# Tabla de contenidos

[Tabla de contenidos 2](#_Toc103636867)

[Capítulo 1 - Introducción 6](#_Toc103636868)

[1.1 Motivación del proyecto 6](#_Toc103636869)

[1.2 Objetivos 6](#_Toc103636870)

[1.3 Concepto de videojuegos 7](#_Toc103636871)

[1.4 Géneros de videojuegos 7](#_Toc103636872)

[1.4.1 Juegos de plataforma 8](#_Toc103636873)

[1.4.2 Endless Runner 8](#_Toc103636874)

[1.5 Partes de un videojuego 9](#_Toc103636875)

[1.5.1 Motor 9](#_Toc103636876)

[1.5.2 Assets 10](#_Toc103636877)

[1.5.3 Scripts 10](#_Toc103636878)

[Capítulo 2 – Diseño de un videojuego 11](#_Toc103636879)

[2.1 Introducción 11](#_Toc103636880)

[2.2 Arquitectura software de videojuegos 14](#_Toc103636881)

[2.2.1 Arquitectura software general 15](#_Toc103636882)

Capítulo 2: Diseño de un videojuego .............................................................. 10

2.1 Introducción …………………………………………………………………………… 10

2.2 Arquitectura software de videojuegos ……………………………………………… 11

2.2.1 Arquitectura software general ........................................................................... 12

2.2.2 Arquitectura software para las entidades de juego ……………………………… 13

2.2.3 Jerarquías de clases ......................................................................................... 14

2.2.4 Arquitectura basada en componentes …………………………………………… 15

2.3 Metodologías para el desarrollo de videojuegos …………………………………... 16

2.4 Metodología ágil para el desarrollo de este proyecto: SUM ……………………… 17

2.4.1 Introducción a la Metodología ........................................................................... 18

2.4.2 Roles ................................................................................................................ 19

2.4.3 Ciclo de Vida .................................................................................................... 20

2.4.4 Concepto .......................................................................................................... 21

2.4.5 Planificación ..................................................................................................... 22

2.4.6 Elaboración ...................................................................................................... 23

2.4.7 Beta .................................................................................................................. 24

2.4.8 Cierre ............................................................................................................... 25

2.4.9 Gestión de Riesgos .......................................................................................... 26

2.4.10 Guías .............................................................................................................. 27

Capítulo 3: Adaptación de la Metodología al Proyecto ................................... 28

3.1 Fase 1: Concepto ................................................................................................ 29

3.1.1 Herramientas .................................................................................................... 30

3.1.2 Elementos de Diseño ........................................................................................ 31

3.2 Fase 2: Planificación ............................................................................................ 47

3.2.1 Planificación Administrativa .............................................................................. 47

3.2.2 Especificación del Videojuego .......................................................................... 47

3.3 Fase 3: Elaboración ............................................................................................. 48

3.3.1 Instalación de Unity y Vuforia ............................................................................ 48

3.3.2 Interconexión de Escenas o Niveles ................................................................. 50

3.3.3 Personaje ......................................................................................................... 63

3.3.4 Enemigos ......................................................................................................... 70

3.3.5 Personaje vs Enemigos .................................................................................... 75

3.3.6 Personaje y la UI .............................................................................................. 79

3.3.7 Ítems ................................................................................................................ 82

3.3.8 Escenarios ....................................................................................................... 90

3.4 Fase 4: Beta ........................................................................................................ 91

3.5 Fase 5: Cierre ...................................................................................................... 92

Capítulo 4: Especificación de Requisitos de Software ................................... 99

4.1 Análisis de Requisitos del Sistema ..................................................................... 99

4.2 Identificación de los Usuarios Participantes ........................................................ 99

4.3 Catálogo de Requisitos del Sistema .................................................................... 99

4.4 Objetivos y Alcance del Sistema ......................................................................... 99

4.5 Definiciones, Acrónimos y Abreviaturas............................................................. 100

4.6 Descripción General .......................................................................................... 100

4.7 Requisitos Funcionales ..................................................................................... 101

4.8 Suposiciones y Dependencias ........................................................................... 101

4.9 Requisitos de Usuario y Tecnológicos ............................................................... 102

4.10 Requisitos de Interfaces Externas ................................................................... 102

4.11 Requisitos de Rendimiento .............................................................................. 102

4.12 Requisitos de Desarrollo .................................................................................. 103

4.13 Restricciones de Diseño .................................................................................. 103

Capítulo 5: Selección del Modelo de Ciclo de Vida y Gestión del Proyecto ..................................................................................................................... 104

5.1 Introducción ....................................................................................................... 104

5.2 Selección de Metodología de Desarrollo de Software ....................................... 104

5.3 Selección de un Modelo de Ciclo de Vida.......................................................... 106

Capítulo 6: Gestión del Proyecto .................................................................. 108

6.1 Introducción ....................................................................................................... 108

6.2 Procedimiento ................................................................................................... 108

6.3 Planificación del Proyecto .................................................................................. 113

6.3.1 Introducción .................................................................................................... 113

6.3.2 Plan del Desarrollo ......................................................................................... 113

Capítulo 7: Modelización y Diseño ............................................................... 115

7.1 Actores .............................................................................................................. 115

7.2 Diagrama de Contexto ....................................................................................... 115

7.3 Listado de Casos de Uso ................................................................................... 115

7.4 Diagrama de Casos de Uso ............................................................................... 115

7.5 Descripción Textual de los Casos de Uso ......................................................... 116

7.6 Diagrama de Actividad ....................................................................................... 117

7.7 Diagrama de Secuencia .................................................................................... 118

7.8 Identificación de Roles ....................................................................................... 119

7.9 Guiones y Escenarios: Diagrama de Transición de Escenarios ........................ 119

7.10 Tabla de Transición de Escenarios .................................................................. 120

7.11 Descripción de Escenarios y Objetos de Escenarios ...................................... 120

Capítulo 8: Implementación y Pruebas ......................................................... 123

8.1 Herramienta Principal para el Desarrollo de la Aplicación ................................ 123

8.1.1 Introducción .................................................................................................... 123

8.2 Desarrollo de la Aplicación ................................................................................ 134

Observaciones ............................................................................................. 144

Conclusiones ............................................................................................... 147

Bibliografía ................................................................................................... 148

# Capítulo 1 - Introducción

## 1.1 Motivación del proyecto

Hoy en día los videojuegos son considerados una de las principales industrias del arte y entretenimiento, estando al nivel o por encima de las industrias del cine y música juntas. El crecimiento del desarrollo *indie* o independiente en los últimos años ha crecido de manera exponencial, principalmente debido a las nuevas herramientas de desarrollo.

Ser desarrollador independiente de videojuegos significa no tener ataduras con lo establecido y luchar por ideas innovadoras que además pueden terminar formando parte de la gran industria. En la actualidad existen muchos desarrolladores independientes que presentaron formulas en la experiencia de la vivencia de un videojuego que no fueron mostradas aún, y terminaron consolidando una nueva forma de diseñarlos.

Por otra parte, dada la circunstancia en la que se encuentra el mundo actualmente, hubo un crecimiento en desarrollo de videojuegos que tienen como temática central a una pandemia o un virus maligno. Es por esa razón que Eggy Runner toma como idea principal combatir al Covid-19.

## 1.2 Objetivos

* Objetivo General: Realizar un videojuego para computadora de escritorio que fomente la competitividad y mejore el uso de reflejos de los jugadores, en una realidad cuyo objetivo sea cuidarse del Covid-19.
* Objetivos Específicos:
* Entender qué es un videojuego y los géneros que existen con sus características, las partes de las que se compone y cómo es usada en cada una de ellas.
* Conocer el proceso de diseño de un videojuego.
* Aplicar en el juego conceptos aprendidos en la carrera, tanto los vistos en asignaturas del ciclo básico como las del ciclo superior.
* Implementar una base de datos en línea que permita acceder a todos los usuarios a información acerca de partidas de otros jugadores.

## 1.3 Concepto de videojuegos

Cuando hablamos de videojuegos, podemos referirnos a los mismos como una experiencia interactiva, o como una herramienta artística o un medio audiovisual más para contar historias.

Sin embargo, las diferentes definiciones hacia estos tienen ciertos aspectos en común:



Fig. 1.1 – Diagrama representativo de las partes de un videojuego

* Un videojuego es un juego que se reproduce a través de un medio electrónico (plataforma)
* Los usuarios (jugadores) se comunican con ese medio a través de una interfaz.
* La cual puede estar compuesta por una pantalla (*output*) y un controlador (*input*) (teclado, mouse, joystick, etc.).
* Los comandos que ingresan por los mismos provocan cambios en el juego (acciones).
* Y el juego comunica estos cambios de estado mediante un *feedback* (mostrar algo por pantalla, sonido, vibrar el mando, etc.).

## 1.4 Géneros de videojuegos

El género de videojuego es una forma de clasificar un videojuego en función fundamentalmente de su mecánica de juego. La mecánica de juego es una regla o conjunto de reglas cuyo objetivo consiste en obtener una serie de resultados coherentes en el seno de un juego.

Existen muchos géneros: juegos de acción, aventura, *survival horror*, RPG *(Role Playing Games)*, estrategia, deportivos, entre otros.

Para este informe se hará foco en dos géneros: juegos de plataforma y *Endless Runner*.

### 1.4.1 Juegos de plataforma

Los videojuegos de plataformas o, simplemente, plataformas, son un género de videojuegos en los que el jugador tiene que saltar sobre plataformas suspendidas o a lo largo de obstáculos y enemigos hasta llegar a una meta. Se considera la presencia de un botón o función de saltar como parte central de la forma de juego de este género, aunque en algunos casos la capacidad de saltar del personaje es reemplazada por otras similares como escalar con ganchos o utilizar resortes. Es muy común que incluyan también elementos de otros géneros como disparos, *beat 'em up* o aventura.

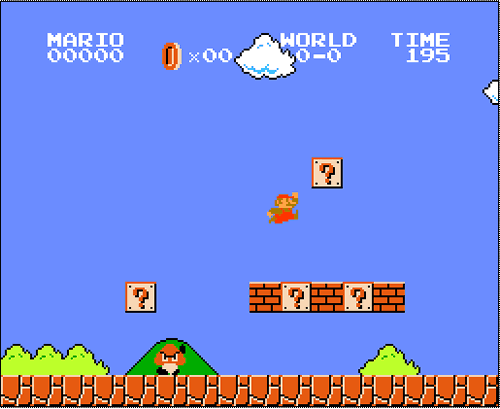


Fig. 1.2 – Gameplay del juego Super Mario Bros

### 1.4.2 Endless Runner

Los juegos Endless Runner son un subgénero de videojuegos estilo plataforma, donde se rigen fundamentalmente por dos características principales:

* El personaje protagonista avanza sin parar, su impulso es ir siempre hacia adelante.
* El escenario nunca termina.

Cuanto más se avanza en la partida, mayores son la velocidad y los premios que se consiguen. El avatar controlado por el jugador también va cada vez más rápido; pasado un tiempo es casi incontrolable. Es interesante señalar que el progreso en estos juegos se mide en función de la distancia recorrida, con el objeto de llegar cada vez más lejos evitando toda clase de obstáculos.



Fig. 1.3 – Gameplay del juego Temple Run

## 1.5 Partes de un videojuego

Se puede resumir las partes de todo videojuego en las siguientes tres:

* Motor
* *Assets* (texturas, mallas, sonidos, animaciones, etc.)
* *Scripts*

### 1.5.1 Motor

Los motores de videojuegos suelen proporcionar un conjunto de herramientas de desarrollo visual y componentes de software que puedan ser reutilizables. Estas herramientas generalmente se proporcionan en un entorno de desarrollo integrado que permiten crear videojuegos de forma rápida y simple a través de una base de datos.​ En otros casos, los motores se distribuyen con una interfaz de programación de aplicaciones (API) incorporada;​ y otros motores, sin embargo, se distribuyen como un conjunto de herramientas que agilizan y simplifican aún más el desarrollo de un videojuego, como por ejemplo: los entornos de desarrollo integrado, scripts pre-programados, y los middlewares (capaces de interconectar varios programas). Esto resulta útil a la hora de conseguir una plataforma de software flexible y reutilizable que evite la compra de otros recursos ajenos, lo cual ayuda a tener todo lo necesario para hacerlo funcional de manera inmediata, reducir los costos, complejidades y tiempos de comercialización.



Fig. 1.4 – Los motores gráficos más icónicos de los videojuegos

Un motor de videojuego se puede dividir en dos grandes categorías: motor gráfico y motor físico. Los motores gráficos tratan el aspecto visual del videojuego, que generan imágenes sintéticas integrando o cambiando información visual y espacial. Los motores físicos se ocupan de integrar las leyes de la física, siendo responsables de simular acciones reales, a través de variables como la gravedad, la masa, la fricción, la fuerza y la flexibilidad. ​

A pesar de la especificidad del nombre, los motores de videojuego también se utilizan para crear otros tipos de aplicaciones interactivas denominadas como "juegos serios",​ tales como visualizaciones arquitectónicas, educación, ​​ avances científicos, ​ simulaciones de entrenamiento, ​herramientas de modelado​ y simulaciones físicas para recrear animaciones.

### 1.5.2 Assets

Son los distintos elementos que realizan los artistas del equipo de trabajo:

* Mallas: son objetos 3D expresados como secuencia de puntos 3D.
* Texturas: son imágenes 2D que se pintan sobre los objetos 3D.
* Animaciones: son secuencias de puntos 3D que representan el movimiento de la malla.
* Sonidos: son los efectos especiales que responden a un evento o un lugar específico.

Cada objeto 3D de un videojuego está modelado con un programa especial para ello (Maya, 3D Studio Max, etc.). Un modelo 3D es un grupo de puntos en el espacio unidos por líneas, conformando una “malla”. Esta malla está compuesta por triángulos. Mientras más triángulos tenga un modelo, más costará pintarlo en pantalla, y será más carga para el CPU. Además de la "malla“, un objeto 3D está compuesto por un material que contiene la textura del modelo, y otra serie de componentes como el mapa de iluminación, normales, etc. Las animaciones son copias de la malla en distintas posiciones emulando un movimiento.

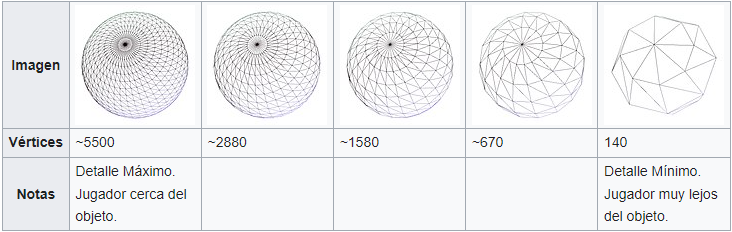


Fig. 1.5 – Nivel de detalle y distancia de dibujado de una malla tridimensional

### 1.5.3 Scripts

Lenguaje de programación de alto nivel, de codificación simple, con pocas funciones (todas ellas relativas al motor del juego). Utilizado para codificar los eventos del juego. Los scripts se pueden usar para hacer todo o casi todo en un videojuego. Se pueden crear objetos dinámicamente, destruirlos, añadirles componentes, quitárselos o crear comportamientos (tanto para el objeto, como para cada uno de sus componentes).

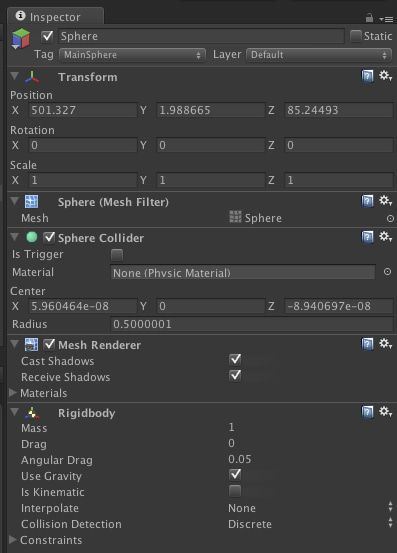


Fig. 1.6 – Componentes de un objeto en Unity

# Capítulo 2 – Diseño de un videojuego

## 2.1 Introducción

Los primeros videojuegos no mostraban más que unos puntos, líneas o figuras geométricas en la pantalla y se controlaban mediante el uso de simples potenciómetros o pulsadores. Sin embargo, hoy en día, muestran gráficos 3D extremadamente realistas y, los modos de control, son tan abundantes como diversos, yendo desde teclado o mandos hasta reconocimiento de movimiento mediante el uso de cámaras.

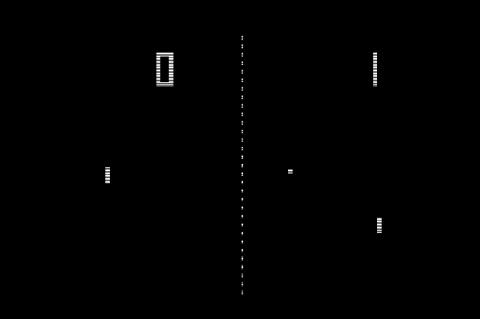
 

Fig. 2.1 – Gameplay del juego Pong (1972) vs God of War: Ragnarok (2022)

En los inicios de los videojuegos, uno o dos programadores se encargaban de todo el proceso de desarrollo mientras que hoy en día, existe a una gran cantidad de gente que cumple diferentes roles dentro del desarrollo, hablando de empresas desarrolladoras de medio y gran tamaño. Debido a este fenómeno de especialización de equipo, ahora hay diferentes roles dentro de un desarrollo, donde ya no existen solamente programadores que se encarguen de todas las tareas (al menos en desarrollos medios y grandes), sino que ahora se tiene un equipo multidisciplinar de personas que provienen de áreas muy diferentes. Estos roles que se necesitan varían dependiendo del proyecto, sin embargo, se puede considerar que hay tres roles básicos que aparecen siempre en el proceso de producción: diseñador, artista y programador.

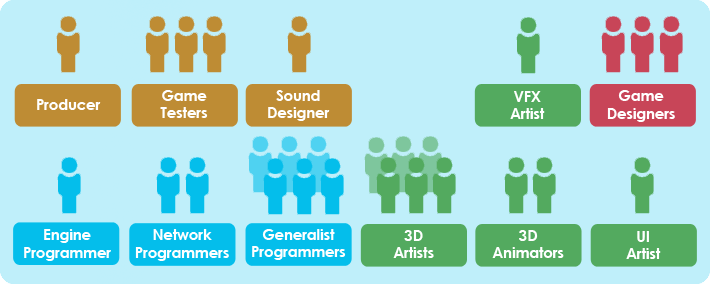


Fig. 2.2 – Roles en la creación de un videojuego

* Los diseñadores son la cabeza pensante del proyecto, son los que deciden el rumbo que éste debe llevar y cuál es el resultado que se desea obtener. Están involucrados desde el principio al fin del proyecto, siendo los encargados de desarrollar la idea inicial del juego y pensar en hasta el último detalle del producto que se quiere obtener, controlando que realmente el rumbo sea el adecuado. Entre sus responsabilidades, aparte de decidir en qué va a consistir el juego, está establecer cuáles van a ser las mecánicas principales de éste, las reglas, cómo va a interaccionar el jugador, qué respuesta visual y sonora va a obtener, cómo van a ser los personajes, cuál va a ser la historia del juego, guion, diálogos o diseñar los diferentes niveles que se tendrán. Una de las tareas más importantes del diseñador es saber transmitir exactamente qué es lo que quiere de cada miembro del equipo y de plasmar perfectamente cuáles son los requisitos.



Fig. 2.3 – Equipo diseñador

* Los artistas son los responsables de la creación de todos los recursos gráficos del juego. Su trabajo es crear y texturizar los modelos de los personajes con sus correspondientes animaciones, elementos del mundo, niveles y todo tipo de contenido 3D, así como realizar las imágenes usadas en el juego para los menús y la interfaz de usuario.



Fig. 2.4 – Equipo artístico

* La responsabilidad del programador es crear todo el código necesario para hacer que todas las ideas descritas por los diseñadores se hagan realidad y funcionen usando los recursos que han generado los artistas. Los programadores están involucrados en todos los aspectos del juego: gráficos, animaciones, lógica de juego, gameplay (jugabilidad), interfaz de usuario, física, sonido, scripts, etc. Los programadores suelen ser ingenieros informáticos y, debido a que están implicados en todos los aspectos del juego y son los que saben acerca de las limitaciones técnicas existentes, deben evaluar los riesgos técnicos del diseño del juego y acordar con los diseñadores qué cosas se pueden hacer y qué cosas no son viables y deben ser rediseñadas. De la misma manera, son los encargados de poner restricciones técnicas a los recursos que generan los artistas y que deben ser incluidos en el juego.



Fig. 2.5 – Equipo programador

Debido a estas cuestiones planteadas, las metodologías clásicas de desarrollo software quedan obsoletas, por lo cual se debe de optar por metodologías actuales, más flexibles y que asimilen relativamente bien cambios en la especificación, primando la velocidad de desarrollo frente a la sobre-ingeniería. La sobre-ingeniería, a grandes rasgos, es el hecho de agregar diversas opciones o funcionalidades a un producto, con la finalidad de que se utilicen en un futuro. En el mundo de la programación, la sobre-ingeniería implica tener un montón de código, que solo hace más pesado el programa, y que además enreda a los desarrolladores, pero que está ahí porque quizá en algún momento se llegue a utilizar, cosa que rara vez pasa.

El desarrollo de mecánicas o características de un juego suele ser un proceso iterativo donde las ideas que van surgiendo se documentan, se implementan, se evalúan y, en función de esa evaluación, se descartan, se aceptan o se refinan, volviendo a pasar de nuevo por todas las fases anteriores. Con el paso de los años la tecnología usada para soportar este tipo de metodologías ha ido evolucionando y, hoy en día, prácticamente todos los juegos del mercado usan una arquitectura basada en componentes para implementar las diferentes entidades u objetos del juego. Este tipo de tecnología es muy flexible, permitiendo todo tipo de modificaciones en las especificaciones y premiando la reusabilidad de código. Estas arquitecturas están basadas en composición en lugar de la herencia, la funcionalidad de las entidades del juego se encuentra fraccionada y repartida por varios componentes software. Esto aporta velocidad a la hora de formar entidades nuevas o modificar ya creadas y ahorra grandes tiempos de compilación, pero dificulta también la comprensión a alto nivel de qué es cada entidad, ya que se limita a un conjunto de componentes.

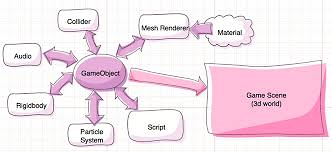


Fig. 2.6 – Conjunto de posibles componentes que pertenecen a un GameObject (entidad)

## 2.2 Arquitectura software de videojuegos

### 2.2.1 Arquitectura software general

Los videojuegos son aplicaciones de tiempo real grandes y complejas que usan una gran cantidad de recursos multimedia, deben proporcionar una experiencia inmersiva y, lo más importante, deben de ser divertidos. Esa búsqueda de la diversión es un proceso iterativo de prueba y error que va desde las primeras etapas de prototipado a las últimas en las que se balancean las mecánicas de juego. Desde el punto de vista de la ingeniería del software, esta situación es muy poco deseable ya que las especificaciones van evolucionando hasta el día en que se entrega el producto final. Aunque hay muchos tipos y complejas descripciones de arquitecturas de videojuego, en su forma más simple se pueden considerar dos grandes bloques:

* *Frontend*: Responsable de retroalimentar la experiencia del jugador procesando los comandos de entrada del usuario (teclado, ratón, etc.) y presentando el juego (motor). Es decir, cargar los recursos externos, procesar la entrada de usuario y presentarle los resultados.
* *Backend*: Encargado de la lógica de juego que especifica las reglas, determina la dinámica de interacción con los objetos, proporciona comportamientos a los personajes no jugadores (*NPC*), las interacciones entre ellos, etc. Es decir, personalizar el juego.

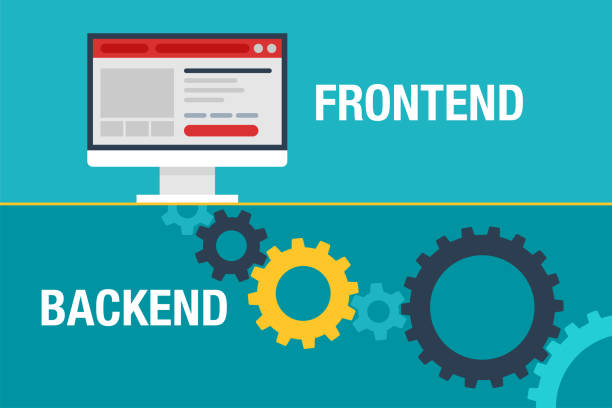


Fig. 2.6 – Frontend vs Backend

### 2.2.2 Arquitectura software para las entidades de juego

Las entidades de juego son piezas de lógica auto contenidas, o piezas de contenido interactivo, que pueden llevar a cabo diferentes tareas como renderizarse a sí mismas, encontrar o seguir caminos, tomar decisiones, etc. Por ejemplo, los NPCs (personajes no jugables), puntos de ruta o disparadores (*triggers*) que controlan la historia del juego.

Sin embargo, estas entidades necesitan un sistema de gestión de éstas para saber cuáles están activas, cuáles se deben crear o destruir, etc. Este módulo forma parte del núcleo de un juego y, debido a su tamaño y complejidad, requiere de esfuerzo en el desarrollo. Ejemplos:

* El juego Half-Life que data de 1999 tiene más de 65,000 líneas de código en este módulo (sin contabilizar líneas vacías ni comentarios).
* El juego Far Cry de 2004 excede las 95,000 líneas de C/C++ incluso teniendo en cuenta que la mayoría del módulo estaba escrito en LUA.
* En 2006, Gears of War tenía 250,000 líneas de C++ y scripts.

Fig. 2.7 – Gameplay de Half-Life y Gears of War

Debido al tamaño, la complejidad y a su naturaleza de especificación cambiante, se hace necesaria una arquitectura altamente flexible y extensible. Al principio de los años 2000, cuando los lenguajes orientados a objetos se fueron imponiendo en el desarrollo de videojuegos, las entidades de juego se organizaban en jerarquías basadas en herencia donde, por ejemplo, jugadores y enemigos son personajes y tanto los personajes como los objetos son entidades. Este tipo de estructura se demostró rápidamente que era rígida y difícil de mantener (se explicará a continuación).

### 2.2.3 Jerarquías de clases

Durante el diseño de un juego es habitual categorizar las entidades mediante el uso de una ontología que modela la relación es-un entre todas ellas. Este conocimiento puede ser usado en la fase de desarrollo mediante herencia en el código del juego por lo que no sorprende que la gestión de entidades tradicional se basase en herencia donde las entidades se implementan como subclases que derivan de la misma clase base.

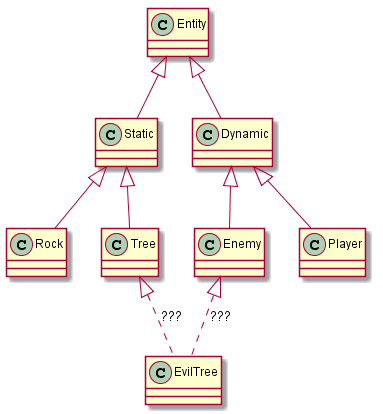


Fig. 2.8 – Árbol de herencia tradicional

En función de la complejidad del juego, las jerarquías pueden ser bastante profundas, con clases abstractas intermedias que agregan funcionalidad común para incrementar la reusabilidad del código. Si se observa la Figura 2.8, en ella se muestra una jerarquía parcial hipotética, donde los nodos hoja representan clases concretas mientras que los nodos internos son clases que pueden no haber sido concebidas durante la fase de diseño, pero creadas para la reutilización de código.

Esta jerarquía trae varias decisiones implícitas debido a la manera en que se parten las clases cuando se desciende por la jerarquía. Estas decisiones pueden volverse problemáticas a la larga, sobre todo en un entorno tan cambiante como el desarrollo de videojuegos.

En la base de la Figura 2.8 se puede observar la clase *EvilTree*, la cual compartiría cualidades y atributos similares tanto a *Tree* como a *Enemy*, es decir que su solución obvia sería que heredase de ambas. Sin embargo, la herencia múltiple no es soportada por todos los lenguajes orientados a objetos, y aunque lo fueren, provocarían herencia en diamante ya que hereda dos veces de *Entity*, lo cual genera ambigüedad, confusión y una complejidad no deseable. La solución a esto conlleva a superclases repletas de funcionalidades y subclases que se encargan de activar o desactivarlas, lo que constituye un diseño orientado a objetos bastante pobre.

Algunas de las consecuencias de las superclases gigantes son el incremento del tiempo de compilación, código difícil de entender y el conocido *fragile base class problem*, el cual provoca que cambios aparentemente pequeños en la clase base tiene un impacto negativo en las subclases. Todos estos problemas no solo surgen en el desarrollo de videojuegos, pero en este campo se acentúan debido a los constantes cambios en la especificación, lo que fuerza a los programadores a revisar y extender una arquitectura bastante estática como es la jerarquía de clases.

### 2.2.4 Arquitectura basada en componentes

La arquitectura basada en componentes es la tecnología que se usa hoy en día para la implementación del módulo encargado de las entidades de juego. Un componente en esta arquitectura representa una pequeña porción de funcionalidad idealmente auto-contenida que puede ser agregada o eliminada de una entidad de juego. Por tanto, un nuevo tipo de entidad es simplemente una combinación de componentes que pueden ser creados en tiempo de ejecución, lo que aporta flexibilidad y potencia la reutilización.

El mayor problema de la jerarquía de clases es su naturaleza estática, ya que los continuos cambios de diseño provocan que la herencia no sea la mejor solución. Por esta razón, los desarrolladores hoy en día tienden a usar sistemas basados en componentes. En lugar de tener entidades de una clase concreta que defina su comportamiento, ahora cada entidad es simplemente un contenedor donde cada funcionalidad o habilidad que tenga la entidad es implementada por un componente. Mediante el uso de estas técnicas se cambia una jerarquía de clases estática creada por mecanismos de herencia por un diseño versátil que permite crear nuevas entidades de manera sencilla por combinación de componentes. Ahora las entidades son listas de componentes y los componentes concretos que conforman una entidad no tienen por qué estar descritos en el código fuente, sino que pueden especificarse en un fichero externo que se procesa en tiempo de ejecución, haciendo que la creación de las entidades esté dirigida por datos. De alguna manera esta aproximación simplifica la creación de nuevos tipos de entidades, ya que no se requiere ninguna tarea de programación sino simplemente seleccionar las diferentes habilidades requeridas para la entidad de un conjunto de componentes.

Motores de juego más sofisticados suelen ser más versátiles aún, permitiendo del uso de lenguajes de scripts para la creación de componentes. El motor CryEngine permite el uso de LUA para la creación de scripts mientras que Unity permite el uso de Unity Script y C#. El uso de este tipo de motores permite a los desarrolladores crear componentes de una manera más rápida. Finalmente cabe resaltar que prácticamente todos los juegos modernos se crean con una arquitectura basada en componentes, la cual ha pasado a ser la principal solución frente a los sistemas basados en herencia.

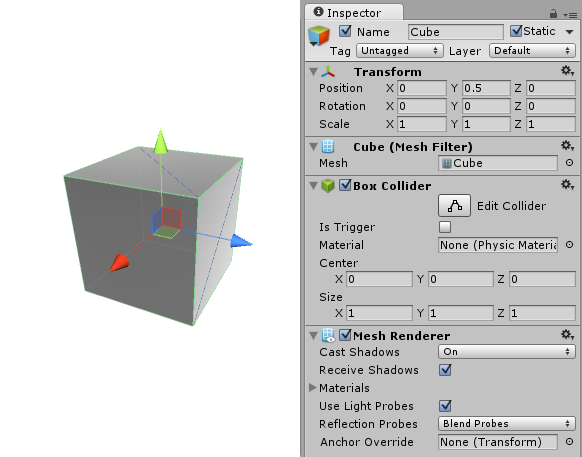


Fig. 2.9 – Entidad cubo con diversos componentes conformándolo

### 2.3 Metodologías para el desarrollo de videojuegos

El proceso de producción de un videojuego es todo el camino que recorre un equipo para transformar una idea en un producto software cerrado listo para ser distribuido. El desarrollo de videojuegos cubre un amplio espectro de desarrollos ya que dentro de este campo caben grandes proyectos de juegos donde trabajan mas de 80 personas durante 2 o tres años, y pequeños desarrollos para móviles, donde grupos pequeños de 4 o 5 personas desarrollan un producto en apenas un par de meses. Este proceso obviamente varía en función del tipo de proyecto que se lleve a cabo, del grupo que lo desarrolle, de los recursos que se tengan, de la plataforma objetivo para la que se desarrolla, de la tecnología elegida etc.

Uno de los grandes problemas de la creación de videojuegos es que, dado que el equipo de desarrollo es multidisciplinar y que es necesario generar funcionalidad jugable lo antes posible, es muy difícil planificar correctamente. Debido al comportamiento emergente del desarrollo de videojuegos, cada vez menos estudios siguen un desarrollo clásico en cascada, donde el equipo de desarrollo crea varios componentes de juego independientes que se unen al final del desarrollo confiando en que todo irá según lo esperado.

Hoy en día, el desarrollo software en general está evolucionando, de manera que el proceso de creación de un producto es cada vez más iterativo. En este proceso cíclico se conciben nuevas ideas o mecánicas, se documentan, se implementan, se evalúan a ver si cumplen los propósitos esperados y en función de eso se descartan, aceptan o se refinan dichas ideas volviendo a empezar el ciclo. Es por eso que, desde hace tiempo, cada vez más desarrolladores de software en general, y de videojuegos en particular, están adoptando metodologías ágiles como “Scrum” o “Extreme Programming” que están enfocadas en realizar un producto software mediante iteraciones.

Las metodologías ágiles pretenden que se genere nuevo software funcional iterativamente cada poco tiempo, donde las iteraciones pueden ir desde una semana a un par de meses, de manera que se trata de maximizar la simplicidad del software realizado, implementando solo lo que se necesita. Esto incentiva a que se mantenga un ritmo de desarrollo constante y que sea más sencillo medir cuál es el progreso del desarrollo. Estos tipos de metodologías son muy flexibles a los cambios en el diseño y permiten un diseño incremental de mejora constante gracias a que se debe usar tecnología flexible como la arquitectura basada en componentes.

Una de las bases de las metodologías ágiles es que todo el mundo puede mejorar cualquier parte del desarrollo cuando lo necesite, y para esto se debe obligar a que haya integración continua de contenidos, donde el trabajo de uno se debe integrar varias veces al día, evitando así grandes conflictos que pueden surgir cuando se trata de juntar dos versiones desarrolladas por separado.

Los principales beneficios que reportan los casos de éxito al utilizar metodologías ágiles son:

 Al ser metodologías iterativas e incrementales se obtienen versiones jugables del producto en intervalos regulares de tiempo. Esto facilita una visión temprana del resultado final del juego, lo cual reduce la probabilidad de cambios de requerimientos en forma tardía y brinda una mayor retroalimentación del cliente.

 Permiten tener una mayor visión y control del avance del proyecto, tanto al cliente como a los desarrolladores. Esto se debe a que se pueden determinar nuevas estrategias, iteración por iteración, para lograr llegar en tiempo y forma a los plazos requeridos.

 Involucran a todo el equipo en las decisiones, lo que logra compromiso y motivación.

2.4 Metodología ágil para el desarrollo de este proyecto: SUM ……………………… 17

2.4.1 Introducción a la Metodología ........................................................................... 18

2.4.2 Roles ................................................................................................................ 19

2.4.3 Ciclo de Vida .................................................................................................... 20

2.4.4 Concepto .......................................................................................................... 21

2.4.5 Planificación ..................................................................................................... 22

2.4.6 Elaboración ...................................................................................................... 23

2.4.7 Beta .................................................................................................................. 24

2.4.8 Cierre ............................................................................................................... 25

2.4.9 Gestión de Riesgos .......................................................................................... 26

2.4.10 Guías .............................................................................................................. 27