayer, permitiendo que los jugadores se envíen mensajes entre sí. En primer lugar, se debe definir la estructura de los mensajes. En Ancient Gale, los mensajes son del tipo: nombre del jugador (nickname) que envió el mensaje y el mensaje en sí. En cuanto a la implementación, esta funcionalidad queda en manos del script `Chat`. Básicamente, este script funciona en base a llamadas a procedimientos remotos que envían los mensajes de un jugador a todos los otros que se encuentran conectados. Este script, posee principalmente dos atributos, primero el nickname del jugador y un arreglo que almacena los mensajes de todos los jugadores. A continuación se presentan las funciones más importantes del script `Chat`:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
1  class Mensaje{
2     var nombre : String= "";
3     var msg : String= "";
4  }
5  ...
6  if(Event.current.type == EventType.keyDown
7     && Event.current.character == "\n" && inputField.Length > 0){
8     sendMsg(inputField);
9 }
10 ...
11 for(var msg : Mensaje in mensajes){
12    if(msg.nombre==""){
13          GUILayout.Label (msg.msg);
14    }else{
15          GUILayout.Label (msg.nombre+": "+msg.msg);
16    }
17 }
18 ...
19 function sendMsg(msg : String){
20    msg = msg.Replace("\n", "");
21    networkView.RPC("setChatText", RPCMode.All, nickName, msg);
22    ...
23 }
24 @RPC
25 function setChatText(name : String, msg : String){
26    var entry : Mensaje = new Mensaje();
27    entry.nombre = name;
28    entry.msg = msg;
29    mensajes.Add(entry);
30    if(mensajes.length> 10){
31          mensajes.RemoveAt(0);
32    }
33 }
```

*Figura 48, El Script Chat*

Como se puede observar en el código de la Figura 48, lo primero que se realiza es crear la estructura del mensaje (líneas 1 a 4), a través de la definición de una clase (un tipo de dato) llamada `Mensaje`, la cual contendrá en su interior el nickname y el mensaje que envía un jugador. Luego, se verifica la entrada del usuario, con el fin de enviar el mensaje. Esto se realiza en base a la captura de eventos, con la ayuda de la clase `Event`, la cual indica con uno de sus atributos (`current`), el evento que se está produciendo actualmente, permitiendo así identificar si la tecla 'enter' (representada por el carácter '`\n`') ha sido presionada. De ser así (siempre y cuando el usuario ha ingresado algún mensaje), se envía el mensaje mediante la función `sendMsg`, a la cual se pasa como parámetro el mensaje (`inputField`) del jugador. La función `sendMsg`, ubicada en la línea 19 dentro del código de la Figura 48, se encarga de realizar la llamada al procedimiento remoto denominado `setChatText`, el cual es invocado en cada uno de los jugadores (incluyendo al local).

Universidad del Bío-Bío. Red de Bibliotecas - Chile



El procedimiento remoto `setChatText` (línea 25), recibe como parámetro el nickname y el mensaje que un jugador envió, los cuales se almacenan en un objeto de la clase `Mensaje`. Luego, éste mensaje (`entry`), se agrega al arreglo (`mensajes`) donde se guardan los mensajes de todos los jugadores. Notar que en el arreglo solamente se almacenan hasta 10 mensajes, ya que se van removiendo los mensajes más antiguos (que están al principio del arreglo) con el fin de no saturar la ventana de chat con demasiados mensajes. El ciclo `for` que se aprecia en la línea 11 de la Figura 48, es el encargado de escribir por pantalla cada uno de los mensajes contenidos en el arreglo `mensajes`.

Obviamente, cada una de estas acciones (enviar y escribir un mensaje), se llevan a cabo en una GUI, en donde se ingresan y muestran los mensajes. A continuación, se presenta la Figura 49 que muestra el chat en funcionamiento. Como se puede apreciar, cada jugador posee, en la esquina inferior derecha, una ventana en la cual puede comunicarse o chatear con los demás jugadores. Además, notar que cuando un jugador se une al juego, se envía un mensaje a cada uno de los jugadores informando de esta situación.

*Figura 49, El Chat en Ancient Gale*

## 5.7 La Interfaz de Usuario

En la siguiente sección, se aborda el tema relacionado con la interfaz de usuario (GUI) de Ancient Gale. La implementación de la GUI en Unity3D se debe llevar a cabo dentro de la función `OnGUI` (en cualquier script), la cual es invocada

Universidad del Bío-Bío. Red de Bibliotecas - Chile



automáticamente por el motor (cada frame) para dibujar por pantalla elementos como botones, ventanas, campos de textos, texturas, etc. Cabe mencionar que, en la realización de este proyecto se debió crear diversos elementos de la GUI, como la ventana de chat, el menú principal, el status del personaje y enemigos (salud y magia), entre otros.

Principalmente, se aborda lo relacionado con el menú principal, ya que abarca la mayoría de los elementos de GUI implementados en el proyecto. Ahora bien, el menú principal está compuesto por 5 ventanas: la de ingreso del nickname, la de selección del color del personaje, la del establecimiento de una conexión con un servidor, la utilizada para crear un servidor y la que se usa para salir del juego. En cuanto a la ventana de ingreso del nickname, ésta se describe a continuación con el siguiente trozo de código:

```
1   function windowName(){
2      GUILayout.BeginHorizontal ();
3      GUILayout.Label("Nickname:");
4      nickName = GUILayout.TextField(nickName,
5                  GUILayout.MaxWidth(130));
6      GUILayout.EndHorizontal();
7      if(GUI.changed)
8            PlayerPrefs.SetString("nickName", nickName);
9      if(nickName == "")
10           enterNick = false;
11     else
12           enterNick = true;
13  }
```

*Figura 50, Ventana de ingreso del nickname*

Como se aprecia en el código de la Figura 50, se crea la ventana de ingreso del nickname del jugador a través de la función `windowName` (invocada desde la función `OnGUI`). Ésta función hace uso de la clase `GUILayout` para posicionar los distintos elementos que poseerá la ventana. En primer lugar se utiliza la función `BeginHorizontal`, la cual indica a Unity3D que todos los elementos de GUI que se añadan a la ventana, se posicionaran horizontalmente en ésta. Luego se añade una etiqueta a través de la función `Label` (de la clase `GUILayout`), pasándole como parámetro la cadena "`Nickname`". Posteriormente, en la línea 4, se agrega un campo de texto por medio de la función `TextField`, también de la clase `GUILayout`, pasándole como parámetros el texto que contendrá por defecto y el número de columnas que poseerá. Esta función retorna el texto ingresado por el usuario en el campo de texto. Finalmente, se invoca a la función `EndHorizontal`, que indica a Unity3D que ya no se

Universidad del Bío-Bío. Red de Bibliotecas - Chile



dispondrán los elementos de la GUI en forma horizontal. El código que viene consecutivamente (línea 7), tiene como objetivo guardar el nickname del usuario, con el fin de conservarlo después que el jugador salga del juego. Esto se logra mediante la clase `PlayerPrefs`, la cual permite almacenar y recuperar valores (enteros, de punto flotante y cadenas de caracteres) desde un registro, en este caso, de Windows. También, se verifica si el usuario ha ingresado su nickname, por lo que de no ser así, no podrá comenzar a jugar.

Por otro lado, se tiene a la ventana de selección del color del personaje principal, en donde el usuario puede seleccionar entre 5 colores (verde, azul, rojo, amarillo, gris y negro), con el fin de personalizar su personaje. Básicamente, en esta ventana se utilizan botones de selección, donde solamente se puede seleccionar uno solo. A continuación, en la Figura 51, se presenta la función que crea esta ventana:

```
1   function windowMenu(){
2      GUILayout.BeginHorizontal ();
3      GUILayout.Label("Selecciona Color:");
4      GUILayout.EndHorizontal();
5      GUILayout.BeginHorizontal ();
6      toggleGreen = GUILayout.Toggle(toggleGreen, colores[0]);
7      if(toggleGreen){
8            toggleBlue = false;
9            toggleRed = false;
10           toggleYellow = false;
11           toggleGris = false;
12           toggleBlack = false;
13     }
14     ...
15  }
```

*Figura 51, Ventana de selección del color del personaje*

Como se observa en la Figura 51, al igual que la ventana anterior, se llama a la función `BeginHorizontal`, para posicionar horizontalmente los componentes, luego se crea una etiqueta, para después, en la línea 6, crear el botón de selección por medio de la función `Toggle` (de la clase `GUILayout`), a la cual se pasa como parámetro el valor booleano por defecto que indica si el botón está seleccionado y la textura correspondiente al color. Notar que `colores` (línea 6), corresponde a un arreglo que contiene cada una de las texturas de los colores que se pueden seleccionar. La función `Toggle`, retorna el valor actual del botón, es decir, si el usuario ha seleccionado o no el botón. A continuación, se verifica si se ha seleccionado el botón, por lo cual se desactivan cada uno de los demás. Cabe mencionar que este proceso se realiza de la

Universidad del Bío-Bío. Red de Bibliotecas - Chile



misma forma con cada uno de los otros botones, por lo que su código no se ha presentado.

La siguiente ventana corresponde a la de salir del juego, la cual permite al jugador cerrar la aplicación. A continuación, en la Figura 52, se presenta parte de su código:

```
1  function windowOpciones(){
2     GUILayout.BeginVertical();
3     GUILayout.Space(5);
4     GUILayout.EndVertical();
5     GUILayout.BeginHorizontal ();
6     GUILayout.FlexibleSpace();
7     if(GUILayout.Button("Salir", GUILayout.Width(115),
8           GUILayout.Height(25))){
9           Application.Quit();
10    }
11    GUILayout.FlexibleSpace();
12    GUILayout.EndHorizontal();
13 }
```

*Figura 52, Salir del juego*

Al apreciar el código de la Figura 52, se puede observar que aparecen nuevas funciones de la clase `GUILayout`, con respecto a las otras ventanas. El primer caso es la de la función `BeginVertical`, la cual permite disponer los elementos en forma vertical, mientras que `Space` permite insertar un espacio (medido en pixeles) dentro de la ventana. Otra, es la función `FlexibleSpace` (línea 6), que agrega un espacio a la ventana, el cual se acomoda flexiblemente dentro de ésta, es decir, si la ventana cambia de tamaño, el espacio se ajustará automáticamente. También, se puede apreciar en la línea 7 la función `Button`, la cual agrega un botón a la ventana. La función `Button`, recibe como parámetro el texto que describirá la acción del botón y el ancho que tendrá. Ésta función retorna un valor booleano que indica si el botón ha sido presionado por el usuario. De ser así, se llama a la función `Quit` de la clase `Application`, la cual permite salir del juego.

En definitiva, las ventanas restantes se crean de forma muy similar a las vistas anteriormente, utilizando la clase `GUILayout` y sus funciones, disponiendo los elementos horizontal o verticalmente dentro de la ventana, capturando los eventos que el usuario podría realizar, esto es, ingresar texto o presionar un botón. Ahora, como se mencionó anteriormente, todas estas ventanas y elementos de GUI se deben crear dentro de la función `OnGUI`. Pues bien, a continuación, en la Figura 53, se presenta

Universidad del Bío-Bío. Red de Bibliotecas - Chile



parte del código que invoca a las funciones presentadas anteriormente desde la función `OnGUI`:

```
1  function OnGUI(){
2      ...
3      if(guiSkin)
4            GUI.skin = guiSkin;
5      GUILayout.Window(0, Rect(Screen.width*0.015,Screen.height*0.25,
6                        240, 50), windowName, "Player");
7      if(!enterNick){
8            return;
9      }
10     GUILayout.Window(4,Rect(Screen.width*0.015
11           Screen.height*0.25+205, 240, 100), windowMenu, "Menu");
12     GUILayout.Window(3, Rect(Screen.width*0.015+235,
13           Screen.height*0.25+205, 240, 100), windowOpciones,
14           "Opciones");
15     ...
```

*Figura 53, La función OnGUI*

Como se observa en la Figura 53, cada una de las ventanas se crea a través de la función `Window` de la clase `GUILayout`, a la cual se pasa como parámetro el *id* de la ventana (el cual debe ser único), la posición y tamaño de la ventana, por medio de la definición de un área rectangular (función `Rect`, de la clase `Rect`), el nombre de la función encargada de construir la ventana (definidas anteriormente) y el titulo que se visualizará en su parte superior. Nótese que si el usuario no ha ingresado su nickname (línea 7), las demás ventanas no se desplegarán. El atributo `guiSkin`, de la línea 3, se refiere al estilo visual que tendrá cada uno de los componentes de la GUI, el cual puede ser establecido a gusto de los desarrolladores por medio de la variable `skin` de la clase `GUI`. De hecho, Unity3D permite crear estilos visuales, en donde los desarrolladores pueden definir sus propias texturas y elementos gráficos para los botones, campos de textos, ventanas, etc. A continuación, se presenta la Figura 54, que muestra el resultado final después de implementar el código que construye el menú principal de Ancient Gale:

Universidad del Bío-Bío. Red de Bibliotecas - Chile

*Figura 54, El menú principal de Ancient Gale*

## 5.8 La Batalla Final, los Enemigos

Otro de los aspectos importantes dentro de cualquier videojuego son los enemigos, ya que son los que hacen frente a los jugadores y dificultan el logro de sus objetivos. Pues bien, Ancient Gale no es la excepción, ya que cuenta con enemigos que impedirán que el jugador llegue sano y salvo a rescatar el castillo de las garras del dragón. Básicamente, Ancient Gale cuenta con tres enemigos, un mago, un caballero y el temible dragón. Cada uno de ellos posee habilidades distintas, distinguiéndose principalmente su forma de ataque. Específicamente, el mago posee la habilidad de arrojar magia al jugador, mientras que el caballero posee una espada con la cual ataca al jugador. El dragón por su parte, arroja ráfagas de fuego y posee la habilidad de volar. Ahora bien, el comportamiento que poseen los enemigos en Ancient Gale consiste en que cada uno de ellos, busca dentro del escenario un objetivo (un jugador) al cual atacar, basándose principalmente en la distancia que hay entre él y la posición del objetivo, por lo que si la distancia es muy grande, lógicamente el enemigo no podrá atacar. Además, cada enemigo busca constantemente un objetivo, seleccionando siempre el más cercano. El siguiente script, resume el comportamiento del mago y el caballero, diferenciándose, como se mencionó anteriormente, en su forma de ataque:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
1   function Update () {
2      ...
3      if(GetComponent(EnemyStatus).getIsDead()){
4            return;
5      }
6      ...
7      if(!target){
8            getTarget();
9      }else{
10           findTarget();
11     }
12     if(target && canAttack() && grounded){
13           ...
14           if(Vector3.Distance(transform.position,
15                        target.position)>distance){
16                  anim.CrossFade("run");
17                  moveDirection = new Vector3(0, 0, 0.8);
18                  moveDirection = transform.TransformDirection(
19                                      moveDirection);
20                  moveDirection *= 6.0;
21           }else{
22                  moveDirection = Vector3.zero;
23                  anim.CrossFade("stand");
24           }
25           if((lastTime+magicTimer)<Time.time){
26                  //Atacar
27                  var magic = Instantiate(magicPrefab, pos,
28                        transform.rotation);
29                  magic.rigidbody.AddForce(transform.forward*900);
30           }
31     }else{
32           moveDirection = Vector3.zero;
33           anim.CrossFade("stand");
34     }
35     ...
36  }
37
38  function getTarget(){
39     var targets = GameObject.FindGameObjectsWithTag("Player");
40     target = targets[0].transform;
41  }
42
43  function findTarget(){
44     var targets = GameObject.FindGameObjectsWithTag("Player");
45     for(var go in targets){
46           if(Vector3.Distance(transform.position,
47                  target.position)>Vector3.Distance(transform.position,
48                                        go.transform.position)){
49                  target = go.transform;
50           }
51     }
52  }
```

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
53  function canAttack(){
54    if(Vector3.Distance(transform.position, target.position)>range){
55          return false;
56    }
57    var hit : RaycastHit;
58    if(Physics.Linecast(new Vector3(transform.position.x,
59                transform.position.y+2, transform.position.z),
60                target.position, hit)){
61          if(hit.collider.gameObject.tag != "Player"){
62                return false;
63          }else{
64                return true;
65          }
66    }
67    return true;
68 }
```

*Figura 55, Comportamiento de los Enemigos*

Como se puede apreciar en el código de la Figura 55, la primera función que se encuentra es `Update`, la cual permite actualizar constantemente el estado y comportamiento del enemigo. Véase que, la primera instrucción de `Update` es la consulta a la función `getIsDead` (del script `EnemyStatus`) con el fin de saber si el enemigo está muerto, por lo que de ser así, no realiza ninguna acción. Por el contrario, si el enemigo no está muerto, en primer lugar consulta si posee un objetivo (`target`), de no ser así, busca el primero que encuentre a través de la función `getTarget`. La función `getTarget` (línea 38), realiza una búsqueda por tag por medio de la función `FindGameObjectsWithTag` de la clase `GameObject`, la cual retorna un arreglo con todos los objetos que posean el tag "`Player`" (jugador).

Ahora, si el enemigo ya poseía un objetivo, realiza otra búsqueda, con el fin de obtener, si es que existe, un objetivo más cercano que el anterior. Para ello, se invoca la función `findTarget` (línea 43), la cual también realiza una búsqueda por el tag "`Player`", pero con la diferencia que selecciona el objetivo más cercano a la posición del enemigo. Esto se realiza por medio de la función `Distance` de la clase `Vector3`, la cual entrega la distancia entre dos posiciones. Luego de obtener un objetivo, el siguiente paso es verificar si el enemigo puede o no atacar a su objetivo, mediante la función `canAttack` de la línea 53. Ésta función, en resumidas cuentas, en primer lugar calcula la distancia entre el enemigo y su objetivo, con el fin de compararla con el rango (`range`) de ataque que posee el enemigo, por lo que si la distancia es mayor que el rango de ataque, el enemigo no podrá atacar. Por el contrario, si el objetivo está dentro del rango (distancia menor), se comprueba mediante la función `LineCast` de la clase `Physics`, si existe algún obstáculo entre el enemigo y el objetivo. La función

Universidad del Bío-Bío. Red de Bibliotecas - Chile



`LineCast` crea una línea invisible entre la posición del enemigo y la del objetivo, entregando un valor verdadero si existe un objeto interceptando ésta línea. Por lo que si existe un objeto entre el enemigo y el objetivo (distinto de un jugador), el enemigo no podrá atacar.

Ahora bien, si el objetivo se encuentra dentro del rango de ataque y no existe un obstáculo de por medio, el enemigo está en condiciones de atacar. Sin embargo, antes se debe calcular otra distancia, que permite saber si se está lo suficientemente cerca del objetivo como para mantener una posición inmóvil, de lo contrario será necesario 'perseguir' al objetivo para alcanzar ésta distancia (`moveDirection`, línea 17). El paso siguiente es atacar al objetivo, considerando un periodo de ataque, con el fin de impedir que el enemigo ataque constantemente al jugador sin dejarlo respirar. Particularmente, en el caso del código de la Figura 55, el modo de ataque corresponde al del mago, el cual instancia dentro de la escena, mediante la función `Instantiate` (línea 27), el prefab que representa su magia, al que se le aplica una fuerza por medio de la función `AddForce` de la clase `RigidBody`, permitiendo así que la magia se mueva a una velocidad constante en dirección hacia el objetivo. Cabe destacar que el modo de ataque del caballero, se diferencia en algunos aspectos en comparación con la del mago, ya que el caballero necesita estar a una distancia más cercana del objetivo para poder atacar, debido a que posee una espada y no como el mago que puede arrojar magia desde distancias más grandes. Por lo que, el caballero solamente podrá atacar sí está a una distancia específica del objetivo.

Como se mencionó anteriormente, Ancient Gale cuenta con un enemigo final, el cual es parte del objetivo principal del juego, esto es, el dragón. Éste enemigo, posee casi la mayoría de las características de los descritos inicialmente, ya que también ataca en base a la distancia con respecto al jugador. Sin embargo, el dragón ostenta la habilidad de volar por los cielos, agregando una dificultad adicional a la batalla. Básicamente, el comportamiento del jefe final de Ancient Gale, consiste en recorrer un arreglo de puntos definidos previamente (una especie de grafo), el cual permite crear el camino que realizará el dragón a través de la batalla con el jugador. Éste camino, se recorre repetidamente, en un ciclo infinito, basándose principalmente en el tiempo. A continuación se describe el script, denominado `Dragon`, que implementa el comportamiento de éste enemigo:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
1   function Update () {
2       ...
3       if(indiceFly == puntosFly.length){
4             indiceFly = 0;
5             isFly = false;
6             indice = Random.Range(0, puntos.length);
7       }
8       if(isFly){
9             targetRotation = Quaternion.LookRotation(
10                  puntoFly[indiceFly].position - transform.position,
11                  Vector3.up);
12            transform.rotation = Quaternion.Slerp(transform.rotation,
13                              targetRotation, Time.deltaTime*1.25);
14            attack.LookAt(target);
15            if(target && canAttack()){
16                  //Atacar
17            }
18            transform.Translate(
19                        Vector3.forward*Time.deltaTime*speed*2);
20            if(Vector3.Distance(transform.position,
21                        puntosFly[indiceFly].position)<1){
22                  indiceFly++;
23            }
24      }else{
25            if(!isGround){
26                  ...
27                  transform.Translate(
28                        Vector3.forward*Time.deltaTime*speed*1.1);
29                  if(Vector3.Distance(transform.position,
30                                    puntos[indice].position)<1){
31                        isGround = true;
32                        lastFlyTime = Time.time;
33                  }
34            }else{
35                  ...
36                  attack.LookAt(target);
37                  transform.rotation.x = 0;
38                  transform.rotation.z = 0;
39                  if(target && canAttack()){
40                        //Atacar
41                  }
42                  if((lastFlyTime+20)<Time.time){
43                        isFly = true;
44                        isGround = false;
45                  }
46            }
47      }
48  }
```

*Figura 56, El Script Dragon*

Uno de los aspectos importantes del script de la Figura 56, es que posee dos estados, uno cuando el dragón está volando (línea 8, `isFly = true`), y el otro cuando está a nivel del suelo. Cuando está volando, el dragón recorre un camino predefinido de puntos (`puntosFly`, línea 10), a través de un índice (`indiceFly`) que indica en que

Universidad del Bío-Bío. Red de Bibliotecas - Chile



segmento del camino se encuentra, y el cual se incrementa cuando la distancia a un punto específico, es suficiente para pasar a otro (línea 20). En la línea 15, queda de manifiesto que el dragón también ataca cuando está volando, utilizando para ello, las mismas funciones descritas para los otros enemigos, esto es, *target* (si tiene un objetivo) y `canAttack` (si el objetivo está a su alcance). Cabe mencionar, que el dragón realiza un ataque que consiste en una ráfaga de fuego, la cual es disparada desde su boca, gracias al punto definido por la variable `attack` (línea 14). Por ende, aquí se desprende que el dragón no necesita estar muy cerca del objetivo para poder atacarlo. Además, por medio de la función `LookAt` (línea 14), el dragón siempre tiene en la mira a su objetivo, por lo que su puntería es muy precisa, pese a la distancia.

Ahora bien, cuando el dragón recorre por completo cada uno de los puntos de su camino (línea 3), pasa al siguiente estado, el que consiste en seleccionar al azar un punto dentro de un arreglo (línea 6), el cual lo posicionará a nivel del suelo. Nótese que recién aquí, el jugador podrá entablar un combate con el enemigo, ya que cuando éste surca los cielos, es imposible para el jugador poder dañarlo.

Una vez seleccionado el punto de aterrizaje, el dragón deberá dirigirse hacia éste (línea 27), hasta tocar el suelo (línea 29, 30 y 31). En el momento que aterriza (`isGround = true`, por ende ejecuta desde la línea 34), el dragón comienza atacar, esta vez del suelo, al jugador, dando la chance de iniciar una batalla cuerpo a cuerpo con éste. No obstante, el dragón solamente se queda en tierra por algunos segundos (línea 42), por lo que cuando transcurre éste tiempo, nuevamente inicia el vuelo, para comenzar otra vez el ciclo descrito anteriormente, esto es, recorrer el camino en el cielo, para luego volver a la tierra. Por último, cabe destacar que, debido a su comportamiento, el dragón es un enemigo un poco más complejo que los vistos anteriormente, por lo que el jugador deberá intentar idear una estrategia, con el fin de seleccionar adecuadamente el momento de realizar un ataque.

Por otro lado, los enemigos no son los únicos que pueden atacar, puesto que los jugadores también pueden defenderse, por lo que se necesita tener un estado del enemigo, el cual permita establecer si éste está vivo o fue 'asesinado' por el jugador. Para ello, se tiene al script `EnemyStatus`, el cual administra la vida del enemigo y lo restablece cuando muere. A continuación, en la Figura 57, se presentan las funciones más importantes de éste script:

Universidad del Bío-Bío. Red de Bibliotecas - Chile



```
1   function OnGUI(){
2      if(isDamage){
3            ...
4            GUI.Box(Rect(screenPos.x+5, screenPos.y*0.25+8,
5                  (life/maxLife)*100, 10), "");
6            GUI.DrawTexture(Rect(screenPos.x, screenPos.y*0.25, 100,
7                        30),frameTexture);
8      }
9   }
10
11  function respawnEnemy(){
12     isDead = false;
13     life = maxLife;
14     transform.position = initPos;
15  }
16
17  function applyDamage(damage : int){
18     ...
19     life -= damage;
20     isDamage = true;
21     lastTime = Time.time;
22     screenPos = Camera.main.WorldToScreenPoint (transform.position);
23     if(life <= 0){
24           life = 0;
25           lastTimeDead = Time.time;
26           isDead = true;
27     }
28  }
```

*Figura 57, el script EnemyStatus*

Al observar el código de la Figura 57, se pueden apreciar tres funciones. La primera de ellas es la función `OnGUI`, la cual se encarga del dibujar en pantalla la vida actual del enemigo, a través de las funciones `Box` y `DrawTexture` de la clase `GUI`. Cabe mencionar que esta vida solamente se muestra por pantalla por algunos segundos, después que el enemigo ha recibido algún daño por parte del jugador. Luego, se tiene a la función `respawnEnemy` (línea 11), la cual se encarga de revivir al enemigo y establecer su posición inicial (`initPos`), después de haber sido muerto por un jugador. Finalmente, está la función `applyDamage` (línea 17), la cual recibe como parámetro el daño (`damage`) producido por un jugador a un enemigo. Ésta función, tiene como fin reducir la vida (`life`) de un enemigo, estableciéndolo como muerto (`isDead = true`) cuando la pierde por completo. La variable `screenPos` (línea 22), representa las coordenadas x, y de la pantalla, con el fin de posicionar correctamente la vida de un enemigo cuando se muestra al usuario. Esto es posible gracias a la función `WorldToScreenPoint` de la clase `Camera`, la cual convierte una posición del mundo 3D (`transform.position`) a una posición 2D, para mostrar por pantalla. La variable `main` de la clase *Camera* representa la cámara principal del juego.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 5.9 Pruebas de Rendimiento

En la siguiente sección se presenta las pruebas de rendimiento realizadas al videojuego Ancient Gale, con el fin de conocer su comportamiento en diferentes computadores y también en situaciones distintas, esto es, jugando en red y solamente en una máquina. Ahora bien, las pruebas fueron realizadas en base a los frames por segundo (FPS), que producía el videojuego, es decir, la cantidad de imágenes que se muestran por pantalla en un segundo. Esto permite conocer la carga del videojuego (cantidad de procesamiento), en diferentes circunstancias. Así, cuando el videojuego experimenta una cantidad de FPS muy bajo (por lo general, inferior a los 20), se percibirá una lentitud, por lo que se concluye que el computador donde corre el videojuego no es apto para poder jugar, ya que su capacidad de procesamiento no es la adecuada para ejecutar fluidamente el videojuego.

Cabe mencionar que, el cálculo de los FPS producidos por el videojuego, fueron realizados a través de un script. Éste script, calcula el promedio de los FPS generados por el videojuego, excluyendo el mínimo y el máximo registrados, ya que por la carga de procesamiento y de circunstancias específicas del computador (sistema operativo y otros programas), se pueden obtener rangos que no son comunes a los obtenidos en situaciones normales, por ende, no se consideran a la hora de calcular el promedio de FPS.

En primer lugar, fue calculado el rendimiento del videojuego en el caso de un solo jugador, donde se utilizó una resolución de 800x600 y se recorrió el juego de principio a fin, es decir, hasta derrotar al dragón. Para ello, se utilizaron dos computadores, cada uno con capacidades diferentes. A continuación se presentan las especificaciones de cada uno de ellos:

**PC1:**

- **CPU**: Intel Pentium Dual Core, 1.6Ghz (2 CPUs).
- **RAM**: 2 GB.
- **Video**: Nvidia Geforce 6200, 512MB de memoria dedicada de video.
- **SO:** Windows XP Profesional.

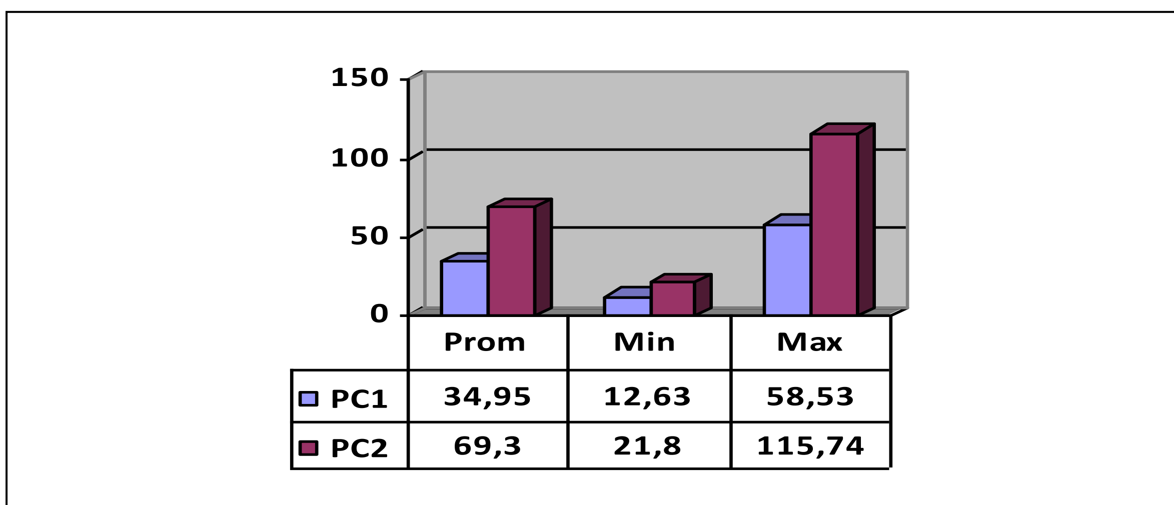
Universidad del Bío-Bío. Red de Bibliotecas - Chile



**PC2:**

- **CPU**: Intel Pentium Dual Core, 2.2Ghz (2 CPUs).
- **RAM**: 2GB.
- **Video**: Nvidia Geforce 310M, 512MB de memoria dedicada de video.
- **SO:** Windows 7 Starter.

El resumen de los cálculos realizados se presenta en el gráfico de la Figura 58, el cual compara el promedio, el mínimo y el máximo de FPS obtenidos en cada computador:




| | Prom | Min | Max |
|---|---|---|---|
| PC1 | 34,95 | 12,63 | 58,53 |
| PC2 | 69,3 | 21,8 | 115,74 |

*Figura 58, Gráfico de rendimiento, caso jugador individual*

Al apreciar el gráfico de la Figura 58, se puede desprender la gran superioridad del PC2 con respecto al PC1, lo cual parece lógico al comparar sus especificaciones. Con respecto a los resultados obtenidos, al realizar las pruebas, se pudo observar que en algunas situaciones del juego, especialmente cuando el jugador lucha contra los enemigos, la cantidad de FPS disminuye. Además, si bien se obtuvieron mínimos por debajo de lo recomendado, en ambos casos el promedio de FPS permitió visualizar fluidamente el videojuego.

Como se mencionó anteriormente, otra de las pruebas realizadas a Ancient Gale fue en el caso de multiplayer, donde se creó una red inalámbrica local punto a punto, con el fin de conectar a los jugadores. Cabe mencionar que en este caso se realizó la prueba con dos grupos de 5 jugadores, a una resolución, al igual que el caso anterior,

Universidad del Bío-Bío. Red de Bibliotecas - Chile



de 800x600, recorriendo el juego de principio a fin. Cada grupo contó con equipos de similares características, las cuales se resumen a continuación:

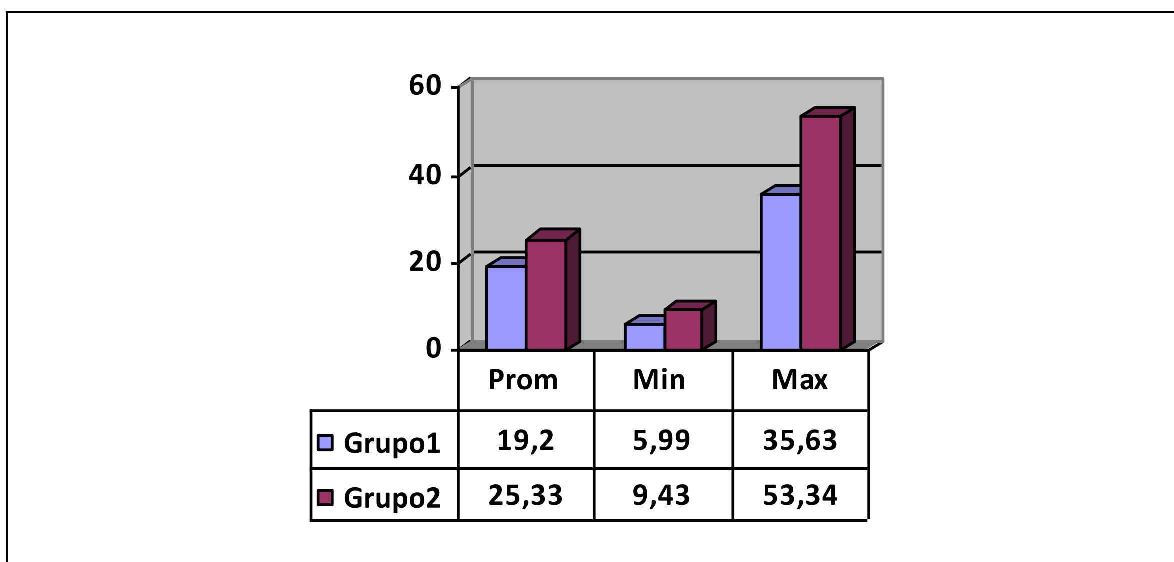
**Grupo1:**

- **CPU**: Intel Core 2 Duo, 2GHz (2 CPUs).
- **RAM**: 1GB.
- **Video**: Mobile Intel 965, 384MB de memoria dedicada de video.
- **SO**: Windows XP Profesional.

**Grupo2:**

- **CPU**: Intel Core i3, 2.13GHz (4 CPUs).
- **RAM**: 3GB.
- **Video**: Intel Graphics Media Accelerator HD, 1024MB de memoria dedicada de video.
- **SO**: Windows XP Profesional.

Los resultados obtenidos de cada grupo, se presentan a continuación en el gráfico de la Figura 59, el cual, al igual que el caso anterior, ilustra el promedio, el mínimo y el máximo de FPS generados en cada grupo (promedio):





| | Prom | Min | Max |
|---|---|---|---|
| Grupo1 | 19,2 | 5,99 | 35,63 |
| Grupo2 | 25,33 | 9,43 | 53,34 |

*Figura 59, Gráfico de rendimiento, caso multiplayer*

Universidad del Bío-Bío. Red de Bibliotecas - Chile



Como se aprecia en el gráfico de la Figura 59, el rendimiento disminuye en comparación con el caso de un solo jugador (Figura 58). Esto se debe lógicamente, al aumento de objetos dentro de la escena, que además, interactúan entre sí y los cuales deben ser sincronizados a través de la red. Por otro lado, a pesar que en el caso del grupo 1, el promedio obtenido de FPS es inferior al recomendado, el juego se pudo observar con una fluidez aceptable, salvo en ocasiones donde interactúan muchos jugadores con varios enemigos, donde se percibe un leve retardo, pero casi imperceptible. Esto puede deberse a que las especificaciones de los equipos del grupo 1 son relativamente bajas, en comparación con las del grupo 2, es más, ambos grupos no cumplen con las especificaciones recomendadas para cualquier videojuego moderno (salvo el procesador y la RAM), producto a que la tarjeta de video que poseen no es muy reconocida en el exigente rubro de las aplicaciones 3D, como lo son, en cambio, las tarjetas Nvidia o ATI.

Así, al analizar los datos obtenidos luego de realizar las pruebas, se puede concluir que para ejecutar fluidamente el videojuego, se requiere como requisitos mínimos las siguientes especificaciones:

- **CPU**: Intel o AMD de 3.0GHz o superior.
- **RAM**: 1GB para Windows XP y 2GB para Windows Vista y Windows 7.
- **Video**: tarjeta aceleradora de gráficos con memoria dedicada de video de 256MB o superior (de preferencia Nvidia o ATI).

Notar que las especificaciones, son aproximaciones en base a los resultados obtenidos, que además, se ajustan a los requisitos de la mayoría de los videojuegos que están disponibles en el mercado.

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# Capítulo VI

# CONCLUSIONES

Luego de finalizar cualquier proyecto, se debe tomar el tiempo para reflexionar y analizar los resultados obtenidos , más aún, considerando que el desarrollo de un videojuego, difiere del proceso de construcción de software tradicional.

Es por esto que, en este último capítulo, se realizarán las conclusiones del trabajo realizado, pasando desde los resultados obtenidos, hasta la experiencia adquirida en el desarrollo del proyecto. Además, se plantearán posibles trabajos futuros, con el fin de extender y ampliar las capacidades, no solamente de Ancient Gale, sino también de otros nuevos proyectos.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 6.1 Conclusión

Después haber finalizado la implementación del proyecto, se pueden sacar las siguientes conclusiones:

- ✓ El desarrollo de un videojuego, implica una fuerte participación de diversas disciplinas artísticas, como el diseño gráfico, por lo que se debió lidiar con nuevos desafíos, tanto como adquirir nuevos conocimiento, así como también establecer una comunicación con integrantes de otras especialidades. Es aquí donde entra en juego la participación y cooperación de la empresa Jobbitgames, cuyos miembros ayudaron a dar movimiento, vida y el toque artístico a Ancient Gale.

- ✓ Sin duda, la utilización de herramientas especializadas, como el caso de el motor Unity3D, ayudan de sobremanera en la producción y desarrollo de software complejo, como lo es un videojuego. Permitiendo así, por ejemplo, que un equipo de trabajo reduzca costos, tiempo y complejidades, a la hora de implementar un proyecto. Por lo tanto, es muy útil buscar tecnologías apropiadas, que entreguen éste tipo de ventajas a la hora de emprender un proyecto.

- ✓ Al utilizar una herramienta, como Unity3D, la cual basa la mayor parte de su funcionamiento en la utilización de componentes, se tiene un software extensible y de fácil mantenimiento.

- ✓ Además, es conveniente mencionar la importancia de seleccionar herramientas de desarrollos con una completa documentación y soporte, como lo es el caso de Unity3D (a diferencia de otras como Torque), siendo un factor clave a la hora de utilizar y conocer todas las funcionalidades que la plataforma ofrece, no dejando de lado la gran ayuda que aporta a la solución de problemas.

- ✓ Con éste trabajo, se mostró que no está muy alejado de la realidad de las empresas chilenas, implementar un videojuego 3D multiplayer online básico, con un notable rendimiento y en un plazo de menos de 4 meses, descontando, eso sí, la etapa de diseño y la construcción de modelos 3D.

- ✓ El logro de la implementación de la totalidad de las funcionalidades propuestas al inicio de éste desarrollo, cumpliendo así, con uno de los objetivos fundamentales de todo proyecto, los plazos y la planificación trazada.

Universidad del Bío-Bío. Red de Bibliotecas - Chile



## 6.2 Resultados Obtenidos

Como principales resultados obtenidos, luego de haber finalizado el desarrollo del videojuego, se pueden mencionar los siguientes:

- ✓ Un videojuego respaldado con componentes y funcionalidades de nivel profesional, que sumerge al usuario final en una experiencia única.
- ✓ Finalizar, en poco más de tres meses (descontando el diseño y el modelado de modelos 3D), un videojuego multiplayer online muy básico y acotado, completamente 3D. Hecho que hubiese sido imposible sin la ayuda de herramientas como Unity3D y con la experiencia adquirida previamente, como resultado de prácticas profesionales en el área y el trabajo continúo con diferentes motores y técnicas de computación gráfica.

## 6.3 Proyección a Futuro

En cuanto a los futuros trabajos y proyectos que podrían realizarse en base a Ancient Gale, existen diversas opciones, destacando especialmente las que siguen:

- ✓ Agregar más escenas, enemigos y niveles de dificultad, permitiendo al jugador explorar otros mundos, con características y retos propios.
- ✓ Permitir a los jugadores detectar automáticamente, los servidores disponibles para unirse a un juego.
- ✓ Aprovechar la conexión con la base de datos, con el fin de llevar un registro de usuarios, permitiendo así un sistema de identificación de jugadores, mediante un nickname y una contraseña.
- ✓ Permitir a los usuarios personalizar su personaje, como por ejemplo poder elegir el sexo, la raza (elfo, enano, mago, etc.), la vestimenta, entre otras, lo cual es muy común en videojuegos de rol y sobre todo si son en línea.
- ✓ Agregar nuevas habilidades al personaje principal, como por ejemplo, subir de nivel (fortalecer su fuerza, vida o magia), obtener nuevas armas e ítems, la posibilidad de poder nadar, volar, etc.
- ✓ Incluir nuevos efectos visuales como la lluvia, el ciclo de día y noche, una tormenta de arena, entre otros.

En fin, un mundo de posibilidades, que solo se ven limitadas por la imaginación y creatividad de quienes desarrollen el proyecto.

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# BIBLIOGRAFÍA


1. BRACKEEN, D. “et al”. 2003. Developing Games in Java. [chm]. 1008p.

2. Definición de Vectores. [En línea]. <http://www.tochtli.fisica.uson.mx/electro/vectores/definici%C3%B3n_de_vectorev.htm>. [Consulta: 6 de Octubre 2010]

3. FINNEY, K. 2004. 3D Game Programming All in One. [pdf]. 850p.

4. GOLDSTONE, W. 2009. Unity Game Development Essentials. [pdf]. 316p.

5. How Much Do You Know About Video Games? Video game industry statistic. [En línea]. <http://www.esrb.org/about/video-game-industry-statistics.jsp>. [Consulta: 15 de Octubre 2010]

6. LILLY, P. 2009. Doom to Dunia: A Visual History of 3D Game Engines. [En línea]. <http://www.maximumpc.com/article/features/3d_game_engines>. [Consulta: 6 de Octubre 2010]

7. MAURINA, E. 2006. The Game Programmer´s Guide to Torque: under the Hood of the Torque Game Engine. [pdf]. 620p.

8. SON, J. 2003. The Rendering Pipeline. [en línea]. <http://www.cse.ttu.edu.tw/~jmchen/cg/docs/rendering%20pipeline/Chap4_1%5B1%5D.ppt>. [Consulta: 3 de Enero 2011]

9. The Video Game Industry: An $18 Billion Entertainment Juggernaut. [En línea]. <http://seekingalpha.com/article/89124-the-video-game-industry-an-18-billion-entertainment-juggernaut>. [Consulta: 6 de Octubre 2010]

10. UNITY TECHNOLOGIES. Unity3D: Manual. 2010. [en línea]. <http://unity3d.com/support/documentation/Manual/index.html>. [Consulta: 29 de Octubre de 2010]

11. UNITY TECHNOLOGIES. Unity3D: Scripting Overview. 2010. [en línea]. <http://unity3d.com/support/documentation/ScriptReference/index.html >. [Consulta: 10 de Noviembre de 2010]

Universidad del Bío-Bío. Red de Bibliotecas - Chile


12. VALIENT, M. 2001. 3D Engines in games – Introduction. [En línea]. <http://pisa.ucsd.edu/cse125/2006/Papers/Introduction_to_3d_Game_EngiEng.pdf>. [Consulta: 6 de Octubre 2010]

13. WARD, J. 2008. What is a Game Engine? <http://www.gamecareerguide.com/features/529/what_is_a_game_.php?page=1>. [Consulta: 6 de Octubre 2010]

14. ZHANG, H. y LIANG, D. 2006. Computer Graphics Using Java 2D and 3D. [chm]. 632p.

Universidad del Bío-Bío. Red de Bibliotecas - Chile

# ANEXOS

## ANEXO A: Historia de Ancient Gale

Ancient Gale se enmarca en una tierra de fantasía, en donde un antiguo reino ha sido invadido por un cruel y temible dragón, que cada 1.000 años siembra su maldición y miedo entre los habitantes de estas tierras. Para lo cual, el rey ha convocado a los más valientes para combatir y liberar al reino de las garras del dragón.

En primer lugar, los viajeros deberán encontrar la espada legendaria, artefacto indispensable para luchar contra el dragón, la cual se encuentra oculta en algún lugar de la aldea. Luego, deberán dirigirse al gran bosque en busca del fuego sagrado, el cual ha permanecido ahí durante milenos, custodiado por peligrosas criaturas. Finalmente, el último paso será entrar al castillo para librar la más grande de las batallas, contra el dragón, utilizando la espada legendaria y el fuego sagrado, que juntos pueden derribar hasta el enemigo más poderoso.

El viajero que logre la hazaña de derrotar al dragón, será nombrado con el título de "Héroe Legendario", el cual ha sido ostentado por generaciones, sólo por los más valientes guerreros.

Universidad del Bío-Bío. Red de Bibliotecas - Chile

## ANEXO B: JavaScript

Unity3D posee una versión propia de JavaScript, a través de la cual es posible implementar la lógica y comportamiento de los objetos dentro de un videojuego. Ahora bien, éste anexo tiene como fin abordar las principales características de la sintaxis del lenguaje en cuestión.

Específicamente, en JavaScript existen dos formas de definir una variable, una a través de especificar explícitamente el tipo de variable y la otra sin especificar el tipo de la variable. La Figura 60 ilustra la definición de variable en JavaScript [11]:

```
1  …
2 var variable1 : Tipo; //variable con tipo especificado
3 var variable 2; //variable sin tipo especificado
4  …
```

*Figura 60, Definición de variables en JavaScript*

Como se puede apreciar en la Figura 60, al igual que en otros lenguajes de programación, JavaScript también cuenta con comentarios, los cuales pueden ser de la forma "//", para comentarios de una línea, o como "/*…*/", para comentarios de más de una línea. Además, se puede observar en el código de la Figura 60 que, cada definición de una variable está precedida por la palabra "var", esto se debe a que ésta es una palabra reservada del lenguaje y sirve para definir variables. También es posible definir el tipo de acceso a una variable, la cual puede ser "public" o "private". Las variables con acceso "public" pueden ser accedidos desde cualquier otro componente, mientras que las "private" solo desde el Script que los definió. En el caso de la Figura 60, las variables son de tipo "public", ya que esto sucede cuando no se coloca explícitamente el tipo de acceso (antes de la palabra "var"). Cabe mencionar que, cada variable definida fuera de una función, se convierte en una variable miembro [11], la cual puede ser accedida desde cualquier otro componente (siempre y cuando sea "public"), e incluso desde el panel de inspección (tratado más adelante). Sin embargo, si una variable es definida dentro de una función, ésta es solamente accesible dentro del cuerpo de la función. Igualmente, una variable puede ser definida como estática (variable global), a través de la palabra clave "static" (entre "public" y "var"), la cual es común para todas las instancias de la clase y la cual puede ser accedida mediante el nombre de la clase (nombre del Script), seguido de un ".", y el nombre de la variable estática (NombreClase.nombreVariableEstática).

Universidad del Bío-Bío. Red de Bibliotecas - Chile

Por otro lado, una función corresponde a un bloque de código que realiza una acción específica, la cual puede ser llamada o invocada a través de otros componentes que posean una referencia de la clase a la cual pertenece la función, o producto de un evento particular (la pulsación de una tecla, la colisión de dos objetos, etc.). En JavaScript, una función se define de la siguiente forma [11]:

```
1   function unaFuncion(parametro1, parámetro2 :  Tipo){
2      //Cuerpo de la función
3      …
4      var unObjeto;
5      return unObjeto;
6   }
7
8   function otraFuncion() : String{
9      //Cuerpo de la función
10     …
11     var unObjeto : String = "Hello World";
12     return unObjeto;
13  }
```

*Figura 61, Definición de una función en JavaScript*

Como se puede observar en el código de la Figura 61, una función se define a través de la palabra clave "function", seguido del nombre de la función. Ésta puede recibir cero o más parámetros (pudiendo o no especificar el tipo), además, de la misma forma que los atributos, se puede especificar su tipo de acceso, "public" o "private", siendo el primero el tipo de acceso por defecto si no se especifica. También es posible retornar un valor u objeto desde una función, a través de la palabra clave "return". Al apreciar la línea 8 del código de la Figura 61, se puede desprender que igualmente es posible especificar el tipo de valor de retorno, así como también se puede obviar, por lo que se deja a elección de los desarrolladores seleccionar la opción más cómoda, la que puede influir a la hora de leer y depurar el código. Del mismo modo que los atributos, se puede declarar una función como estática,  a través de la palabra clave "static" (antes de "function"), la cual podrá ser accedida a través de otros componentes por medio del nombre de la clase (nombre del Script), seguido de un ".", y el nombre de la función estática (NombreClase.nombreFuncionEstática()).

Universidad del Bío-Bío. Red de Bibliotecas - Chile

**ANEXO C: Funciones de JavaScript**

A continuación se presenta un resumen de las funciones más importantes de JavaScript que Unity3D provee [11]:

| Función | Descripción |
|---|---|
| `Update() : void` | Es invocada por cada frame (cuadro de imagen) que ocurre en el juego. Esta función es la más comúnmente utilizada para implementar cualquier tipo de comportamiento, ya que sirve para actualizar estados, como animaciones, manejar y capturar la entrada del usuario (la pulsación de una tecla, por ejemplo), etc. |
| `Awake() : void` | Es llamada cuando una instancia del Script es cargada. Se utiliza para inicializar variables, ya que se invoca antes que cualquier llamada a la función `Start`. Además, esta función se llama solo una vez durante el ciclo de vida del Script. |
| `Start() : void` | Es invocada justo antes que cualquier llamada al método `Update`. Al igual que la anterior, esta función se llama solo una vez, y es utilizada, por ejemplo, para intercambiar información entre objetos, considerando que en la llamada a `Awake`, ya se inicializaron las variables. |
| `OnEnable() : void` | Esta función es llamada cuando el Script se vuelve activo. Cabe mencionar, que un objeto o componente de una escena, puede ser activado o desactivado, lo que se traduce en que esté o no presente en una escena. |
| `OnDisable() : void` | Se invoca cuando el Script es desactivado. También, cuando el objeto al cual pertenece el |

Universidad del Bío-Bío. Red de Bibliotecas - Chile

| | |
|---|---|
| | Script es destruido o eliminado de una escena. |
| `OnGui(): void` | Esta función es utilizada para renderizar y manejar los eventos de la GUI (Graphics User Interface, Interfaz Gráfica del Usuario). Es decir, dentro de esta función se debe implementar todo lo relacionado con la interfaz de usuario. |
| `OnTriggerEnter(other : Collider) : void` | Esta función se invoca cuando un objeto `Collider` (que representa otro objeto dentro de la escena, que posee un componente que le permite colisionar), ingresa al "Trigger" del objeto al cual pertenece el Script. Un "Trigger" consiste en un área especial de un objeto, la cual puede ser representada por un cubo, esfera u otra geometría 3D, y en la cual cuando otro objeto ingresa en ella o la intercepta, se suelta el "gatillo", por lo que se produce la llamada a la función que realizará la funcionalidad o acción que trate este evento. El parámetro que recibe esta función, representa al objeto que ingreso al "Trigger". |
| `OnTriggerStay(other : Collider) : void` | Se diferencia del anterior en que es llamada una vez cada frame, mientras un objeto `Collider` permanece dentro del "Trigger". |
| `OnTriggerExit(other : Collider) : void` | Se invoca cuando un objeto `Collider`, sale del "Trigger" del objeto al cual pertenece el Script. |
| `OnCollisionEnter(collisionInfo : Collision)` | Es llamada cuando el objeto al cual pertenece el Script, el cual posee los componentes de `collider` o `rigidbody`, colisiona o toca a otro objeto (que también posee uno de estos |

Universidad del Bío-Bío. Red de Bibliotecas - Chile

| | |
|---|---|
| | componentes). El parámetro que recibe esta función, contiene toda la información acerca de la colisión, esto es, puntos de contacto, velocidad, el otro objeto con el cual se colisiono, etc. Cabe mencionar, que existen dos funciones más (al igual que las funciones que manejan los "Trigger"), que controlan cuando dos objetos se mantienen en colisión (`OnCollisionStay`) y cuando dejan de estarlo (`OnCollisionExit`). |
| `OnApplicationQuit() : void` | Se invoca antes de que se salga o cierre el juego. |
| `OnMouseEnter() : void` | Se invoca cuando el mouse entra o pasa por encima de un elemento de la GUI o por un objeto `Collider`. |
| `OnMouseOver() : void` | Se invoca por cada frame, mientras el mouse este sobre un elemento de la GUI o en un objeto `Collider`. |
| `OnMouseExit() : void` | Se llama cuando el mouse no sigue estando sobre un elemento de la GUI o un objeto `Collider`. |
| `OnMouseDown() : void` | Se invoca cuando se presiona un botón del mouse, sobre un elemento de la GUI o en un objeto `Collider`. |
| `OnMouseUp() : void` | Se invoca cuando se suelta un botón del mouse, sobre un elemento de la GUI o en un objeto `Collider`. |
| `OnMouseDrag() : void` | Es llamada cuando se presiona un botón del mouse sobre un elemento de la GUI o en un |

Universidad del Bío-Bío. Red de Bibliotecas - Chile

| | |
|---|---|
| | objeto `Collider`, y el cual es mantenido. Por ende, esta función se llama cada frame mientras se mantenga presionado el botón del mouse. |
| `OnPlayerConnected(player : NetworkPlayer) : void` | Esta función se invoca en un servidor, cuando un jugador (`player`), se conecta correctamente a éste. El parámetro que recibe esta función, representa la estructura de datos que contiene la información del jugador que se conecto al servidor, esto es, su dirección IP, su puerto, entre otros datos. Esta función se utiliza principalmente para crear los objetos correspondientes a través de la red, asociados al jugador que se conecto. |
| `OnPlayerDisconnected(player: NetworkPlayer) : void` | Esta función se invoca en un servidor, cuando un jugador (`player`), se desconecta de éste. Esta función se utiliza para destruir y eliminar los objetos de la red, asociados al `player` que se desconecto. |
| `OnConnectedToServer() : void` | Es invocada en un cliente, cuando se ha conectado correctamente a un servidor. |
| `OnDisconnectedFromServer(mode : NetworkDisconnection) : void` | Se llama en un cliente cuando se pierde la conexión con el servidor o cuando se desconecta voluntariamente de éste. El parámetro de esta función, indica cual fue la razón de la desconexión. |