



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

GRADO EN INGENIERÍA INFORMÁTICA

**TECNOLOGÍA ESPECÍFICA DE
COMPUTACIÓN**

TRABAJO FIN DE GRADO

**Virus Breaker
Desarrollo de un videojuego con Unity3D**

Pedro Romero González

Febrero, 2018

VIRUS BREAKER
DESARROLLO DE UN VIDEOJUEGO CON UNITY3D



**UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA**

Tecnologías y Sistemas de Información

**TECNOLOGÍA ESPECÍFICA DE
COMPUTACIÓN**

TRABAJO FIN DE GRADO

**Virus Breaker
Desarrollo de un videojuego con Unity3D**

Autor: Pedro Romero González

Director: Dr. David Vallejo Fernández

Febrero, 2018

Pedro Romero González

Ciudad Real – España

E-mail: pedro9romero4gonzalez@gmail.com

Teléfono: 689 426 432

Web site: <https://gamejolt.com/@nexus64>

© 2018 Pedro Romero González

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Se permite la copia, distribución y/o modificación de este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.3 o cualquier versión posterior publicada por la *Free Software Foundation*; sin secciones invariantes. Una copia de esta licencia esta incluida en el apéndice titulado «GNU Free Documentation License».

Muchos de los nombres usados por las compañías para diferenciar sus productos y servicios son reclamados como marcas registradas. Allí donde estos nombres aparezcan en este documento, y cuando el autor haya sido informado de esas marcas registradas, los nombres estarán escritos en mayúsculas o como nombres propios.

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE

VOCAL

SECRETARIO

Fdo.:

Fdo.:

Fdo.:

Resumen

En este proyecto se ha realizado el desarrollo de un videojuego completo para PC utilizando el motor de juegos Unity3D. El juego es una reinvención del clásico *Breakout*, adaptándolo a las tres dimensiones mediante cambios en su diseño.

Esta memoria describe la situación socioeconómica del mercado del videojuego tanto mundial como nacional a modo de justificación del desarrollo del proyecto, citando entre otros factores el enorme crecimiento que está teniendo esta industria. A continuación, se realizará un análisis del proceso de desarrollo de un videojuego, y una descripción de la estructura de equipo de desarrollo de videojuegos. También realizaremos un estudio del uso de la inteligencia artificial en el campo del videojuego.

La memoria también describe en detalle la implementación de este proyecto, describiendo la estructura de su código y explicando los patrones de diseño en los que se basa el motor. Acompañando al análisis general, se realizará una revisión de los puntos más destacados del proyecto como las mecánicas de control del personaje principal, el sistema de carga de niveles y la implementación del jefe final del juego.

Finalmente, el documento cerrará con una revisión del resultado final del proyecto, junto con una lista de posibles mejoras para futuras versiones.

Índice general

Resumen	v
Índice general	vii
Índice de figuras	ix
1. Introducción	1
1.1. Motivación	1
1.2. Diseño de juego	2
1.2.1. Concepto	2
1.2.2. Jugabilidad	2
1.2.3. Estilo y Ambientación	4
1.2.4. Plataforma	5
2. Estado del arte	7
2.1. El mercado de los videojuegos	7
2.1.1. Industria mundial del Videojuego	7
2.1.2. La industria de los videojuegos en España	8
2.1.3. Retos y tendencias actuales	9
2.2. El proceso de desarrollo de videojuegos	17
2.2.1. Introducción	17
2.2.2. Etapas del desarrollo	18
2.2.3. Estructura típica de un equipo de desarrollo	21
2.3. Motores de juegos	28
2.3.1. Descripción	28
2.3.2. Ejemplos de Motores	30
2.4. Inteligencia Artificial en Videojuegos	35
2.4.1. Historia	35
2.4.2. IA Aplicada a Videojuegos: Contexto Actual	36
2.4.3. Métodos de la IA	38

0. ÍNDICE GENERAL	
2.4.4. Futuros campos de aplicación	41
3. Arquitectura	43
3.1. Estructura de clases del Proyecto	43
3.1.1. Escenas	43
3.1.2. GameObjects, Componentes y Prefabs	44
3.1.3. Estructura de Clases	46
3.1.4. Prefabs	47
3.2. Carga y Estructura de Niveles	54
3.3. Control de movimiento y físicas	56
3.3.1. Personaje principal	56
3.3.2. Pelota	59
3.4. Jefe Final	61
3.4.1. Implementación	62
4. Conclusiones	65
4.1. Resultados	65
4.2. Mejoras Futuras	66
4.2.1. Escenarios	67
4.2.2. Enemigos	68
4.2.3. Sistemas Complementarios	69
Bibliografía	71
Videojuegos	74

Índice de figuras

1.1.	Breakout (Atari, 1976)	2
1.2.	Relación entre desafío y la ansiedad/aburrimiento.	4
1.3.	El estilo del juego está influenciado por títulos como Megaman Legends(Capcom, 1997)	5
1.4.	Gamejolt	6
1.5.	Itch.io	6
1.6.	Juegos desarrollados con Unity3D	6
2.1.	Crecimiento Mundial de la industria del videojuego (por plataformas)	7
2.2.	Distribución por regiones del mercado del videojuego.	8
2.3.	Diez principales mercados del videojuego.	9
2.4.	Mercado del Videojuego en Europa.	10
2.5.	Distribución de las empresas en España por número de empleados.	11
2.6.	Distribución de las empresas en España por número de empleados.	11
2.7.	Fotografía del torneo Intel® Extreme Masters en Katowice, Polonia	12
2.8.	Crecimiento del mercado del eSport	12
2.9.	<i>League of Legends</i> (Riot Games, 2009), uno de los eSports más populares	13
2.10.	Previsiones de crecimiento del mercado de Realidad Virtual y Aumentada (miles de millones de dólares))	14
2.11.	De izquierda a derecha HTC Vive, Oculus Rift, Samsung VR y PlayStation VR	15
2.12.	Áreas de la Industria 4.0	16
2.13.	Impresora 3d “replicator” de la compañía Makerbot.	17
2.14.	Etapas del desarrollo de un videojuego.	18
2.15.	Distribución de roles en un equipo de desarrollo.	22
2.16.	Shigeru Miyamoto (creador de <i>Super Mario Bros.</i> (Nintendo, 1985), <i>The Legend of Zelda</i> (Nintendo, 1986) y <i>Donkey Kong</i> (Nintendo, 1981)) es posiblemente el diseñador de videjouuegos más famoso.	23
2.17.	Jonh Romero es el co-fundador de ID Software y programador de juegos como <i>Doom</i> (id Software, 1993) o <i>Quake</i> (id Software, 1996).	24

0. ÍNDICE DE FIGURAS

2.18. Akira Toriyama, el autor de Dragon Ball, ha trabajado como artista en juegos como <i>Dragon Quest</i> (Chunsoft, 1986) o <i>Chrono Trigger</i> (Square, 1995).	25
2.19. Yoko Shinomura es una compositora conocida por su trabajo en videojuegos como <i>Final Fantasy XV</i> (Square Enix, 2016) o <i>Street Fighter II: The World Warrior</i> (Capcom, 1991).	26
2.20. <i>Heretic</i> (Raven Software, 1994), es un juego desarrollado con el motor de Doom.	28
2.21. Captura del entorno de desarrollo de Unity	31
2.22. <i>A Hat in Time</i> (Gears for Breakfast, 2017)	32
2.23. <i>Hollow Knight</i> (Team Cherry, 2017)	32
2.24. Juegos desarrollados con Unity3D	32
2.25. Captura del entorno de Unreal Engine	32
2.26. <i>Unreal</i> (Epic Games, 1998)	33
2.27. <i>Batman: Arkham Knight</i> (Rocksteady Studios, 2015)	33
2.28. Juegos desarrollados con Unreal Engine	33
2.29. Captura del entorno de Game Maker Studio	34
2.30. <i>Undertale</i> (Toby Fox, 2015)	34
2.31. <i>Hyper Light Drifter</i> (Heart Machine, 2016)	34
2.32. Juegos desarrollados con Game Maker	34
2.33. El Gran Maestro de ajedrez Garri Kaspárov (izquierda) enfrentándose al ordenador Deep Blue	36
2.34. En <i>The Sims</i> (Maxis, 2000), los personajes pueden tomar decisiones basándose en sus gustos y necesidades.	37
2.35. <i>Minecraft</i> (Mojang, 2011) es un ejemplo claro de generación procedimental de terrenos.	38
2.36. Ejemplo de comportamiento tóxico en <i>League of Legends</i> (Riot Games, 2009).	42
3.1. Diagrama de las escenas del juego	44
3.2. Inspector de objetos mostrando los componentes del objeto <i>Paddle</i> . . .	45
3.3. Estructura de clases	47
3.4. Lista de GameObjects en la escena principal	49
3.5. Prefabs de la escena Principal	52
3.6. Lista de GameObjects en los menús	53
3.7. Homenaje a Space Invaders(Taito, 1978) dentro de Arkanoid: Doh it Again (Taito, 1997)	54
3.8. Modelo del personaje principal	57

3.9. Diagrama de estados del personaje principal	58
3.10. Personaje principal activando la paleta	59
3.11. Pelota moviéndose	60
3.12. Direcciones que tomaría la pelota dependiendo del punto de la paleta que golpee (Aproximada)	60
3.13. Efectos de partículas: impacto en muro (izquierda) y explosión (derecha)	61
3.14. Modelo del Jefe.	62
3.15. Inspector mostrando los componentes del Jefe.	63
3.16. Diagrama de estados del jefe.	64
4.1. Relación entre desafío y la ansiedad/aburrimiento.	65
4.2. Relación entre desafío y la ansiedad/aburrimiento.	66
4.3. Relación entre desafío y la ansiedad/aburrimiento.	67

Capítulo 1

Introducción

1.1 Motivación

Independientemente de la rama del desarrollo de los videojuegos en la que se pretenda trabajar, el factor decisivo que usan las empresas para determinar a qué profesional contratar son los juegos que ha desarrollado[BS12].

A través de la lista de juegos en las que una persona ha trabajado, los reclutadores y directores de recursos humanos pueden determinar la valía de un futuro empleado. Especialmente, sirve para valorar la capacidad de la persona para terminar un proyecto de gran envergadura, realizar una gestión de recursos correcta y su creatividad e iniciativa. De forma más específica, en el caso concreto de los programadores, una lista de proyectos completados (que pueden ser tanto juegos como herramientas de desarrollo) permite valorar su forma de desarrollar, mostrando si es capaz de escribir código comprensible y libre de errores.

La visibilidad que te da tener juegos completados no solo afecta a las empresas que buscan empleados. Tener una lista de juegos también sirve como carta de presentación para los jugadores, los cuales la podrán utilizar a la hora de decidir si deben o no adquirir un juego de dicho desarrollador, o invertir en una campaña de *crowdfunding*.

Desarrollar videojuegos también es beneficioso para cualquier aspirante a desarrollador de videojuegos. Durante el desarrollo de cualquier videojuego se plantearán problemas (“¿Cómo se modela el comportamiento de un enemigo?”, “¿Qué editor de niveles es más conveniente?”, “¿Cuánto tiempo me llevará implementar esta característica concreta?”) los cuales muchas veces volverán a aparecer en muchos, sino todos, los futuros proyectos, por lo que con cada juego, el desarrollador ganará experiencia que le facilitará el trabajo en sus siguientes proyectos y le permitirá emprender proyectos cada vez más ambiciosos.

Con estas ventajas en mente se eligió el desarrollo de un videojuego completo como proyecto de Fin de Grado.

1.2 Diseño de juego

1.2.1 Concepto

Este juego será de estilo-Breakout, género que se caracteriza por controlar una paleta con la que se redirige una pelota con el objetivo de destruir un muro de ladrillos o bloques. Sin embargo, este juego transcurrirá en un entorno totalmente tridimensional (a diferencia que otros juegos del género, los cuales tiene una área de juego bidimensional).

En esta adaptación a las tres dimensiones, determinados elementos del juego han sido modificados respecto al estándar del género, inspirándose en otros juegos o deportes como el Squash¹



Figura 1.1: Breakout (Atari, 1976)

1.2.2 Jugabilidad

En este juego, el jugador controla a un personaje encerrado en una sala cubica. El objetivo del juego es el de salir de la sala abriendo una enorme puerta que ocupa la totalidad de la pared norte de la sala. Para lograrlo, el jugador debe golpear una pelota usando una paleta para golpear con ella la puerta. Tras varios golpes, la puerta se abrirá. Sin embargo, la puerta estará cubierta por un muro de ladrillos que el jugador deberá romper primero para poder alcanzar la puerta. Por otro lado, si la pelota golpea tres veces consecutivas el suelo, el jugador perderá la partida.

El control del personaje principal es sencillo, ya que solo cuenta con dos acciones: moverse por el suelo de la sala y activar la paleta. El movimiento del personaje es un movimiento en ocho direcciones (delante, atrás, derecha, izquierda y diagonales), con

¹<http://www.realfederaciondesquash.com/>

una aceleración alta para dar una buena sensación de control. La activación de la paleta permitirá controlar la dirección de la bola, haciendo que esta rebote, impidiendo que caiga en el suelo y, idealmente, redirigiéndola hacia la puerta. La paleta estará activa durante un tiempo limitado, durante el cual el personaje permanecerá inmóvil. Si el personaje es golpeado por la pelota directamente, cuando la paleta está desactivada, el personaje quedará aturdido unos instantes, durante los cuales no podrá realizar ninguna acción.

La pelota se moverá por la sala rebotando de forma natural por sus paredes. Al golpear la puerta, o los bloques que la cubren, la pelota les causará daño. Los bloques se destruirán tras recibir daño suficiente, mientras que la puerta se abrirá, permitiendo al jugador salir de la sala. Si la pelota rebota tres veces en el suelo de la sala sin que el jugador la golpee con la paleta, esta será destruida y la partida se acabará.

La puerta de la sala está cubierta por un muro de ladrillos o bloques. Los bloques se colocan sobre la puerta alineados en una cuadricula de 16 X 16 unidades. Existen varios tipos de bloques, con distintas propiedades, los cuales pueden estar colocados con una rotación de 0° o de 90° y pintados de distintos colores. Los tipos de bloques son los siguientes:

- Bloque Básico: Es el bloque normal, el cual se rompe al recibir un golpe de la pelota. Las dimensiones de este bloque son una unidad de altura y dos de anchura.
- Bloque Triple: Idéntico en dimensiones al bloque básico, pero se romperá tras recibir tres golpes en lugar de uno.
- Bloque Divisible: Se trata de un bloque cuadrado de dos unidades de altura y anchura. Al ser golpeado, este bloque se romperá en cuatro Bloques Pequeños.
- Bloque Pequeño: Es un bloque de pequeño tamaño (una unidad de altura y anchura).
- Bloque Irrompible: Este bloque no puede ser destruido por la pelota, por muchos golpes que reciba. Es también un bloque de gran tamaño, ya que mide el doble que el bloque básico (cuatro unidades de ancho y dos de altura).

El juego contendrá varias salas. Las salas tendrán unas dimensiones constantes, pero distintas configuraciones de bloques cubriendo la puerta, y distintos valores de “puntos de vida” para la puerta. Cuando el jugador abra la puerta de una sala, el personaje saldrá de ella y se moverá a otra sala distinta. Las salas están ordenadas con dificultad creciente, de modo que el jugador se encontrará primero con las salas más sencillas antes de tener que enfrentarse a las más complicadas. Este aumento gradual mantendrá al jugador en un estado mental llamado “fluir” en el cual se tiene una concentración extrema en el juego [Bar]. De forma similar, los distintos tipos de bloques serán introducidos de forma progresiva

1. INTRODUCCIÓN

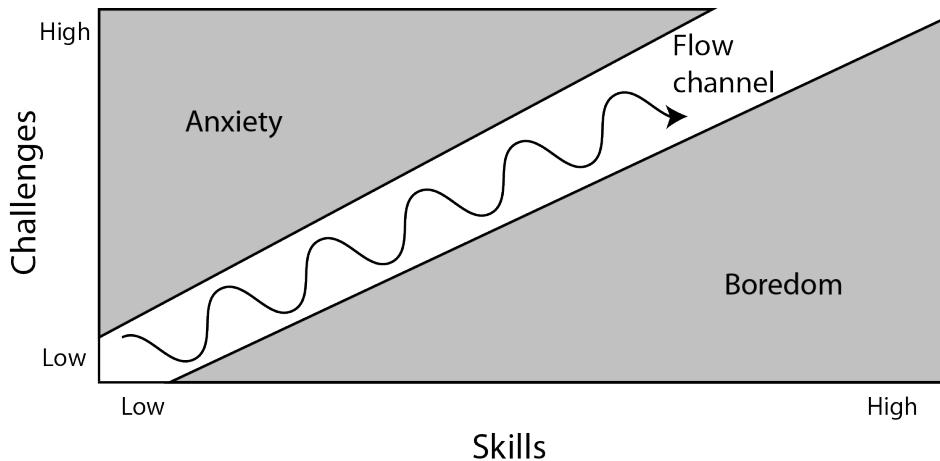


Figura 1.2: Relación entre desafío y la ansiedad/aburrimiento.

1.2.3 Estilo y Ambientación

El equipo de desarrollo de este juego está formado por una única persona, la cual debe realizar la producción completa del juego, que incluye diseño, programación, producción de los assets gráficos y creación de la música y sonidos. Dado que la responsabilidad de la producción artística cae en manos de un desarrollador que debe, además, ocuparse de otras responsabilidades dentro del proyecto en lugar de en un profesional que se dedique a tiempo completo en la producción, los posibles estilos gráficos y sonoros son limitados.

La elección del estilo artístico se limitaba a dos opciones: utilizar assets con licencia de libre uso, obtenidos de páginas como OpenGameArt.com², o producir el arte utilizando un estilo sencillo y minimalista que pueda producirse en poco tiempo. La opción que se eligió fue la producción propia utilizando un estilo minimalista, ya que daría al juego un aspecto más único y cohesivo. Dado que el diseño del juego requiere de gráficos tridimensionales, se optó por un diseño de personajes y escenarios con un bajo número de polígonos, al mismo tiempo que se empleaban texturas con una resolución y paleta de colores muy limitada, creadas usando el estilo artístico “Pixel Art”, el cual imita las limitaciones gráficas de los ordenadores y videoconsolas antiguas. Los modelos resultantes de aplicar este estilo recuerdan a modelos de papel o maquetas construidas con bloques.

Las opciones para la producción musical son mucho más reducidas. Debido a la dificultad de crear una banda sonora desde cero, la única opción es utilizar música con licencia de libre uso. Para que la música esté en consonancia con el estilo artístico, se utilizará música de estilo “Chiptune” el cual, al igual que los gráficos, imita las limitaciones técnicas de videoconsolas y ordenadores antiguos. Esto mismo se aplicará a los efectos de sonido, también con licencia de libre uso e inspirados en los sonidos de

²<https://opengameart.org/>



Figura 1.3: El estilo del juego está influenciado por títulos como Megaman Legends(Capcom, 1997)

videojuegos antiguos.

El estilo artístico resultante es muy “Digital”, con polígonos visibles, píxeles de gran tamaño, colores limitados y música y sonido sintéticos. La consolidación de este estilo sirvió para definir la ambientación narrativa del juego: se decidió que el juego estará ambientado en el interior de un ordenador, con los personajes y entornos representando programas. Aunque el juego no hace uso de esta narrativa, al carecer de diálogos y cinematográficas, esta temática se empleó en el diseño de los gráficos.

1.2.4 Plataforma

Este juego ha sido diseñado para ser jugado en un PC. El control del personaje principal se realiza mediante las flechas de dirección (movimiento) y la tecla Espacio (paleta y navegar por menús).

La exportación de este juego a dispositivos móviles como Android o IOS no sería posible debido a que el sistema de control no es compatible con pantallas táctiles, por lo que sería necesario realizar una adaptación, ya sea implementado un sistema de control del personaje basado en la pantalla táctil o mediante un controlador virtual en pantalla. Ambas aproximaciones aumentarían el coste de desarrollo del juego. Además, el juego no está optimizado para este tipo de dispositivos, de menor potencia que los PC, por lo que podría no funcionar correctamente.

La distribución del juego se realizará online, a través de las páginas de distribución de juegos Game Jolt³ y itch.io⁴. Estos sitios ofrecen a sus usuarios la posibilidad de subir, sin ningún tipo de coste, el juego a la página para que otros usuarios puedan

³<https://gamejolt.com/>

⁴<https://itch.io/>

1. INTRODUCCIÓN

descargarlos (o, en el caso de Game Jolt, jugarlos directamente en el navegador si el juego es compatible). Ambas páginas ofrecen la posibilidad tanto de vender el juego como de distribuirlo de forma gratuita. Dada la naturaleza del proyecto, se ha optado por la distribución gratuita del mismo.

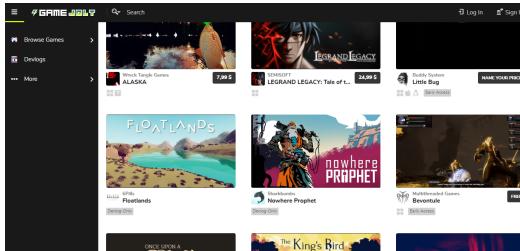


Figura 1.4: Gamejolt

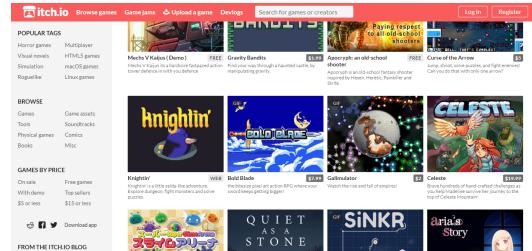


Figura 1.5: Itch.io

Figura 1.6: Juegos desarrollados con Unity3D

Capítulo 2

Estado del arte

2.1 El mercado de los videojuegos

2.1.1 Industria mundial del Videojuego

El mercado del videojuego es actualmente la industria de ocio y entretenimiento más grande, superando en tamaño tanto a la industria cinematográfica como a la discográfica. Se trata de una industria creciente, con una tasa del crecimiento del 6,6 % y que cuenta ya con más de 2.200 millones de jugadores en todo el mundo. Se espera que, en el año 2020, la cotización anual alcance los 143.500 millones de dólares [DEV17].

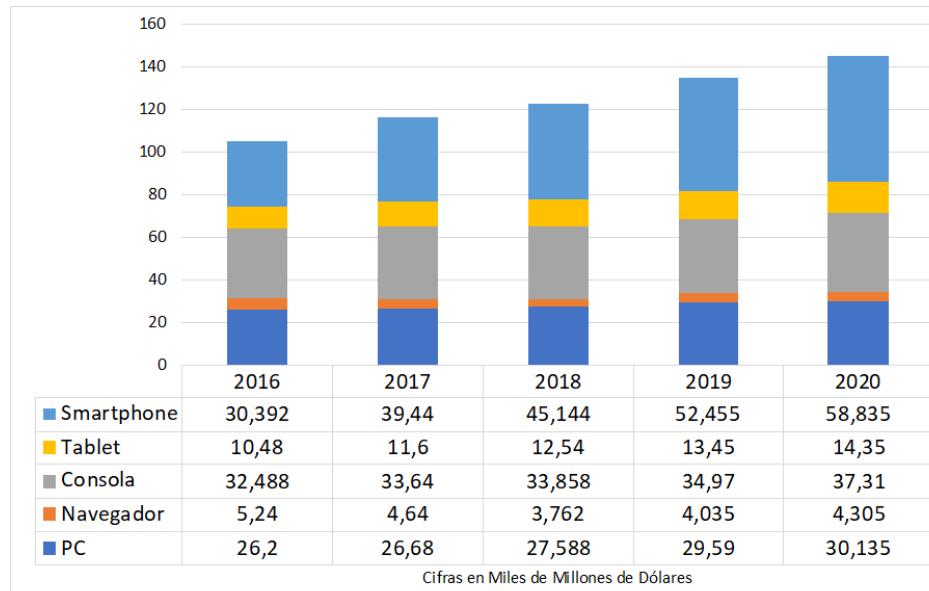


Figura 2.1: Crecimiento Mundial de la industria del videojuego (por plataformas).

Actualmente, la plataforma de distribución que ocupa un segmento mayor del mercado son los dispositivos móviles. Debido al constante incremento de potencia de los Smartphones, así como a su ubicuidad en la sociedad actual, el mercado de videojuegos para estas plataformas ha experimentado un incremento constante en los últimos años, superando al mercado para PC y al mercado para Videoconsolas de sobremesa. A fecha de 2017, el mercado Smartphones ha alcanzado los 39.440 millones de dólares, lo que representa el 34 % del mercado. Por otro lado, el mercado de las consolas portátiles y el de los juegos Web casuales son los que presentan un declive más pronun-

2. ESTADO DEL ARTE

ciado, debido seguramente a que ocupan un nicho de mercado similar al de los juegos móviles[DEV17].

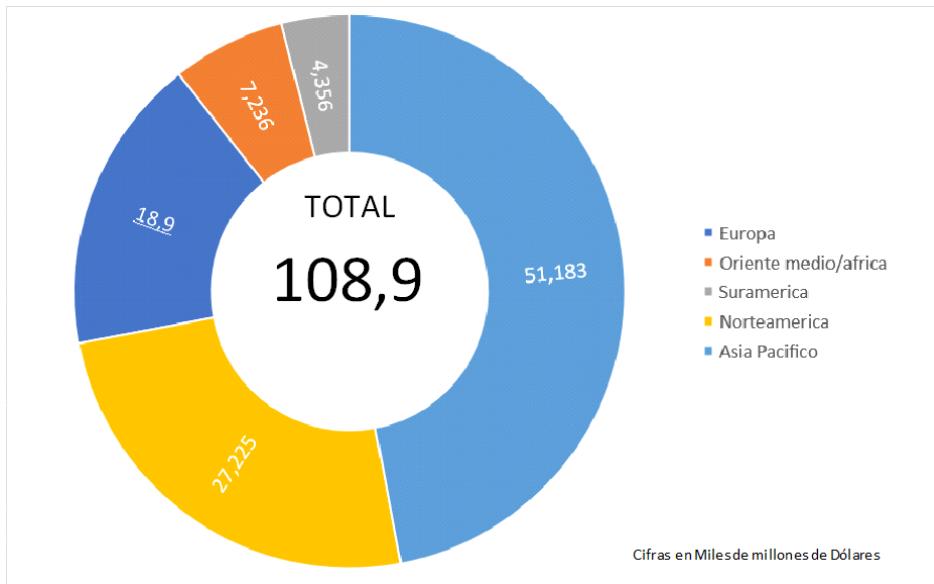


Figura 2.2: Distribución por regiones del mercado del videojuego.

Si dividimos el mercado en las distintas regiones geográficas, podemos observar que Asia Pacifico lidera el mercado con un beneficio de 51.200 millones de dólares en el año 2017, un 47 % del total de los ingresos globales, de los cuales más de la mitad son generados por China, con 227.500 millones de dólares. En segunda posición se encuentra el mercado norteamericano, con 23.500 millones de dólares en 2016, representado prácticamente en su totalidad por Estados Unidos. En el ranking de los 10 mercados con mayores ingresos podemos encontrar cinco países europeos: Alemania, Reino Unido, Francia, España e Italia. Sin embargo, la diferencia en tamaño con respecto a los tres primeros mercados de la lista (China, Estados Unidos y Japón) es abismal[DEV17].

2.1.2 La industria de los videojuegos en España

La industria del videojuego española es la cuarta mayor de Europa (por detrás de Alemania, Reino Unido y Francia) y la novena mayor a nivel mundial. A fecha de 2017, el mercado español del videojuego facturó un total de 1.900 millones de dolares, con un crecimiento del 20 % con respecto al año anterior. Más de la mitad de estos ingresos provienen de la venta de videojuegos españoles al extranjero, gracias a la casi total falta de fronteras para la distribución internacional de productos. Este enfoque en el mercado internacional se ve reflejado en factores como la mayor frecuencia del inglés en las producciones españolas que el propio español (99 % contra 95 %)[DEV17].

En total, el sector cuenta con 480 empresas en activo, a las que debemos añadir las 130 iniciativas y proyectos empresariales, que se encuentran a la espera de consolidarse como empresas en el corto o medio plazo. La mayor parte de estas empresas tienen

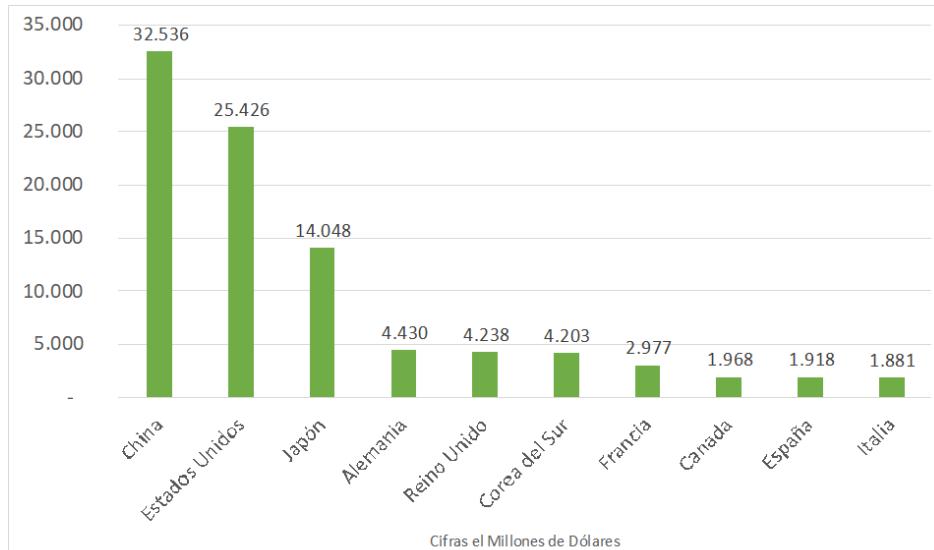


Figura 2.3: Diez principales mercados del videojuego.

una plantilla de menos de 5 empleados, formando el 47% de la industria. Esto se debe en parte a la adecuación de las pequeñas empresas a la creación de juegos de pequeña escala para dispositivos móviles (el principal mercado), pero también se debe a una escasez de puestos de trabajo en las empresas de tamaño mediano y grande y a la saturación del mercado que dificulta el crecimiento de las empresas[DEV17].

La actividad empresarial del país se encuentra centrada en dos comunidades autónomas: Cataluña y la comunidad de Madrid. De estos dos centros principales destaca Cataluña, donde se concentra el 52% de la facturación del país. Detrás de las dos comunidades principales se encuentran la Comunidad Valenciana, el País Vasco y Andalucía, las cuales suman entre las tres un 28% de las empresas. El resto de comunidades se quedan muy por detrás de estas cinco primeras[DEV17].

2.1.3 Retos y tendencias actuales

El videojuego ha sido y es una industria muy cambiante, que siempre ha intentado integrar las tecnologías más punteras, desde innovadores algoritmos de renderizado gráfico hasta exóticos dispositivos de interacción persona-ordenador.

A continuación, listaremos algunas de las tendencias que van a influir fuertemente en el mercado en los años venideros:

eSports

Los eSports, también llamados “deportes electrónicos”, es el nombre por el cual se conocen las competiciones de videojuegos multijugador. En los eSports, los jugadores profesionales compiten entre ellos en diferentes categorías: disparos en primera persona, lucha, estrategia en tiempo real, MOBAs (Multiplayer Online Battle Arena)... La

2. ESTADO DEL ARTE

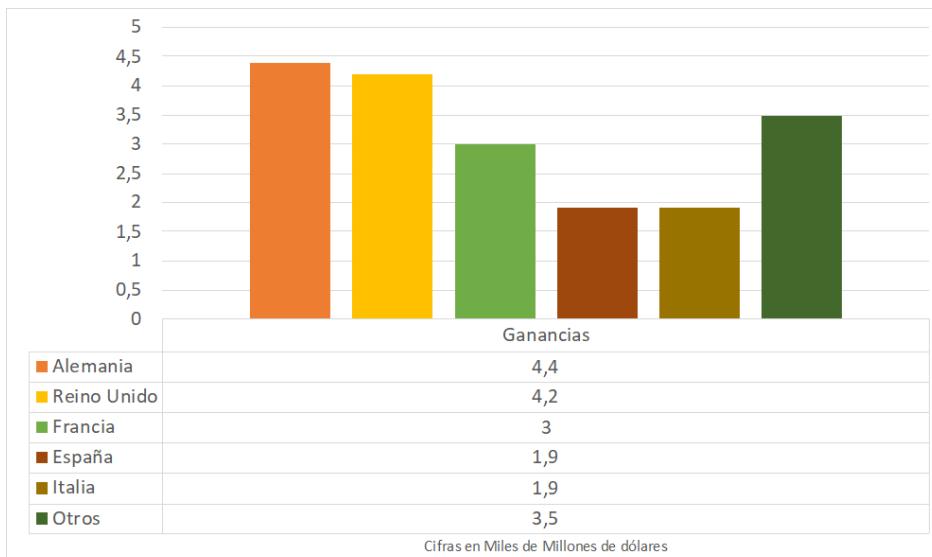


Figura 2.4: Mercado del Videojuego en Europa.

popularidad de este fenómeno ha llegado al punto en el que los principales torneos se celebran en grandes estadios, están retransmitidos en streaming por Internet e incluso están dotados con premios de grandes cantidades de dinero y que en ocasiones superan el millón de euros. Se trata de unos eventos de gran popularidad que cuentan enormes con perspectivas de crecimiento (a una tasa anual del 34 %). Esto ha provocado que se haya posicionado como un fenómeno de ocio estratégico[DEV17].

El mercado de los eSports, generó en 2017 600 millones de dólares de ingresos, creciendo del 67 % con respecto al año anterior. Si la situación se mantiene, se espera que la cifra supere los mil millones de dólares en el año 2019, manteniendo un crecimiento anual superior al 34 %. En 2016, hubo 385,5 millones de personas que vieron partidos de competiciones de diversos deportes electrónicos, de los cuales 115 millones pueden ser clasificados como “entusiastas”, es decir, espectadores regulares y participantes no muy distintos de los que encontraríamos en los deportes convencionales[DEV17].

Existen en España diferentes empresas que en la actualidad apuestan por el desarrollo de productos que aspiran a convertirse en eSports, es el caso de Digital Legends¹, Mercury Steam², PixelCream Studio³, Mechanical Boss⁴, entre otros. Sin embargo, en la mayoría de las ocasiones un videojuego online se convierte en eSports de manera orgánica, basada en el reconocimiento que la comunidad online de jugadores activos de ese videojuego le otorga. Existen eso si casos en las que una gran marca apuesta porque su producto se convierta en un eSports, realizando grandes inversiones en infraestructura, personal dedicado a la comunidad, servidores escalables para una gran

¹<http://www.digital-legends.com/>

²<https://www.mercurysteam.com/>

³<http://pixelcreamstudio.com/>

⁴<http://www.mechanicalboss.com/>

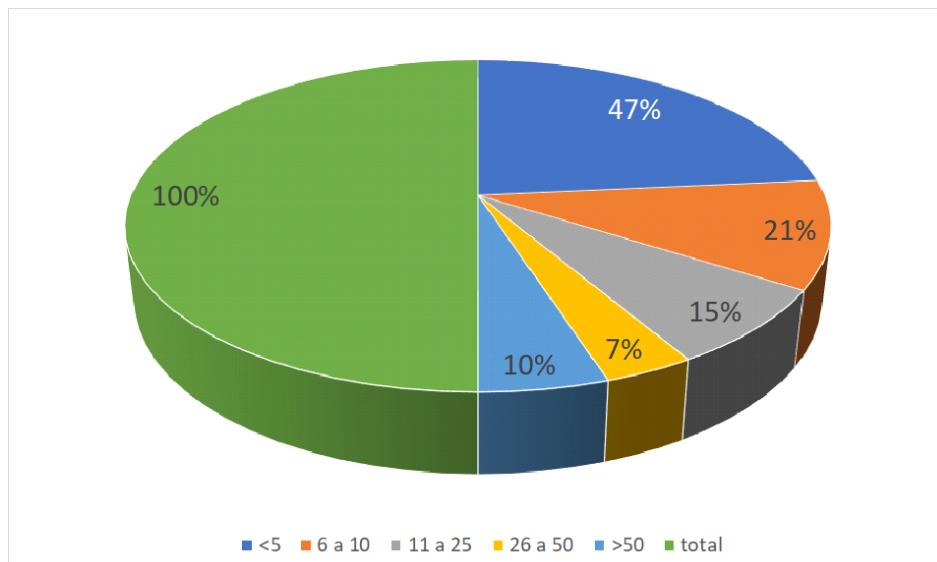


Figura 2.5: Distribución de las empresas en España por número de empleados.

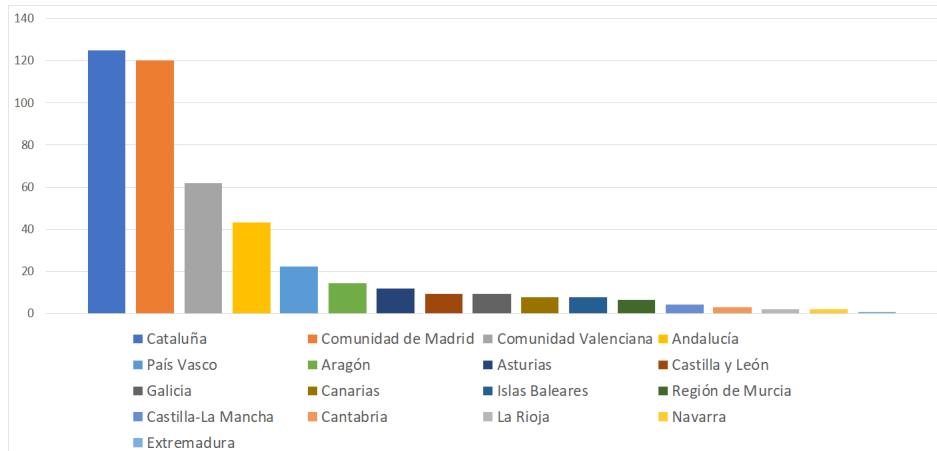


Figura 2.6: Distribución de las empresas en España por número de empleados.

masa de jugadores, premios para los torneos, entre otras muchas.

Sin embargo, esta inversión no siempre asegura que su producto se convierta en un éxito. Para que un videojuego pueda convertirse en un eSport necesita contar con características básicas: tener un fuerte factor de competición, partidas cortas de no más de 1 hora, sin progresión in-game (la progresión debe basarse en las habilidades del jugador) atractivo sistema de espectador y tener un enfoque al 100 % internacional.

Pese a su gran dificultad, conseguir posicionar un producto como eSports, aporta una serie de beneficios y posibilidades:

- Crear una base de fans, una comunidad, algo que aporta un núcleo de consumidores fieles al producto y que le da una nueva dimensión social, muy atractiva para muchos de los consumidores de videojuegos.
- Prolongar la vida del producto; al ser competitivo, el jugador fija sus metas ante

2. ESTADO DEL ARTE



Figura 2.7: Fotografía del torneo Intel® Extreme Masters en Katowice, Polonia

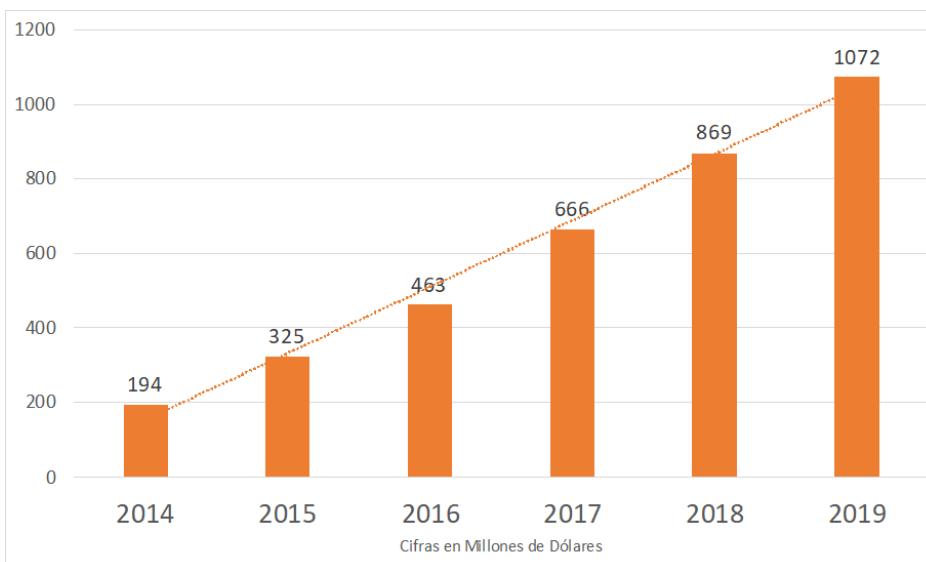


Figura 2.8: Crecimiento del mercado del eSport

los otros jugadores, esto incentiva al usuario y le proporciona una motivación para seguir consumiendo.

- Proporcionar mayor visibilidad, ya que a pesar de que los productos asentados son extremadamente sólidos, su numero es muy reducido, por lo cual hay una demanda latente de usuarios que buscan nuevos eSports.
- Aumentar la fidelidad de los usuarios al tratarse de un mercado donde los usuarios tienen un índice de fidelidad mucho más alto que en otros.
- Los jugadores, al estar involucrado con un producto competitivo, ven streaming, leen noticias, siguen torneos, participan en foros, lo que disminuye el riesgo de abandono del producto.



Figura 2.9: *League of Legends* (Riot Games, 2009), uno de los eSports más populares

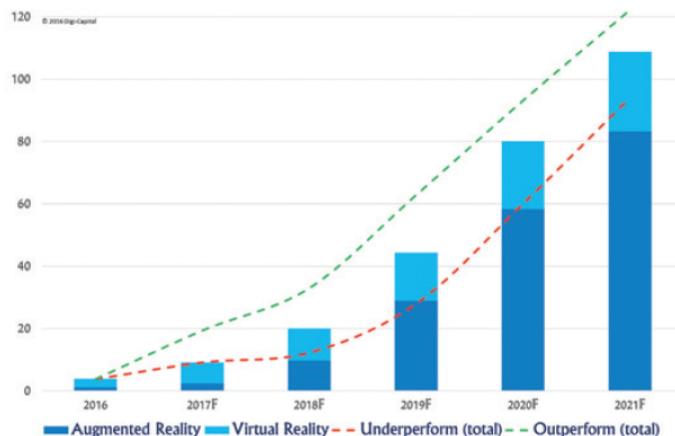
Para una empresa pequeña, la producción de un eSports es, en principio, inabarcable. Esto se debe principalmente la elevada inversión mencionada anteriormente. Sin embargo, la asociación con unos los partners correctos, ya establecidos en el sector, que pudieran invertir en el producto y le den visibilidad, puede dar una gran oportunidad a los pequeños desarrolladores que cuentan con una mayor flexibilidad con respecto a las grandes compañías, lo que les permite adaptarse más rápidamente a un mercado tan cambiante y mantener una relación directa con el feedback del usuario.

Realidad Virtual y Realidad Aumentada

La realidad virtual (normalmente abreviada como VR por las siglas inglesas de Virtual Reality) es la tecnología generada por sistemas informáticos que proporcionan un entorno audiovisual en 3D el que el usuario puede experimentar una inmersión total. Para ello, se hacen usos de cascos especiales equipados con pantallas y sensores de movimiento, los cuales normalmente se complementan con mandos equipados también con sensores para permitir una interacción más natural. Por otro lado, la Realidad Aumentada o AR es una tecnología que superpone una capa de gráficos generados por ordenador sobre el entorno que rodea al jugador, con la que este puede interactuar en tiempo real. A diferencia de la VR, no se requiere obligatoriamente de un hardware especial para poder implementar AR; basta únicamente de un dispositivo equipado con una pantalla y una cámara de vídeo, como podría ser un Smartphone.

Actualmente, existen varias propuestas de diversas compañías en lo que a equipo de VR se refiere. Vamos a mencionar algunas de las más importantes:

2. ESTADO DEL ARTE



Fuente y gráfico original: Digi-Capital

Figura 2.10: Previsiones de crecimiento del mercado de Realidad Virtual y Aumentada (miles de millones de dólares))

- **HTC Vive⁵**: es la propuesta de HTC y Valve, orientada a jugadores “hardcore” de PC. Disponible desde abril de 2016, el dispositivo requiere de un PC de gama alta (Valve recomienda un PC con una gráfica GeForce GTX 970). El kit de hardware incluye el casco equipado con dos pantallas de 1080x1200 puntos y 90Hz de frecuencia de actualización, dos sensores espaciales y dos mandos para registrar los movimientos de ambas manos, lo que crea le permite crear un entorno 100 % virtual en el que sumergir al jugador.
- **OCULUS Rift⁶**: Es la propuesta más veterana de la lista. Empezó como un exitoso proyecto de Kickstarter en 2012 que más tarde fue adquirida por la empresa Facebook dos años más tarde. Al igual que HTC Vive, Oculus está formado por un casco equipado con pantallas de alta resolución, mandos con sensores de movimiento y dos sensores de posición. El equipo necesita estar conectado a un PC de alta gama para poder funcionar correctamente.
- **Samsung Gear VR⁷**: La propuesta de Samsung es mucho más sencilla y económica, orientado más a la reproducción de vídeo en 360º (concepto similar a la realidad virtual pero con interactividad limitada). El casco incluye una única pantalla y sus mandos carece de detección de movimiento. Estas limitaciones conllevan, por otro lado, un precio mucho más accesible que el de las otras alternativas (99€ contra los más de 500€ de las propuestas más completas)
- **SONY PlayStation VR⁸**: la propuesta de Sony fue lanzada en el año 2016. Al igual que otras alternativas, el sistema se basa en un casco equipado con

⁵<https://www.vive.com/>

⁶<https://www.oculus.com/rift/>

⁷<http://www.samsung.com/es/wearables/gear-vr-sm-r325nzbaphe/>

⁸<https://www.playstation.com/explore/playstation-vr/>

dos pantallas y sensores de movimiento, pero su principal punto de venta es su compatibilidad con la consola PlayStation 4 de la misma marca. Esto permite aprovechar la potencia y los mandos de control de ésta de la consola.



Figura 2.11: De izquierda a derecha HTC Vive, Oculus Rift, Samsung VR y PlayStation VR

Web 4.0

El término Industria 4.0 fue acuñado por el Ministerio de Educación y Desarrollo alemán en su plan estratégico de 10 puntos del año 2016 para mejorar la educación, investigación e industria del país para adaptarlas a las tecnologías de Internet [Lyd]. La estrategia trata cinco áreas principales:

- Fuerte cooperación entre la investigación científica y las empresas.
- Aumentar la innovación en el sector privado.
- Diseminar las tecnologías punteras.
- Internacionalizar la investigación y desarrollo.
- Fondos para individuos con talento.

De entre las distintas tecnologías que podrían categorizarse como parte de la industria 4.0, vamos a describir aquellas que tienen mayores aplicaciones en el desarrollo de videojuegos:

La computación en la nube, o Cloud Computing, es la tecnología que permite el acceso a servicios informáticos de forma rápida y sencilla a través de Internet. Aunque aún no se ha podido implementar correctamente el Cloud Gaming (donde el juego es íntegramente ejecutado en la nube, reduciendo la exigencia de potencia del sistema del jugador), si se utilizan sistemas en la nube en distintas áreas de los videojuegos. Especialmente notable es su uso para el control y almacenamiento de información en juegos multijugador en línea.

El Internet de las cosas es como se conoce a la tecnología que permite dotar de conexión a Internet a todo tipo de pequeños dispositivos como relojes, sistemas de domótica, drones, sensores de todo tipo, robots, etc. Esto permite implementar videojuegos en todo tipo de sistemas, desde consolas portátiles cada vez más pequeñas y económicas, pasando por juguetes interactivos y llegando a la posibilidad de gamificar con facilidad procesos industriales.

PRINCIPALES ÁREAS DE LA INDUSTRIA 4.0



Figura 2.12: Áreas de la Industria 4.0

Big Data es el proceso de clasificar grandes volúmenes de datos para poder obtener relaciones interesantes y no evidentes entre ellos. El principal uso de las técnicas de BigData en la industria del Videojuego es el análisis de la información de los jugadores. Analizando datos de los jugadores tales como el género, la edad, la localización geográfica, los intereses, los gastos realizados, etc. es posible obtener estrategias de negocios eficientes.

Los sistemas Ciberfísicos son un nuevo tipo de sistemas con unos componentes hardware y software estrechamente interconectados, cada uno operando en su propio ámbito, operando e interaccionando de forma distinta dependiendo del contexto [Sid14]. Entre sus aplicaciones se encuentran las redes eléctricas inteligentes, los sistemas de conducción automática de aviones y automóviles o la monitorización médica. Las interfaces de estos sistemas requieren de unas interfaces con un fuerte “lado humano” que permita un uso sencillo e intuitivo. Aquí se podrían utilizar los principios de diseño de juego que

permitirían desarrollar un mejor puente entre el lado máquina y la parte de usuario.

La Impresión 3D, también conocida como la producción aditiva es una tecnología que permite producir objetos de forma más sencilla que con las técnicas anteriores. En combinación con las técnicas de escaneado 3D, los estudios de videojuego pueden generar de forma rápida y eficiente modelos 3D de todo tipo (personajes, mapas, objetos...).



Figura 2.13: Impresora 3d “replicator” de la compañía Makerbot.

Las nuevas tecnologías de la industria 4.0 serán de gran ayuda para el desarrollo de videojuego. Pero es posible que la industria 4.0 también ofrezca valor a la industria 4.0 en su conjunto. Dado que la creación de videojuegos es una actividad industrial que está vinculada a diferentes áreas de conocimiento que trabajan juntas para conseguir ofrecer un producto, las técnicas y paradigmas utilizados tienen mucho en común con la nueva forma de trabajar de la industria 4.0, por lo que es posible que puedan extrapolarse a otras industrias, permitiendo una mejor adaptación a los cambios.

2.2 El proceso de desarrollo de videojuegos

2.2.1 Introducción

El desarrollo de un videojuego, como el de cualquier otro producto software, debe de ser planificado correctamente y ejecutado siguiendo una metodología adecuada. Sin embargo, El diseño y desarrollo de un videojuego requiere de la participación de campos ajenos a la informática como el diseño de juegos, el diseño gráfico o la composición musical. Una parte importante de la producción consistirá en organizar a un equipo multidisciplinario para poder terminar el proyecto dentro del tiempo y presupuesto acordados [Dav15].

2. ESTADO DEL ARTE

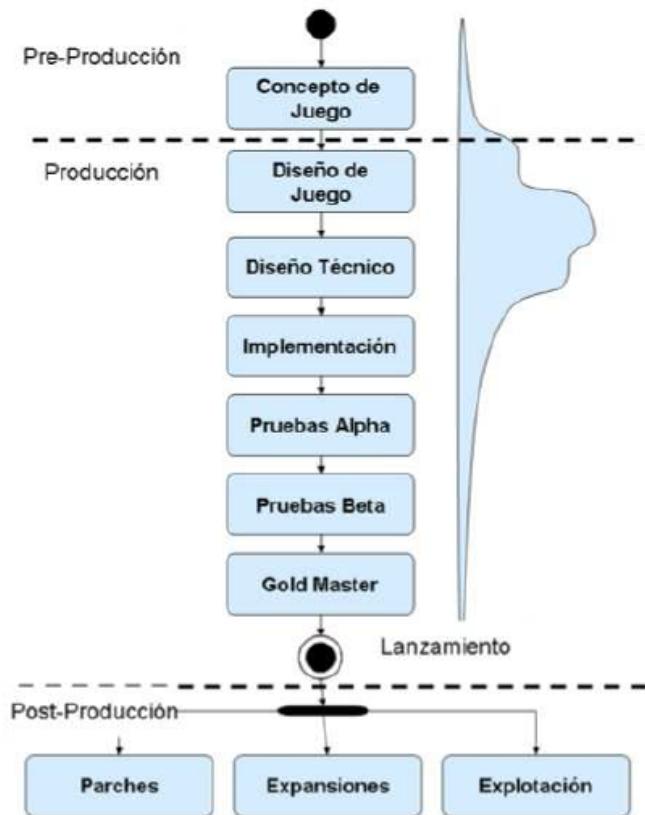


Figura 2.14: Etapas del desarrollo de un videojuego.

2.2.2 Etapas del desarrollo

Desarrollo del Concepto

El desarrollo de todo videojuego comienza con una idea. Durante la fase de desarrollo del concepto se tomará dicha idea para obtener un diseño preliminar listo para preproducción. El objetivo principal de esta etapa es decidir sobre qué tratará el juego y ponerlo por escrito para que cualquier miembro del equipo lo pueda entender con claridad, decidiendo las principales mecánicas, creando el arte conceptual y escribiendo el argumento [Bat04].

Al final de la etapa se habrán elaborado tres documentos: El *High Concept*, el *Pitch Document* y el *Concept Document*. El *High Concept* consiste en una o dos frases que describen a grandes rasgos cómo será el juego, en especial que lo hace distinto de la competencia. El *Pitch Document*, o Propuesta de Juego, es un pequeño documento de en torno a dos páginas, orientado a ser leído en las reuniones donde el juego sea propuesto a inversores. El documento resume las características del juego y explica por qué será exitoso y rentable.

Finalmente, el *Concept Document* es un extenso documento que explica en detalle las características del juego. Se trata de una versión extendida de *Pitch Document* que

trata temas como:

- El High Concept
- El género del juego
- Características principales y jugabilidad.
- Ambientación e historia.
- Estimación del presupuesto y de la planificación.
- Equipo de desarrollo.
- Análisis de Riesgos.

Pre-Producción

La pre-producción es la fase de preparación: en la que el equipo diseña y planifica los elementos que serán desarrollados en la etapa de producción. Al final de esta etapa, se debe haber completado el diseño de juego, creado la biblia de arte, elaborado el documento de diseño técnico, establecido el plan de producción, y creado un prototipo del juego final[Bat04].

El diseño del juego quedará establecido en un Documento de Diseño de Juego (o GDD, por sus siglas en inglés). Se trata de un documento “vivo”, que está en constante modificación para adaptarse a los ajustes concretos de diseño que se realizarán durante el desarrollo.

La biblia de arte es una colección de arte conceptual que servirá para definir el estilo artístico del juego desde un primer momento. La biblia incluirá también una librería de imágenes de referencia que puedan ser de ayuda a los artistas que desarrollen los elementos gráficos finales.

El Documento de Diseño Técnico contiene una descripción en detalle la parte técnica del proyecto definiendo las tareas que los desarrolladores deberán afrontar, estimando el coste que dichas tareas tendrán tanto en tiempo como en número de personas y especificando las herramientas y técnicas que utilizará el equipo.

El Plan de producción recopila la información acerca de cómo se va a desarrollar el proyecto. Este incluye las tareas a realizar junto a los tiempos, costes y dependencias de estas, divididos en varios documentos menores para poder ser organizado mejor:

- **Plan de mano de obra:** Listado del personal, sus horarios y su salario.
- **Plan de recursos** Estimación del coste los recursos externos al proyecto (música, arte, herramientas...)
- **Documento de seguimiento:** Documento donde se realiza un control de los tiempos y plazos del proyecto.

2. ESTADO DEL ARTE

- **Presupuesto:** Contiene el coste mensual del proyecto y el cálculo del presupuesto general
- **Ganancias y Pérdidas:** Estimación de las ganancias y pérdidas del proyecto. Debe ir actualizándose según se avanza en el desarrollo.
- **Definición de Hitos:** Lista de las distintas “metas” del proyecto, que son puntos del desarrollo donde se habrá terminado una cantidad de trabajo importante.

Una vez diseñado y planificado el proyecto, el equipo empezara a trabajar en la creación de un Prototipo. Un prototipo es una pieza de Software funcional que contiene una pequeña fracción del software final. El desarrollo prototipo servirá para varias funciones: poner a prueba el diseño de juego, concretando de forma más precisa la jugabilidad; realizar un simulacro de desarrollo para determinar las dinámicas del equipo y producir una muestra del juego final a inversores y publicadores.

Producción

La producción es la etapa principal del desarrollo del juego. Durante esta etapa se elaborarán e implementarán los elementos descritos y diseñados durante la pre-producción: los programadores implementarán los sistemas del juego, los artistas elaboraran los gráficos definitivos, los diseñadores crearán misiones y niveles, etcétera. Dependiendo del juego en cuestión, una producción normal suele durar entre seis meses y dos años, aunque el desarrollo de juegos pequeños para, por ejemplo, dispositivos móviles puede realizarse en menos tiempo aún.

La etapa de producción es de naturaleza iterativa: El juego va construyéndose en varias etapas en las que se implementan pequeñas porciones de este. Entre etapa y etapa, el juego pasa por un proceso de pruebas en la que se verifica su usabilidad y robustez frente a fallos. Los resultados de las etapas se utilizan como base para recalcular los tiempos y presupuestos de las etapas posteriores. Dividir el desarrollo de esta forma hace que sea más sencillo afrontar problemas y contratiempos que el equipo podría encontrar en las etapas tardías.

Final de Producción y Lanzamiento

Durante las últimas etapas de la producción, el paradigma de desarrollo cambia, pasando el objetivo de implementar contenido nuevo a pulir y ajustar el contenido preexistente. Esta parte de la producción puede dividirse en dos etapas: Alpha y Beta.

La fase **Alpha** o Code-Complete es el punto del desarrollo donde el juego se encuentra en un estado jugable, a falta solo de ciertos vacíos como gráficos provisionales o mini juegos o sub-sistemas incompletos. El objetivo de esta etapa es el de encontrar y corregir todos los fallos posibles y también probar y ajustar la jugabilidad[Bet03].

En la fase **Beta** o Content-Complete la mayor parte del contenido del juego deberá estar terminado y debe haber pocos o ningún fallo importante en el juego. En esta etapa el juego es puesto en manos de equipos de testing externos a la empresa para realizar análisis exhaustivos en busca de fallos que se le pueden haber escapado al equipo de testing interno. Es también en esta etapa donde la campaña de publicidad del juego deberá ser más fuerte[Bet03].

Una vez superadas las dos etapas de pruebas, la Versión Final del juego es enviada a la distribuidora para que comience la producción de las copias físicas, o para que el juego aparezca en las plataformas de distribución.

Post-Producción

Una vez lanzado el juego, el equipo entra en la etapa de Post-Producción. En esta etapa el equipo trabajará en corregir fallos y problemas que los jugadores encontraron tras el lanzamiento y en la elaboración de contenido adicional descargable.

La duración y trabajo de la post-producción depende mucho del juego en cuestión. Hasta hace relativamente poco, los juegos lanzados en videoconsolas carecían completamente de esta etapa debido a la dificultad para modificar los juegos que ya se encontraran en el mercado. por otro lado, hoy en día la mayor parte de los videojuegos reciben parches y actualizaciones sin importar su plataforma de distribución[Bet03].

Un caso especial sería el de los juegos con un fuerte componente Online, como los MMORPG, los MOBA o los shooter en línea. Los desarrolladores de este tipo de juegos, para mantener contenta a su base de jugadores y evitar el estancamiento, lanzan de forma periódica actualizaciones que ofrecen nuevo contenido, mejoras y ajustes. Algunos de estos juegos pueden recibir este tipo de actualizaciones durante años, como *World of Warcraft* (Blizzard Entertainment, 2004) o *Team Fortress 2* (Valve, 2007), ambos en activo desde hace más de 10 años.

2.2.3 Estructura típica de un equipo de desarrollo

El desarrollo de un videojuego puede llevarse a cabo por equipos de desarrollo muy distinto dependiendo de la extensión del proyecto, desde una sola persona creando un pequeño juego independiente, el cual realiza todo el trabajo por sí mismo; hasta los equipos de cientos de personas que desarrollan los juegos “triple A”, organizados en múltiples departamentos, cada uno con su estructura jerárquica.

A pesar de esto, existen una serie de roles que están presentes en todos los desarrollos. En los proyectos pequeños será normal encontrar que una misma persona ejerce varios roles, mientras que las grandes compañías pueden tener departamentos enteros encargándose de un único rol.

2. ESTADO DEL ARTE

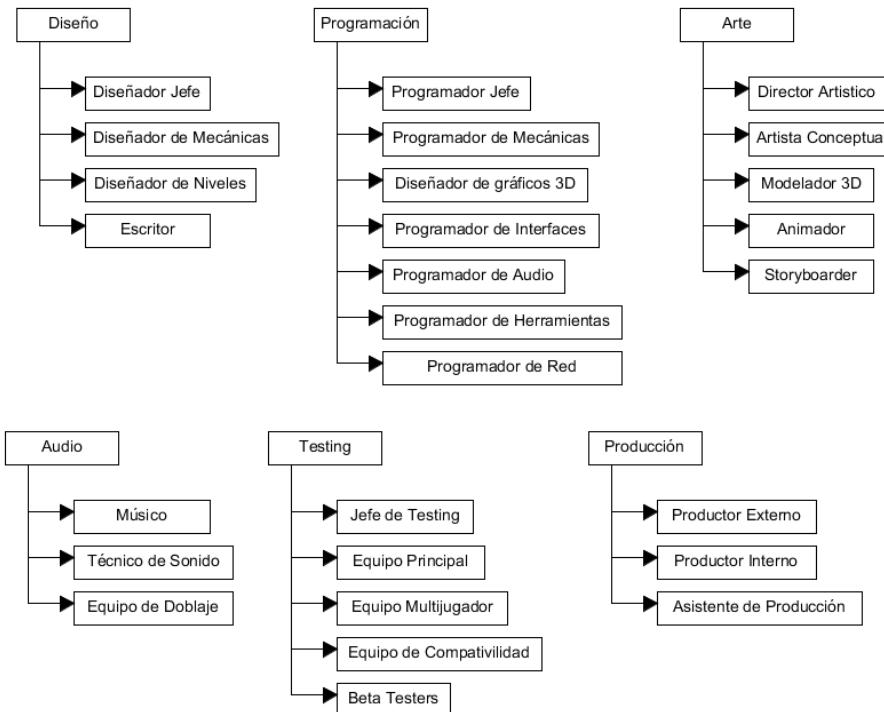


Figura 2.15: Distribución de roles en un equipo de desarrollo.

Diseño

La primera y más importante parte del desarrollo de un juego es el Diseño de este. El trabajo del diseñador es de describir el juego con un alto nivel de detalle, definiendo de con precisión todas las mecánicas, personajes, mapas, misiones del juego. Deberá también, hasta cierto punto, coordinar y dirigir el trabajo del resto de miembros del equipo para que puedan implementar correctamente el juego.

En equipos grandes, el rol de diseñador puede dividirse en las siguientes categorías[Bet03]:

1. **Diseñador Jefe:** Es el encargado de dirigir al resto de diseñadores, decidiendo que contenido entra o no en el juego. Suele ser la persona que tuvo la “idea” original del juego.
2. **Diseñador de mecánicas:** El diseñador de mecánicas es el encargado de diseñar los distintos sistemas de juego, sirviendo como puente entre el diseñador jefe y los programadores. Debido a esto, el diseñador de mecánicas suele tener un trasfondo de programador.
3. **Diseñador de niveles:** También llamados diseñadores de misiones, son los encargados de crear las distintas etapas que componen el juego, ya sean niveles, misiones, desafíos o puzzles.
4. **Escritor:** La tarea del escritor es la de crear la historia del juego, así como la

de escribir los distintos textos de este, como diálogos o descripciones. Se trata de una tarea muy distinta de la de un escritor de novelas o de guiones de película, ya que debe conciliar la narrativa con las exigencias de otros componentes del juego como el diseño, el arte o las limitaciones técnicas.

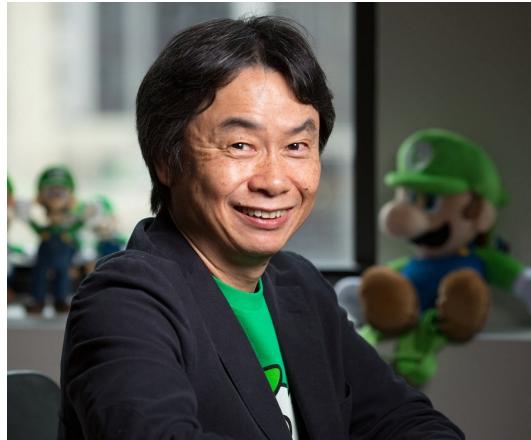


Figura 2.16: Shigeru Miyamoto (creador de *Super Mario Bros.* (Nintendo, 1985), *The Legend of Zelda* (Nintendo, 1986) y *Donkey Kong* (Nintendo, 1981)) es posiblemente el diseñador de videjuegos más famoso.

Programación

El rol del programador es el de implementar el juego en forma de código ejecutable. Esto supone el diseño e implementación de todo tipo de componentes imprescindibles: motor de renderizado, librerías para trabajar con sistemas de audio o conectarse por Internet, herramientas para integrar fácilmente el contenido artístico entre otras.

Cuando el equipo de desarrollo es grande, el rol de programador tiende a dividirse para cubrir tareas más específicas[Bet03]:

1. **Programador Jefe:** El programador jefe es frecuentemente el programador con más experiencia del equipo y él es el encargado de resolver las tareas más complicadas e importantes del proyecto. Cuando el equipo es muy grande, suelen realizar también tareas de coordinación.
2. **Programador de Mecánicas:** Es el encargado de convertir el diseño de juego en código ejecutable. Entre sus tareas se encuentra definir las físicas del mundo del juego, definir las funciones de los distintos objetos y modelar el comportamiento de los personajes.
3. **Programador de gráficos 3D:** Es el responsable de implementar los sistemas para la creación y renderizado de gráficos 3D. El programador de gráficos 3D necesita contar con conocimientos avanzados en cálculo, matemática vectorial y matricial, trigonometría y álgebra.

2. ESTADO DEL ARTE

4. **Programador de Interfaces:** Es el encargado de implementar los sistemas de interacción entre el jugador y el juego, normalmente interfaces de control, menús y HUDs (Head-Up Displays).
5. **Programador de Audio:** Es la persona responsable de la implementación de los distintos sistemas que se utilizarán para reproducir música y sonido en el juego
6. **Programador de Herramientas:** El programador de herramientas tiene la responsabilidad de crear las distintas herramientas que el resto del equipo pueda necesitar para realizar, o acelerar, su trabajo. Un tipo especializado de programador de herramientas es el programador del editor de niveles, debido a la importancia de esta herramienta para el desarrollo del juego y por la posibilidad de que dicho editor sea lanzado al público como parte del juego.
7. **Programador de Red:** Es el encargado de escribir el código que permite a los juegos ser ejecutados entre varios equipos, ya sea código máquina de bajo nivel o la integración de un librería de alto nivel.



Figura 2.17: Jonh Romero es el co-fundador de ID Software y programador de juegos como *Doom* (id Software, 1993) o *Quake* (id Software, 1996).

Arte

Se denomina artista a la persona o grupo encargado de generar los componentes gráficos del juego: modelos 3D de personajes y objetos, texturas, diseño de menús e interfaces, bocetos, animaciones y demás.

Existen varias categorías de artistas distintas dependiendo de en qué rama se especialicen y de cuál sea su rol en la estructura del proyecto[Bet03]:

- **Director Artístico:** Asignado al artista con mayor experiencia en la industria, el papel del director artístico es el de organizar y coordinar al resto de artistas para que realicen correctamente su trabajo y el de revisar las piezas producidas para asegurarse de que son consistentes con el estilo artístico establecido.

- **Artista Conceptual:** El artista conceptual es el encargado de producir bocetos provisionales que servirán como base para construir los gráficos definitivos del juego.
- **Artista 2D:** Los artistas 2D son expertos en las técnicas tradicionales del dibujo y pintura. Su rol es más notable en los juegos 2D, donde deben producir la mayoría de los componentes gráficos como fondos, *tiles* y *sprites*; pero también tienen un papel notable en los juegos 3D, donde suelen ser los encargados de diseñar interfaces gráficas, crear las texturas de los modelos 3D e incluso realizar trabajos ajenos al propio juego como la creación de imágenes promocionales.
- **Modelador 3D:** Es el encargado de producir los modelos 3D de los distintos componentes del juego tales como personajes, objetos, mapas... Es relativamente común que los modeladores 3D tengan ciertos conocimientos de programación debido a que eran necesarios para trabajar con los primeros programas de modelado 3d.
- **Animador:** Es el encargado de animar los diferentes elementos del juego, desde el simple movimiento de un molino de viento hasta las complicadas expresiones de una cara. Existen dos alternativas para realizar la animación: la técnica de keyframing, que consiste en realizar poses estáticas de los personajes que el programa utiliza para generar la animación; y la captura de movimiento, en la que los movimientos de un actor son capturados y transferidos al juego mediante un equipo especializado.
- **Storyboarder:** Es el artista encargado de diseñar escenas del juego. Para ello, el Storyboarder crea unas secuencias de arte conceptual que describen los tiempos, diálogos y eventos de las escenas, lo que permite valorarla y validarla antes de iniciar el costoso proceso de producción.



Figura 2.18: Akira Toriyama, el autor de Dragon Ball, ha trabajado como artista en juegos como *Dragon Quest* (Chunsoft, 1986) o *Chrono Trigger* (Square, 1995).

Audio

El trabajo de audio en un videojuego viene en tres categorías: música, efectos de sonido y doblaje. Existen especialistas que se dedican exclusivamente a una sola de estas categorías, aunque no es raro encontrarse en pequeños estudios a una persona encargarse tanto de la música como del sonido. Reflejando los tipos de audio, los tres tipos de profesionales son[Bet03]:

1. **Músico:** Es el artista encargado de escribir las composiciones musicales que se escucharán a lo largo del juego. Es muy común que el músico también se haga cargo de interpretar sus composiciones mediante programas de síntesis de música, aunque las grandes producciones pueden permitirse contratar interpretaciones en vivo.
2. **Técnico de sonido:** Los técnicos de sonido son profesionales que se dedican a fabricar o adaptar sonidos para el proyecto en el que trabajen.
3. **Equipo de doblaje:** El trabajo de doblar un videojuego requiere del trabajo de varios profesionales. En primer lugar, está el actor de voz, un actor especializado que interpreta con su voz a uno o varios personajes del juego. El trabajo de los actores está supervisado por un director de doblaje, que además suele encargarse de adaptar el guion y de dirigir a los técnicos de sonido que van a grabar y manipular las voces.



Figura 2.19: Yoko Shinomura es una compositora conocida por su trabajo en videojuegos como *Final Fantasy XV* (Square Enix, 2016) o *Street Fighter II: The World Warrior* (Capcom, 1991).

Testing

El Aseguramiento de la Calidad (o QA por sus siglas en inglés) es un requisito clave para el desarrollo de un videojuego, a la vez de un proceso lento y costoso que debe comenzarse lo más pronto posible para evitar un sobrecoste[Bet03]. El trabajo del tester es el de revisar las distintas versiones del juego en busca de fallos para que los

desarrolladores puedan arreglarlos.

Normalmente, los testers de un juego se agrupan en equipos, dirigidos por un jefe de testing. Cada equipo de testers se encarga de revisar una faceta distinta del juego, el equipo principal se encarga de probar la jugabilidad y los modos de juego individuales, el equipo multijugador se ocupa de revisar la jugabilidad en línea, así como los componentes técnicos de las conexiones, el equipo de compatibilidad prueba el juego en diversas plataformas y PCs con distintos componentes y el equipo de localización comprueba que se halla realizado una correcta traducción a distintos idiomas.

Para evitar la pérdida de punto de vista critico que el equipo principal y el equipo de multijugador pueden sufrir tras haber trabajado con el juego desde el principio del desarrollo, es normal cambiar los equipos en las últimas etapas del desarrollo por equipos nuevos que no estén involucrados en el juego.

Junto al trabajo de los equipos profesionales de testing se suelen realizar campañas de Beta Testing. El Beta testing consiste en liberar una versión incompleta del juego para que jugadores aficionados los prueben. Las resultados y opiniones de los jugadores son recogidos para utilizarse en el desarrollo de la versión completa. Para realizar una exitosa campaña de beta testing es necesario contar con uno o más organizadores que puedan gestionar la retroalimentación de los usuarios.

Producción

La función principal del productor es servir de puente entre el equipo de desarrollo y el resto de la empresa. El productor debe tener un conocimiento profundo del juego y de los demás miembros del equipo, de forma que pueda explicarlo de forma correcta en las muchas reuniones que se tendrán con otros departamentos, como por ejemplo el de marketing[Bat04].

El productor se encarga de realizar la gestión del proyecto, coordinando al equipo, realizando la programación de las etapas del proyecto y gestionando los posibles riesgos.

Existen tres tipos de productores dependiendo de su especialidad. Estos son:

1. **Productor Externo:** Este tipo de productor trabaja para la compañía editora y se encarga de supervisar al equipo de desarrollo para asegurarse de que se cumplen los acuerdos establecidos por ambas partes.
2. **Productor Interno:** Esta clase de productor trabaja en la compañía desarrolladora y se encarga tanto de realizar una gestión interna del proyecto como de actuar de representante de del equipo.
3. **Asistente de Producción:** Los asistentes de producción se encargan de realizar las tareas a las que el productor jefe del proyecto no puede dedicarse personal-

2. ESTADO DEL ARTE

mente. Normalmente se trata administrar detalles concretos como administrar recursos, realizar el papeleo o gestionar los servidores y la página web.

2.3 Motores de juegos

2.3.1 Descripción

Un motor de juego es un framework software que facilita el desarrollo de videojuegos proveyendo al programador de la funcionalidad general necesaria para cualquier juego[War].

El término “Motor de Juegos” se remonta a mediados de los años noventa, cuando apareció el género de los juegos de Tiros en Primera Persona. *Doom* (id Software, 1993), uno de los primeros juegos de este género, había sido diseñado de forma que existía una separación bien definida entre los componentes software principales (como el motor de renderizado 3D o el sistema de detección de colisiones) y los assets gráficos, los mundos y las reglas del juego. Esta separación permitía el desarrollo de nuevos juegos solo cambiando el arte, niveles o armas de títulos anteriores, manteniendo intacto el motor[Gre09].



Figura 2.20: *Heretic* (Raven Software, 1994), es un juego desarrollado con el motor de *Doom*.

Este concepto inició las comunidades de “Modding”, que son grupos de aficionados que desarrollaban juegos nuevos modificando juegos antiguos, y para finales de los noventa, juegos como [Sof99] y [Meg98] habían sido diseñados pensando en la reusabilidad

de su código. Actualmente, la mayor parte de las compañías desarrolladoras de juegos adquieren la licencia de motores desarrollados por terceros, mucho más económico que desarrollar desde cero todos los componentes.

La linea que separa el motor del juego suele ser difusa y depende en gran medida del juego concreto. El principal factor que se utiliza para distinguir un motor de juego de un juego que haya sido desarrollado de forma “íntegra” es la presencia de una Arquitectura orientada a datos. Se trata de un paradigma de programación que se basa en diseñar software con la intención de procesar datos, en lugar de ejecutar secuencias instrucciones fijas.

Idealmente, debería ser capaz de “reproducir” cualquier juego a partir de los datos de su contenido, de la misma forma que un programa reproductor de música leer y reproducir canciones. Sin embargo, en la realidad los motores de juegos suelen estar optimizados para un determinado género juego o una plataforma específica debido a que de esa forma es posible obtener software más eficiente y con mayores prestaciones, aplicando técnicas y patrones de diseño específicos de esos géneros/plataformas.

Componentes de un Motor

Los motores de juego son softwares muy complejos, por lo que suelen estar construidos a partir de módulos independientes, lo que facilita enormemente su mantenimiento. Normalmente, un motor de juegos se compone de los siguientes módulos[Gre09]:

- **El Núcleo:** Se trata de una aplicación compleja que recoge gran cantidad de herramientas y utilidades necesarias para el desarrollo del juego. El núcleo suele incluir una librería de funciones matemáticas, herramientas para la gestión eficiente de memoria, estructuras de datos y clases personalizadas, etc.
- **Motor de Renderizado:** Posiblemente el componente más grande y complejo del juego, el motor de renderizado es el software encargado de generar los gráficos del juego. Estos motores suelen estar construidos siguiendo una estructura de capas: primero el **render de bajo nivel**, que se encarga de dibujar primitivas a la mayor velocidad posible; el **grafo de escena** determina qué sección de la escena es visible, lo que permite reducir el número de llamadas al render de bajo nivel; la capa de **efectos visuales**, que contiene el sistema de partículas, los efectos de pantalla completa, o las luces dinámicas; y finalmente la capa de **frontend**, la cual sirve para el renderizado de imágenes 2D que se superpondrán a la escena tridimensional del juego, como los menús, el HUD (Head Up Display) o videos pre-renderizados. Aparte del motor gráfico, los motores de juego actuales suelen incluir también un **Sistema de Animación**, el cual permita dotar de movimientos naturales a los personajes y elementos del juego.
- **Gestor de Recursos:** Se trata de una interfaz que permite un acceso unificado

2. ESTADO DEL ARTE

a los distintos assets que forman el juego (modelos, texturas, sonidos, scripts...).

- **Motor de Físicas:** El motor de físicas permite realizar la detección de colisiones entre entidades del juego, así como la simulación de comportamientos físicos realistas para dichas entidades. Hoy en día, las compañías no suelen programar sus propios motores de físicas, en su lugar adquieren motores desarrollados por terceros, como *Havok*⁹ o *PhysX*¹⁰.
- **Entrada y Salida del Jugador:** Este sistema se encarga de gestionar la información de entrada del jugador, suministrada mediante el mando de juego o el teclado y ratón. El sistema toma la información en bruto de entrada y permite al programador acceder a ella de forma más útil, limpiando los datos de entrada, creando eventos de activación de teclas y proveyendo de sistemas para mapear funciones a distintas teclas o botones y para detectar secuencias de pulsaciones. Este sistema también provee funciones para la salida de datos relacionado con los mandos de control, como activar y desactivar la vibración o emitir sonidos.
- **Sistema de Audio:** Es el componente encargado de la reproducción de la música y efectos de sonido del juego. Aunque su complejidad depende en gran medida de las necesidades del motor concreto, la mayoría incluyen sistemas para producir efectos como sonido 3D o música dinámica.
- **Networking:** Son los sistemas encargados de realizar la conexión del juego con Internet para, en la mayoría de los casos, realizar partidas en línea con otros jugadores. El soporte de sistemas para el juego multijugador tiene un gran impacto en la mayoría de componentes del motor, por lo que estos suelen ser desarrollados pensando desde el principio en el modo multijugador, e implementando el modo de un jugador como un caso específico del multijugador.
- **Fundamentos de la Jugabilidad:** Esta capa implementa una serie de sistemas que permiten implementar la Jugabilidad. Suele incluir un lenguaje de Scripting, un sistema de eventos, inteligencia artificial, cámaras...
- **Herramientas de depuración:** Estas herramientas facilitan la tarea de depurar y optimizar el juego. Incluyen herramientas para dibujar en pantalla, consolas de comandos, sistemas para grabar y reproducir sesiones de juego...

2.3.2 Ejemplos de Motores

Unity 3D

Unity¹¹, conocido popularmente como **Unity3D**, es un motor de juego con su propio Entorno de Desarrollo Integrado (IDE) lanzado en el año 2005 por la compañía Unity

⁹<https://www.havok.com/physics/>

¹⁰<https://www.geforce.com/hardware/technology/physx#source=gss>

¹¹<https://unity3d.com/unity>

Technologies. El desarrollo de este motor comenzó en el año 2002, encabezado por los desarrolladores David Helgason, Joachim Ante y Nicholas Francis. Se trata de uno de los motores más utilizados del mercado, especialmente para el desarrollo de juegos para dispositivos móviles (más del 30 % de los 1000 juegos más exitosos para dispositivos móviles fueron desarrollados con este motor).

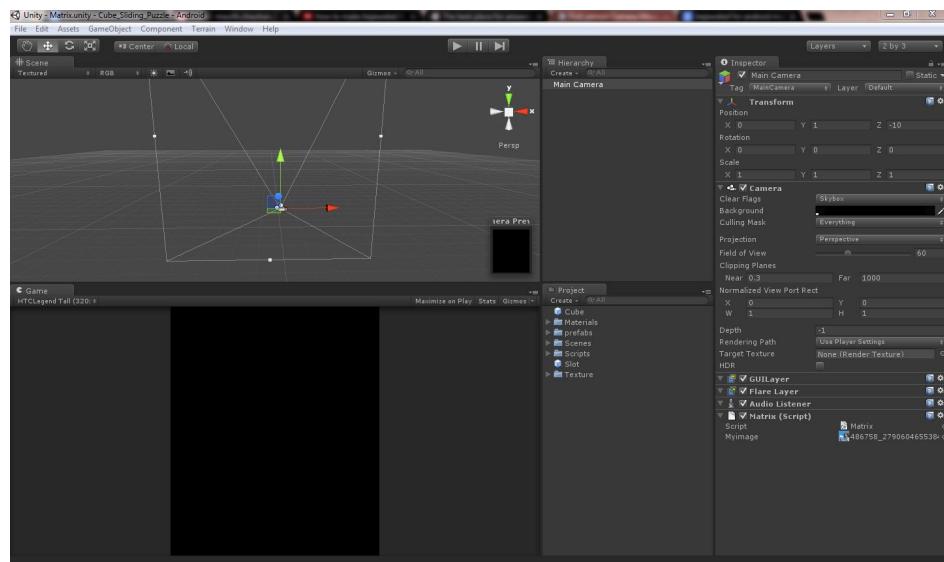


Figura 2.21: Captura del entorno de desarrollo de Unity

La cualidad más importante de Unity es la facilidad de su uso: El motor cuenta con una arquitectura de componentes fácil de entender, el proceso de exportación es muy simple y además cuenta con una tienda de Assets integrada. El Scripting en Unity se realiza en los lenguajes **C#** o **UnityScript** (una variante de JavaScript). Unity ofrece la posibilidad de exportar juegos a 27 plataformas distintas, desde dispositivos móviles a consolas de sobremesa, pasando por los sistemas operativos más comunes de los ordenadores personales. Debido a que su uso es muy extendido, Unity cuenta con una amplia base usuarios, por lo que es sencillo encontrar ayuda en los foros de desarrollo. La facilidad de uso viene a costa de la potencia: su motor gráfico no es tan potente como el de otros motores de la competencia y su rendimiento no es muy óptimo para el desarrollo de juegos de gran envergadura.

Unity Technologies ofrece distintos planes de pago a los desarrolladores interesados en adquirir la licencia de su motor: La versión gratuita, Unity Personal, que ofrece la funcionalidad básica; Unity Plus que por 35€ al mes ofrece mayor personalización y herramientas de monetización y análisis de rendimiento; y la versión profesional, Unity Pro (125€ al mes) la cual ofrece las prestaciones de la versión Plus más acceso al código fuente y mejor atención al cliente.

2. ESTADO DEL ARTE



Figura 2.22: A Hat in Time (Gears for Breakfast, 2017)



Figura 2.23: Hollow Knight (Team Cherry, 2017)

Figura 2.24: Juegos desarrollados con Unity3D

Unreal Engine

El **Unreal Engine**¹² es un popular motor de juego desarrollado por la compañía Epic Games. Originalmente desarrollado como motor propietario para el juego *Unreal* (Epic MegaGames, 1998), Epic Games pronto empezó a cerrar tratos con otras compañías que querían utilizar el motor en sus proyectos. Actualmente el motor se encuentra en su versión 4 y es uno de los más populares del sector, habiendo ganado incluso el premio Guinness al "motor de juegos más exitoso" con un total de 408 juegos (a fecha de julio de 2014) desarrollados con Unreal.

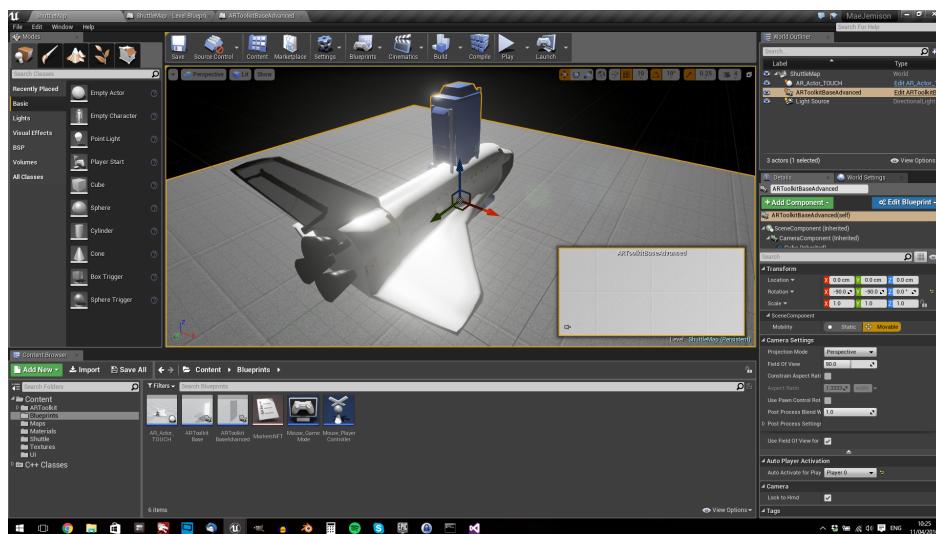


Figura 2.25: Captura del entorno de Unreal Engine

Unreal Engine es un motor orientado al desarrollo de juegos “AAA”, proyectos de gran envergadura llevados por equipos grandes. Uno de sus puntos fuertes es su potente motor de rendering el cual da soporte a gráficos fotorrealistas y permite el uso de efectos de post-procesados complejos entre otras características. El scripting en Unreal se

¹²<https://www.unrealengine.com/>

realiza mediante el sistema **Blueprint** de Scripting visual, el cual permite programar conectando de forma gráfica bloques de código. El motor permite también escribir código directamente en C++, lo que aumenta su flexibilidad. El paquete de herramientas del motor también incluye programas como generadores de terreno, editores de materiales, herramientas para animación... Sin embargo, se trata de un motor complicado y difícil de aprender a utilizar, además de que la potencia que requiere lo hace poco adecuado para el desarrollo para plataformas móviles.



Figura 2.26: Unreal (Epic Games, 1998)

Figura 2.27: Batman: Arkham Knight (Rocksteady Studios, 2015)

Figura 2.28: Juegos desarrollados con Unreal Engine

La licencia de uso de Unreal Engine es gratuita, sin embargo, en proyectos comerciales Epic Games cobra un 5 % de las ganancias a partir de los 3.000\$ por trimestre. En casos especiales, es posible negociar otros tipos de licencias con Epic Games.

Game Maker Studio

Game Maker Studio¹³ es un motor de juegos desarrollado por Yoyo Games. El programa fue lanzado originalmente en 1994 bajo el nombre de **Amino** como una herramienta para la creación de animaciones. Desde entonces, el programa ha ido evolucionado hasta convertirse en un motor de juegos de calidad profesional.

Game Maker Studio está diseñado para ser muy sencillo de usar: su principal uso es como una herramienta para gente sin conocimientos de programación, la cual permite desarrollar juegos sin necesidad de escribir una línea de código; o para el desarrollo rápido de juegos pequeños o prototipos. La programación de scripts en Game Maker se realiza mediante el sistema “**Drag and Drop**”, con el que se programa conectando diversos bloques de código; o el mediante un lenguaje de programación propio llamado **Game Maker Language**. El motor permite exportar con facilidad a distintas plataformas como PC, dispositivos móviles o HTML5. El entorno integrado de Game Maker

¹³<https://www.yoyogames.com/gamemaker>

2. ESTADO DEL ARTE

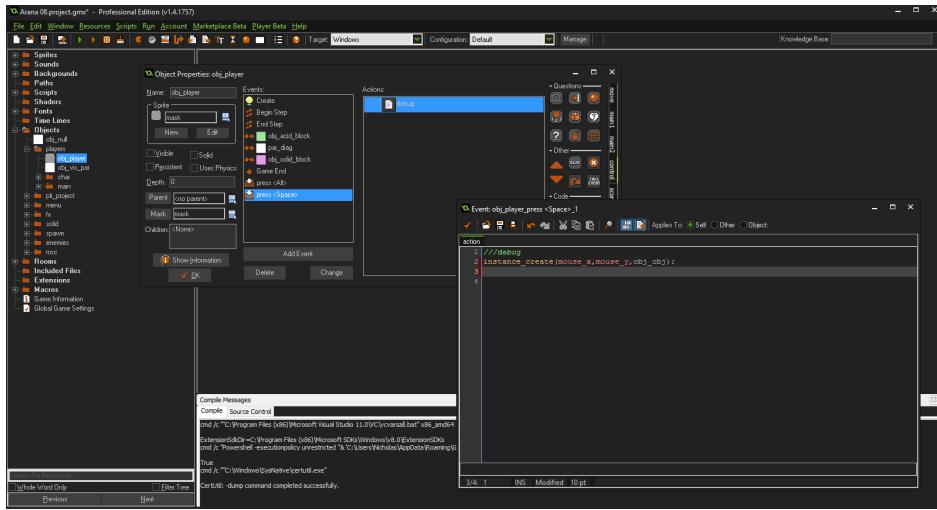


Figura 2.29: Captura del entorno de Game Maker Studio

incluye herramientas complementarias como un editor gráfico y un editor de mapas para centralizar el desarrollo. Sin embargo, la sencillez del motor también se refleja en su potencia: Game Maker Studio carece de soporte para gráficos tridimensionales complejos, y su arquitectura dificulta el desarrollo de proyectos de gran envergadura.



Figura 2.31: Hyper Light Drifter (Heart Machine, 2016)

Figura 2.30: Undertale (Toby Fox, 2015)

Figura 2.32: Juegos desarrollados con Game Maker

La licencia de la versión actual de Game Maker Studio (Game Maker Studio 2) se encuentra a la venta por distintos precios dependiendo de la plataforma de distribución para la que se quiera trabajar: desde la versión básica por 39\$ anuales hasta la versión profesional permanente con posibilidad de exportación a IOS, Android y consolas por 399\$. Existen también una versión de prueba gratuita que cuenta con unas prestaciones reducidas y un plan de pago para su uso en centros educativos.

2.4 Inteligencia Artificial en Videojuegos

2.4.1 Historia

Desde los principios de la inteligencia artificial, los expertos han dedicado una importante cantidad de tiempo y esfuerzo para construir sistemas inteligentes con el propósito de que pudiesen jugar a juegos con un nivel igual o superior al humano. Este enfoque en el estudio de los juegos se debe a que ofrecen un campo de estudio ideal para la inteligencia artificial: los juegos son problemas complejos que ofrecen desafíos para múltiples campos de la inteligencia artificial y además son tan populares que se dispone de cantidades inmensas de información sobre ellos[YT18].

Los primeros programas capaces de jugar contra humanos surgieron en los años cincuenta. Uno de los ejemplos más antiguos es el algoritmo ajedrecista de Alan Turing de 1948[Tur53], el cual era “ejecutado” en papel por un humano que seguía manualmente los pasos del algoritmo. El primer Software capaz de jugar a un juego fue la inteligencia artificial para el juego *OXO* (Alexander S. Douglas, 1952). En 1959, Arthur L. Samuel[Sam59] programó una inteligencia artificial para jugar a las Damas la cual contaba con un sistema de aprendizaje.

Sin embargo, estos programas eran muy simples y no planteaban reto alguno contra jugadores experimentados cuando se trataba de juegos con una cierta complejidad, como el ajedrez. Hubo que esperar a los años noventa para que empezaran a surgir programas capaces de derrotar a grandes maestros de distintos juegos: en el año 1994 el programa Chinook Checkers derrotó al campeón mundial de la Damas Marion Tinsley¹⁴ y 3 años más tarde el super-ordenador Deep Blue venció a Gary Kasparov¹⁵. Hoy en día, la inteligencia artificial ha demostrado ser capaz de superar a los jugadores humanos en casi cualquier juego, con ejemplos tales como la inteligencia artificial Watson ganando el concurso de televisión Jeopardy en 2011¹⁶ o el programa AlphaGo derrotando a Ke Jie, el jugador número uno de Go¹⁷ (un juego con una complejidad varios ordenes de magnitud superior al ajedrez).

Hoy en día, con el auge de los videojuegos, ha comenzado el estudio para la resolución de juegos continuos en tiempo real (en contraste a los juegos de mesa tradicionales, que son discretos). Estos juegos presentan un desafío mayor, pero ya se han realizado avances, como el algoritmo desarrollado por Google DeepMind en 2014 para la resolución de varios juegos de la consola clásica Atari 2600¹⁸.

¹⁴<https://webdocs.cs.ualberta.ca/~chinook/project/>

¹⁵<http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/>

¹⁶https://www.youtube.com/watch?v=WFR3lOm_xhE

¹⁷<https://www.theverge.com/2017/5/25/15689462/alphago-ke-jie-game-2-result-google-deepmind-china>

¹⁸<https://deepmind.com/research/publications/playing-atari-deep-reinforcement-learning/>

2. ESTADO DEL ARTE



Figura 2.33: El Gran Maestro de ajedrez Gari Kaspárov (izquierda) enfrentándose al ordenador Deep Blue

2.4.2 IA Aplicada a Videojuegos: Contexto Actual

El uso de la Inteligencia Artificial en la industria del videojuego difiere en varios puntos a su aplicación habitual en el campo académico de los juegos que hemos visto en el apartado anterior. La principal diferencia es el tipo de problema que se intenta resolver utilizando inteligencia artificial en cada una de las ramas: en la Inteligencia Artificial aplicada a jugar a juegos se busca obtener un sistema capaz de jugar de manera óptima para derrotar a cualquier adversario humano, sin embargo, en la Inteligencia Artificial orientada a videojuegos lo que se busca es crear sistemas con los que mejorar la experiencia de juego del jugador. Esta diferencia de objetivos hace que la Inteligencia Artificial no se aplique únicamente a la creación de oponentes virtuales, sino también al modelado del comportamiento de Personajes No Jugadores y a la Generación Procedimental de Contenido[YT18].

En el ámbito de los videojuegos, a la Inteligencia Artificial que interacciona con el juego como un jugador más suele llamarse "Bot". Estos bots predominan en juegos competitivos donde es necesario un oponente con un grado de inteligencia elevado para suponer un reto al jugador, como los juegos de estrategia, los juegos de lucha o los juegos de disparos en primera persona. Debido al elevado coste y complejidad de desarrollar una Inteligencia Artificial potente para estos juegos, por no hablar de la potencia requerida para ejecutarla, en la mayor parte de los videojuegos la Inteligencia Artificial "hace trampas", juega teniendo acceso una serie de ventajas que los jugadores humanos no tiene. Estos sistemas podrían, por ejemplo, acceder a información oculta del jugador (como la posición o recursos) a la hora de planificar estrategias o dispondrían de más y mejores recursos que sus adversarios.

En la mayoría de los juegos, la Inteligencia Artificial no se dedica al modelado de bots como los descritos anteriormente, sino que lo más común es que se utilice para controlar el comportamiento de Personajes no Jugadores, o NPCs (siglas inglesas de Non-Player Character). Estos pueden tener comportamientos muy variados dependiendo de su papel en el juego: pueden actuar como adversarios, servir de ayuda para el jugador, formar parte de un puzzle, contar una historia o simplemente formar parte del trasfondo de la acción. En el diseño de NPCs, lo que se busca son dos cosas: crear una Ilusión de Inteligencia, de forma que los jugadores crean que el NPC es un ser inteligente, aunque sea controlado por un código muy sencillo, y, en especial con adversarios, que sea predecible, de forma que el jugador pueda desarrollar estrategias para enfrentarse/interaccionar con ellos.



Figura 2.34: En *The Sims* (Maxis, 2000), los personajes pueden tomar decisiones basándose en sus gustos y necesidades.

La otra aplicación principal en los videojuegos es la Generación Procedimental de Contenido. La generación Procedimental de Contenido es el nombre que reciben los métodos que permiten generar el contenido de un juego de forma automática o con solo un mínimo de intervención humana. Actualmente, la mayor parte de los juegos que hacen uso de la generación procedural la utilizan para la creación automática de mapas o niveles o para la creación de objetos. La generación procedural de contenido puede utilizarse tanto como una herramienta durante el desarrollo, que serviría para generar contenido que luego sería refinado por los desarrolladores; o podría formar parte del juego final, generando nuevo contenido al gusto del jugador de forma automática.

2. ESTADO DEL ARTE



Figura 2.35: *Minecraft* (Mojang, 2011) es un ejemplo claro de generación procedural de terrenos.

2.4.3 Métodos de la IA

Existen varios tipos de métodos y algoritmos los cuales pueden ser utilizados para construir inteligencias Artificiales. La elección entre los distintos tipos de métodos debe realizarse dependiendo del tipo de problema que se intenta resolver y de los recursos de los que se dispone, tanto las prestaciones del dispositivo en el que van a ser implementados como margen de tiempo máximo de la ejecución del algoritmo[YT18].

Los Métodos de la inteligencia artificial pueden ser agrupados en las siguientes categorías, debido a sus características y aplicaciones similares:

Métodos Ad Hoc

Esta clase de métodos de IA es una de la primera y, en el sector del videojuego, la más común. Su nombre proviene de la locución latina que, traducida, significa “para esto”, y hace referencia a que se tratan de soluciones precisas para problemas concretos, las cuales no pueden generalizarse ni aplicarse a otros problemas distintos[YT18].

Pese al notable problema que provoca la falta de reusabilidad de estos métodos, son los más utilizados en el desarrollo de videojuegos. Esto se debe a que, por lo general, son muy fáciles de diseñar, visualizar, implementar y depurar; además requerir de muy pocos recursos cuando se aplican a problemas pequeños.

Algunos tipos de métodos Ad hoc son:

- **Máquinas de estados finitos:** Es un tipo de sistema experto en el que la información es representada por un grafo dirigido, donde los nodos representan **Estados** en los que se puede encontrar un comportamiento, proceso o algoritmo y las aristas las **Transiciones** condicionales entre los distintos estados. Fácil de diseñar e implementar, pero su complejidad crece muy rápido, por lo que no es apto para problemas grandes.

- **Arboles de comportamiento:** Este tipo de sistema experto utiliza una estructura en árbol para el modelado de información, donde cada nodo es un **comportamiento**. El algoritmo recorre el árbol en profundidad, ejecutando el comportamiento de dicho nodo. El comportamiento del nodo padre depende del resultado de los hijos. Los arboles de comportamiento ofrecen una mayor flexibilidad que las máquinas de estados a cambio de mayor complejidad.
- **IA basada en utilidad:** Esta técnica se basa en el uso de una **Función de Utilidad**, la cual asigna un valor de utilidad a todas las opciones del agente inteligente basándose en información del entorno. El agente elige la opción que tenga una utilidad más alta.

Búsquedas en Árbol

Una de las bases de la inteligencia artificial son los **algoritmos de búsqueda en árbol** son una de las bases de la inteligencia artificial, dado que la mayor parte de los problemas de la inteligencia artificial podrían plantearse usando este tipo de algoritmos[YT18].

La búsqueda en árbol consiste en la construcción de un **Árbol de búsqueda**, un tipo de grafo dirigido en el cual los nodos o hojas representan estados mientras que las aristas o ramas representan las acciones que provocan la transición de un estado a otro. El agente inteligente recorre el árbol partiendo del nodo raíz buscando la secuencia de ramas que resuelvan el problema siguiendo un algoritmo concreto. Cuando se aplican a juegos, los arboles de búsqueda suelen aplicarse para modelar el estado del juego: la inteligencia artificial recorre el árbol buscando la secuencia de acciones que lleve al estado de victoria, o evitando el estado de derrota.

Existen numerosos algoritmos de búsqueda distintos, los cuales pueden agruparse en las siguientes categorías:

- **Búsqueda no informada:** Se trata de los algoritmos más básicos, en los que se realiza la búsqueda sin ningún tipo de información adicional sobre el objetivo. Sus variantes más simples son el algoritmo **Primero en Anchura**(explora todas las acciones de un estado antes de pasar al siguiente) y el **Primero en Profundidad**(explora una secuencia de ramas todo lo que puede antes de volver e intentar otra secuencia distinta)
- **Búsqueda Primero el Mejor:** En estos algoritmos se cuenta con información adicional sobre el nodo objetivo, la cual se utiliza para determinar que nodos deben explorarse primero. Un tipo de algoritmos Primero el Mejor son los algoritmos **A***, en los cuales los nodos se seleccionan basándose en su **Coste**(suma de la distancia entre el nodo y el nodo raíz y la distancia estimada al objetivo).

2. ESTADO DEL ARTE

- **Búsqueda MiniMax:** Este tipo de algoritmos se utilizan para resolver problemas que involucran a dos adversarios enfrentados. En estos algoritmos se va alternando entre jugadores **Min** y **Max**, los cuales intentan llegar a sus respectivos estados de victoria opuestos. El espacio de búsqueda de estos algoritmos suele ser muy grande, por lo que suelen usar funciones de evaluación para evitar recorrer el árbol de búsqueda completo.
- **Árbol de búsqueda de Monte Carlo:** Se trata de una familia de algoritmos diseñados para resolver problemas **No deterministas** y/o de **Información Imperfecta**. Para evaluar la calidad de un estado dado, el algoritmo utiliza simulaciones aleatorias de partidas a partir de ese estado. La siguiente acción será con la que empezó el mayor número de partidas victoriosas.

Algoritmos de Optimización

Los algoritmos de optimización, a diferencia de la búsqueda en árbol, se centran en obtener una solución correcta, ignorando los pasos que llevaron a esta. Para ello, el algoritmo empieza tomando una solución sub-optima, la cual va modificando en múltiples iteraciones utilizando una **función de Aptitud** como guía hasta obtener una solución con una Aptitud máxima[YT18].

Existen varios tipos de algoritmos de optimización, dependiendo de los métodos que elijan para la selección de la solución inicial, para la evaluación de aptitud o para la modificación:

- **Búsqueda Local:** Este tipo de algoritmos consisten en, dada una solución, revisar todas las soluciones que difieran de dichas soluciones por una distancia mínima. Si alguna de las soluciones mejora a la solución original, se remplaza la solución original por la nueva solución y repite el algoritmo; si no, la solución original es elegida como la solución óptima.
- **Algoritmos Evolutivos:** Los algoritmos Evolutivos son un tipo de algoritmos de optimización basados en la evolución por selección natural Darwiniana que se observa en la naturaleza. Su funcionamiento se basa en un conjunto amplio de soluciones candidatas. EN cada iteración del algoritmo, las distintas soluciones se “cruzan” para obtener soluciones mixtas, las cuales son evaluadas. Finalmente, se forma un nuevo conjunto de soluciones a partir de las soluciones más exitosas. El algoritmo se repite hasta obtener la solución más óptima.

Aprendizaje Automático

El aprendizaje automático es una rama de la computación que permite dotar a los ordenadores de la capacidad de “aprender” a realizar una tarea utilizando datos en lugar de programarlo específicamente para esa tarea[Sam59]. Los algoritmos de aprendizaje

automático suelen aplicarse a problemas donde es muy difícil, o incluso imposible, programar un algoritmo implícito que pueda resolverlo, como por ejemplo el filtrado de correos o la visión por ordenador.

Una forma de clasificar los algoritmos de aprendizaje automático es basándose en el tipo de **Retroalimentación** que reciben. De esta forma, surgen las siguientes categorías:

- **Aprendizaje Supervisado:** Se trata de algoritmos que extraer los atributos y características comunes de los integrantes de grupos etiquetados. Estos algoritmos funcionan recibiendo conjuntos de muestra formado por datos etiquetados en distintos grupos y categorías. Analizando los conjuntos de muestra, el algoritmo debe ser capaz de asignar nuevos datos a la categoría correcta. Existen muchos tipos de algoritmos de Aprendizaje supervisado, como por ejemplo las **Redes Neuronales**, que funcionan imitando la estructura de las neuronas del cerebro.
- **Aprendizaje por Refuerzo:** Los algoritmos de Aprendizaje por Refuerzo se inspiran en el **Conductismo Psicológico**, basándose en la forma en la que los animales aprenden a tomar decisiones basándose en los estímulos positivos y negativos de su entorno. Estos algoritmos suelen implementarse mediante un agente inteligente que interacciona con un entorno mediante acciones. Con cada acción, el agente recibe un estímulo positivo o negativo, que almacena con la intención de descubrir la secuencia de acciones que maximice el estímulo positivo a largo plazo mediante técnicas similares a los algoritmos de Optimización
- **Aprendizaje No Supervisado:** El aprendizaje No Supervisado son un conjunto de algoritmos que sirven para encontrar asociaciones y patrones en un conjunto de datos sin ningún tipo de información de refuerzo adicional. Existen varias aproximaciones a este tipo de algoritmos, como el **Clustering**, que consiste en agrupar elementos de un conjunto basándose en su similitud entre ellos y su diferencia con el resto de grupos.

2.4.4 Futuros campos de aplicación

La Inteligencia Artificial orientada a juegos evoluciona de manera paralela a los propios videojuegos, que viven ahora más que nunca una época dorada debido al aumento constante tanto en popularidad y prestigio, lo que supone una mejora tanto en la potencia de las máquinas que los ejecutan como en las técnicas que se utilizan en su desarrollo. Esta situación cambiante ha causado la aparición de nuevos problemas que se esperan poder solucionar con el uso de técnicas de inteligencia artificial.[YT18] A continuación mencionaremos algunos de los futuros campos de aplicación de las técnicas de inteligencia artificial:

Una las posibles aplicaciones de la Inteligencia Artificial es la de realizar **Testing** de

2. ESTADO DEL ARTE

juegos. El programa jugaría a los juegos en busca de fallos tanto informáticos como de diseño (como problemas de balanceo o maneras de hacer trampas en el juego) de forma automatizada, ahorrando una gran cantidad de trabajo a los desarrolladores. Aunque ya existen herramientas que ofrecen una forma rudimentaria de testing automático, aún es necesario mejorar aspectos como la categorización de los errores encontrados.

La Minería de Datos de Juego es otro uso prometedor de la Inteligencia Artificial. Se trata de una técnica alternativa al testing habitual de juegos en la que se recolecta y analiza la información de comportamiento de la base de jugadores de un juego dado con el fin de mejorar dicho juego[Yan12]. Esta técnica se ha popularizado gracias a la proliferación de juegos con un fuerte componente online que facilita la recogida de datos. Sin embargo los volúmenes de datos con los que se trabaja son tan masivos que los algoritmos de minería de datos actuales no son capaces de analizarlos completamente [Yan12].

La **Dirección de Juegos** consiste en una Inteligencia Artificial que modifica eventos del juego en tiempo real basándose en las reacciones de los jugadores con acciones tales como modificar la dificultad, reproducir música y sonido o modificar el entorno con tal de mejorar la experiencia del jugador. Actualmente, los juegos que mejor han implementado un sistema con esta propiedad es la saga Left 4 Death de Valve¹⁹. Se trata de una rama con mucho potencial que aún no ha sido explorado completamente.

Finalmente, una tarea de gran importancia para los juegos online, a pesar de no formar parte de los juegos en si, es la **motorización de los chats**. El gran volumen de mensajes que se envían a través de los chats de los juegos online más populares hace que sea imposible la moderación manual, lo que crea un entorno de juego toxico para los jugadores. Compañías como Riot Games están empezando a utilizar algoritmos de aprendizaje automático para entrenar sistemas para detectar y eliminar los mensajes inapropiados de los chats²⁰.



Figura 2.36: Ejemplo de comportamiento tóxico en *League of Legends* (Riot Games, 2009).

¹⁹<http://www.l4d.com/blog/>

²⁰<https://www.nature.com/news/can-a-video-game-company-tame-toxic-behaviour-1.19647>

Capítulo 3

Arquitectura

3.1 Estructura de clases del Proyecto

En este apartado, vamos a explicar la estructura software de este proyecto, mostrando las escenas en las que se divide el juego, las clases en las que se agrupa el código como estas se relacionan entre sí.

Sin embargo, antes de mostrar la estructura, realizaremos una pequeña exposición acerca de la forma de trabajar en Unity, hablando del lenguaje de programación usado y de los patrones de diseño en los que se basa este motor.

3.1.1 Escenas

Al igual que muchos otros motores de juegos en tres dimensiones, Unity organiza las entidades del juego haciendo uso de **Grafos de Escena**. Los Grafos de Escena son un tipo de estructura de datos que permite representar las relaciones jerárquicas que existen entre los distintos objetos del juego o “nodos” haciendo uso de un grafo dirigido sin hijos [Dav15]. El uso de este tipo de estructura permite gestionar con facilidad escenas tridimensionales complejas, permitiendo que se apliquen las transformaciones (traslación, rotación y escalado) y otras operaciones de un nodo padre a todos sus nodos hijos de forma automática.

La implementación de grafos de escena que Unity utiliza da soporte a la creación de múltiples escenas independientes, pudiéndose realizar cambios de escenas o cargar varias escenas simultáneamente. Esto nos permite dividir el juego en múltiples secciones separadas, cada una con su propia funcionalidad. Nuestro proyecto está formado por cuatro escenas:

- **Escena de Titulo:** Una sencilla escena de título con el nombre del juego. Al pulsar la tecla Espacio se inicia el juego.
- **Escena de Juego:** Es la escena donde sucede el juego en sí, y contiene todos los elementos que facilitan la jugabilidad. A pesar de que el juego tiene múltiples niveles, se utiliza una única escena de juego que carga desde un fichero la configuración de bloques correspondiente.
- **Escena de fin del Juego:** A esta escena se accede cuando el jugador pierde

3. ARQUITECTURA

durante el juego. Es muy similar a la pantalla de inicio ya que también cuenta con un texto y la lógica para iniciar la escena de juego.

- **Escena de Victoria:** Esta escena se muestra cuando el jugador completa todos los niveles del juego. Funcionalmente es idéntica a la escena de fin del juego.

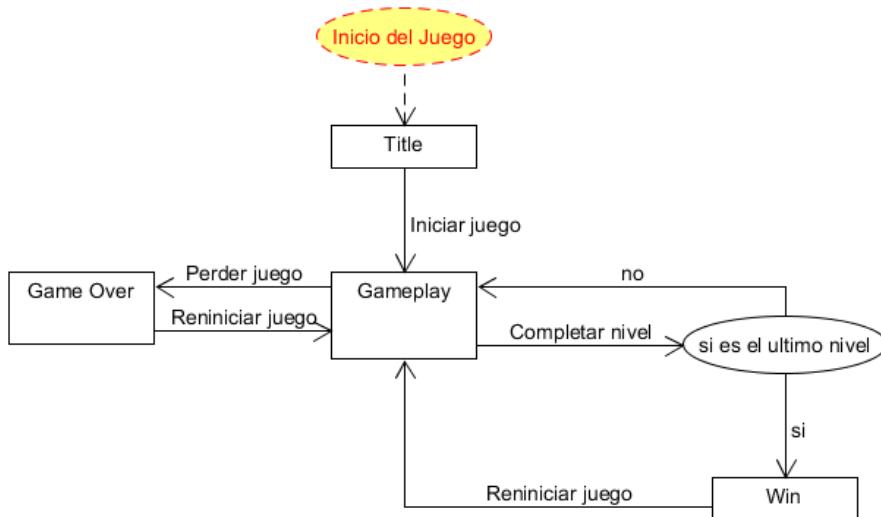


Figura 3.1: Diagrama de las escenas del juego

3.1.2 GameObjects, Componentes y Prefabs

En Unity, los nodos de escena son implementados mediante la clase **GameObjects**. El concepto de los GameObjects es uno de los más importantes a la hora de trabajar con Unity[Tec], ya que todos los objetos que el programador puede crear usando el editor de gráfico pertenecen a esta clase. Sin embargo, la clase en sí misma carece de funcionalidad. La funcionalidad viene dada de la adición de Componentes a los GameObjects, los cuales les otorgan distintas propiedades. Los componentes son un patrón de diseño muy utilizado que permite desacoplar el código de los programas manteniendo la posibilidad de crear entidades con mucha funcionalidad[Nys04].

Unity incluye un gran número de componentes por defecto, como los *Colliders* que se utilizan en la detección de colisiones, los *Renderers* que permiten asignar modelos y texturas a los GameObjects o los **Audio Emitters** con los que se puede emitir música y sonido.

De estos componentes por defecto destaca el **Transform**. Este componente sirve para almacenar y manipular la información relativa a la posición, rotación y escala de los GameObjects. Un *Transform* puede tener asignado otro *Transform* a modo de parent, creando estructuras jerárquicas en las que los cambios de posición, rotación y escala del parent se aplican también al hijo. Debido a que su funcionalidad es muy común y necesaria, este componente está incluido por defecto a todos los GameObjects.

Para la creación de nuevos componentes (llamados Scripts por Unity), el programador debe utilizar uno de los dos posibles lenguajes de programación soportados por Unity: **C#**, un lenguaje de programación orientado a objetos desarrollado originalmente por Microsoft para su framework; y **UnityScript**, un lenguaje basado en JavaScript creado específicamente para Unity. Para este proyecto se optó por usar C#.

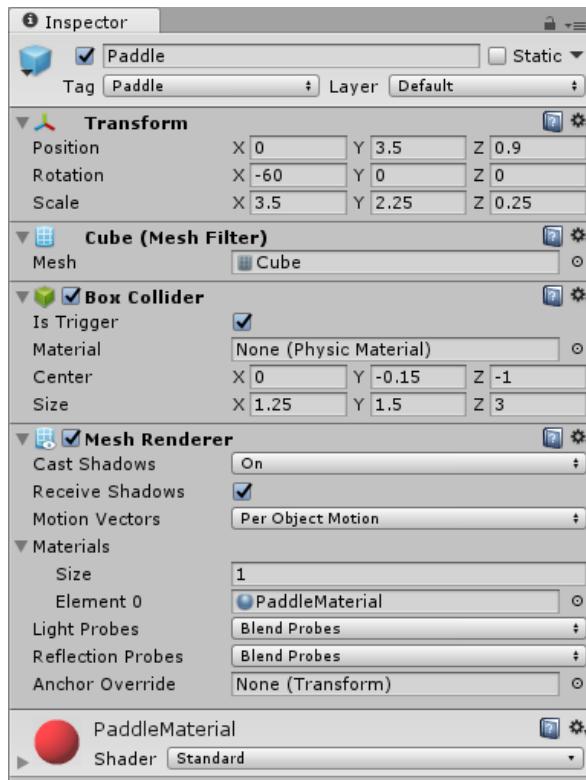


Figura 3.2: Inspector de objetos mostrando los componentes del objeto *Paddle*

Para que los scripts puedan funcionar como componentes, es necesario que sus clases hereden de la clase **MonoBehaviour** que incluye Unity. Esta clase contiene la funcionalidad necesaria para los componentes, como métodos de acceso a las propiedades del GameObject o funciones que son llamadas por Unity cuando ocurren ciertas acciones dentro del juego, como **Start**, que es llamado cuando crea el GameObject o **Update** que se llama una vez por cada fotograma del juego.

La creación de instancias de GameObjects, así como la asignación de componentes a estos se puede realizar tanto de forma gráfica mediante el editor de Unity como por código de programación. Como este proceso es muy lento y laborioso (especialmente realizándolo por código), Unity ofrece la posibilidad de crear múltiples GameObjects con el mismo conjunto componentes adheridos mediante el uso los Prefabs. Los prefabs son un tipo de Assets basados en el patrón de programación *Prototype*[Nys04] permite guardar GameObjects con componentes adheridos para poder generar copias a partir de él, tanto en el editor como por código en tiempo de ejecución.

3. ARQUITECTURA

3.1.3 Estructura de Clases

A continuación, se encuentra una gráfica con la estructura de clases del proyecto. Aclaramos que solo se incluyen las clases creadas expresamente para este proyecto concreto, ignorando las clases incluidas por defecto en Unity o por librerías externas utilizadas. También se han resumido sus atributos y métodos para facilitar la lectura.

La estructura de clases es la siguiente:

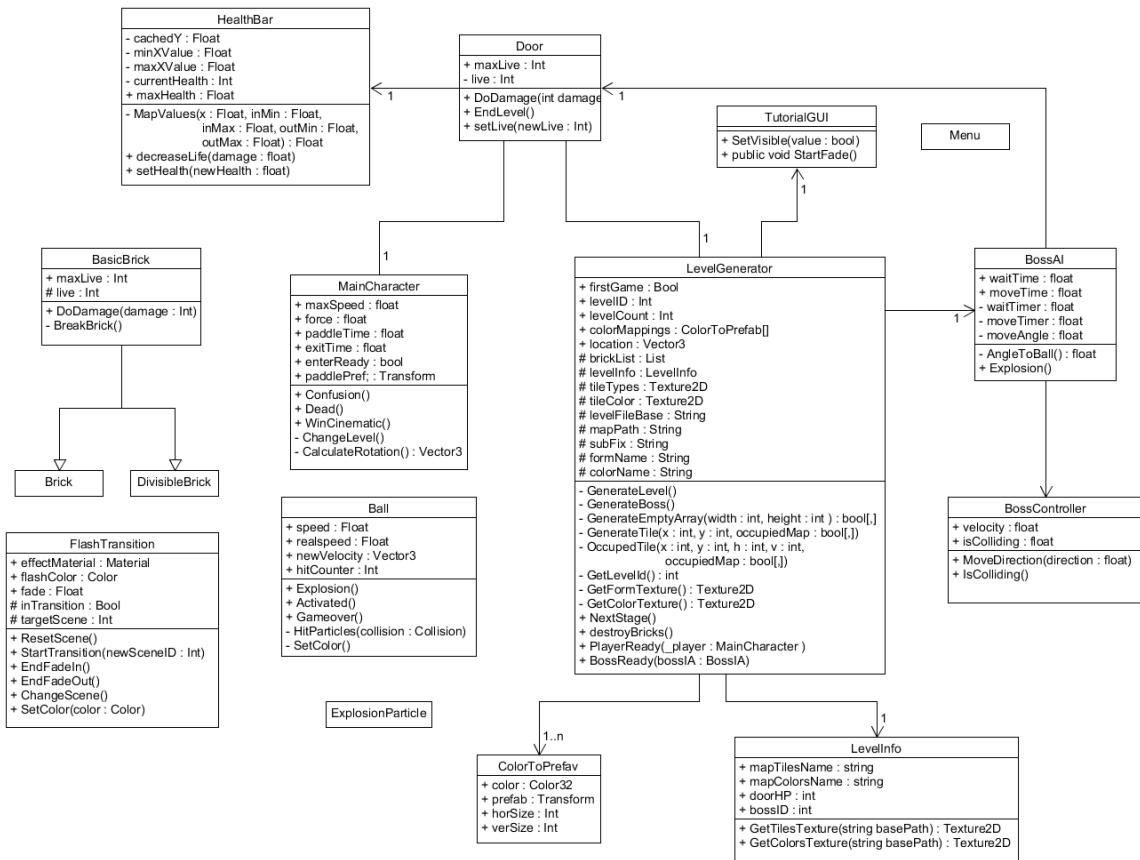


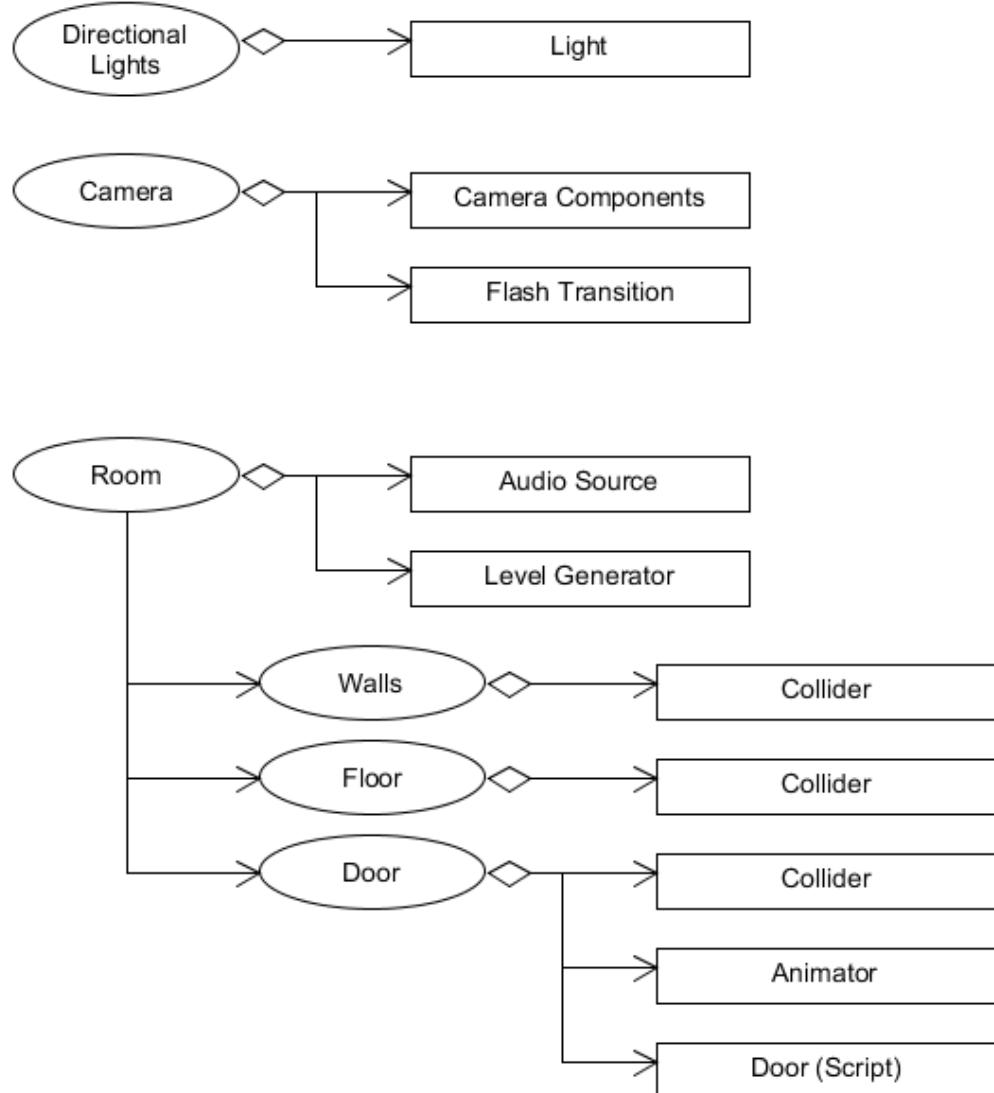
Figura 3.3: Estructura de clases

3.1.4 Prefabs

Sin embargo, la estructura de clases del juego no tiene mucho sentido por si misma si no se conoce como se asocian entre sí para formar GameObjects. Por ese motivo, en este apartado vamos a mostrar la colección de gameObjects que se utiliza en las distintas escenas del juego.

3. ARQUITECTURA

En primer lugar, la escena principal (donde se desarrolla la acción del juego) cuenta con los siguientes GameObjects:



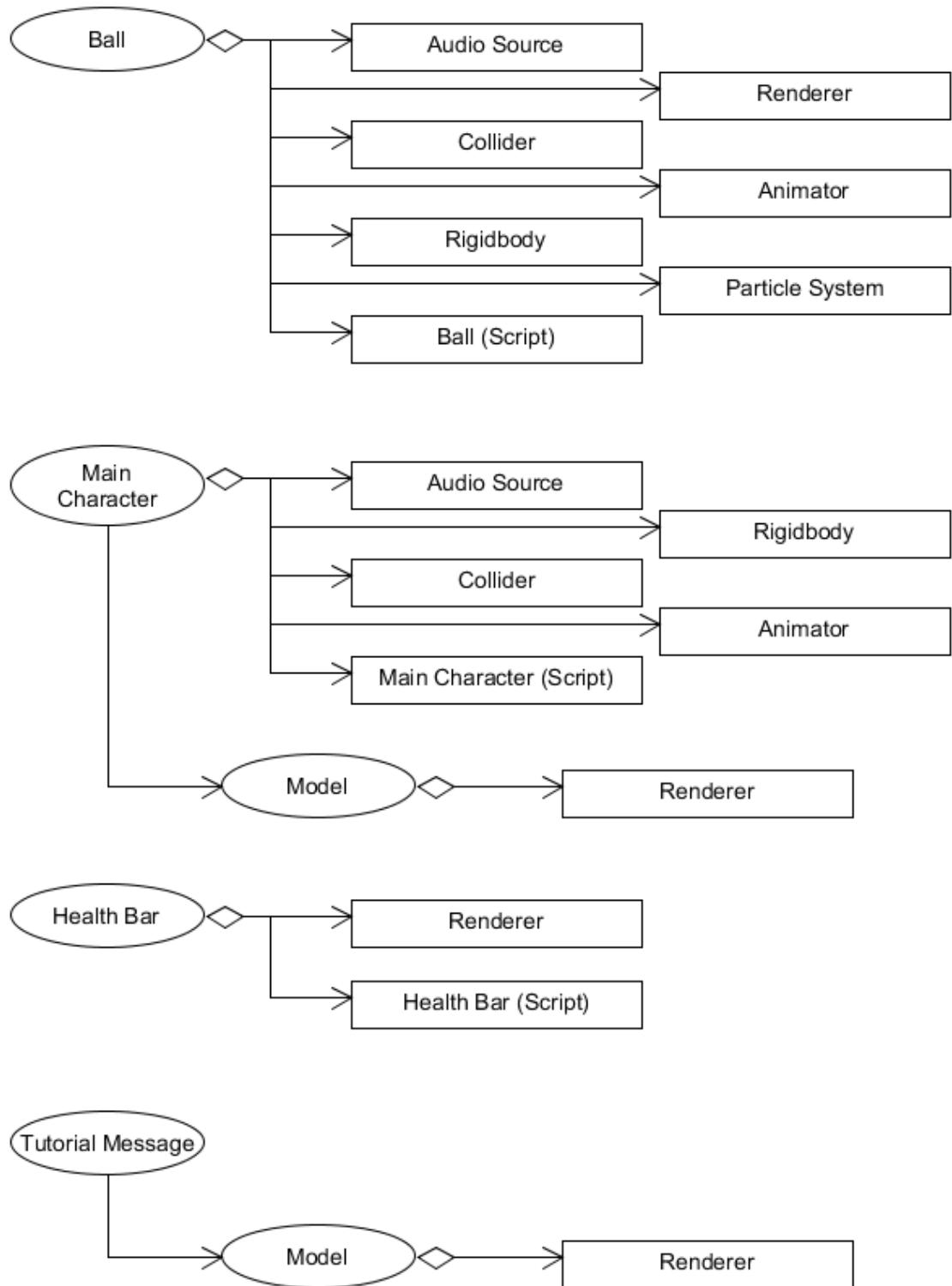
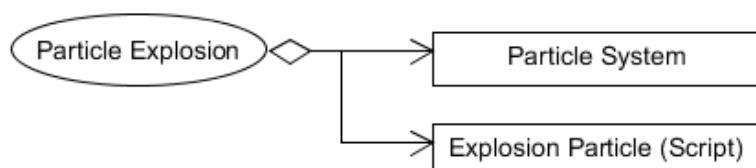
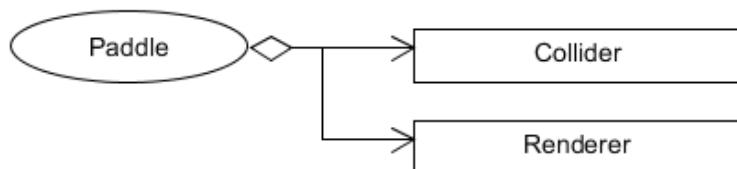
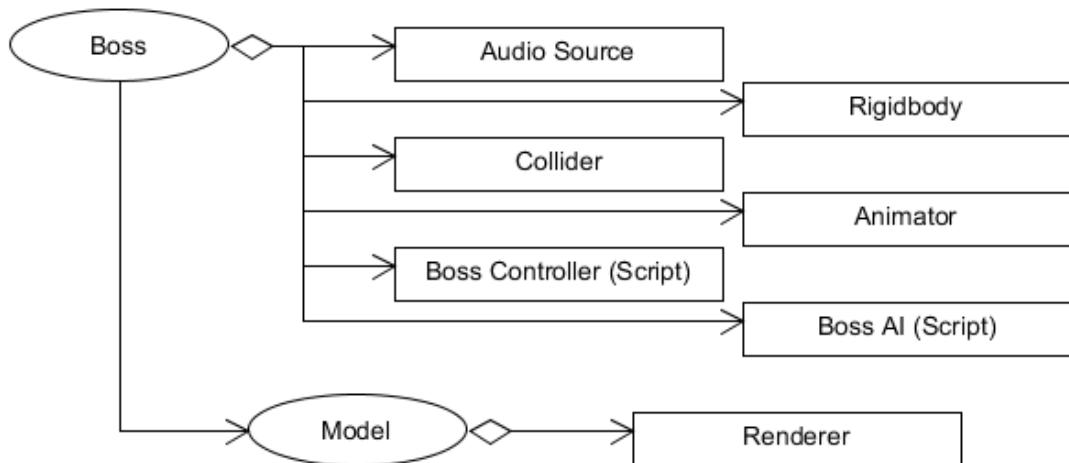
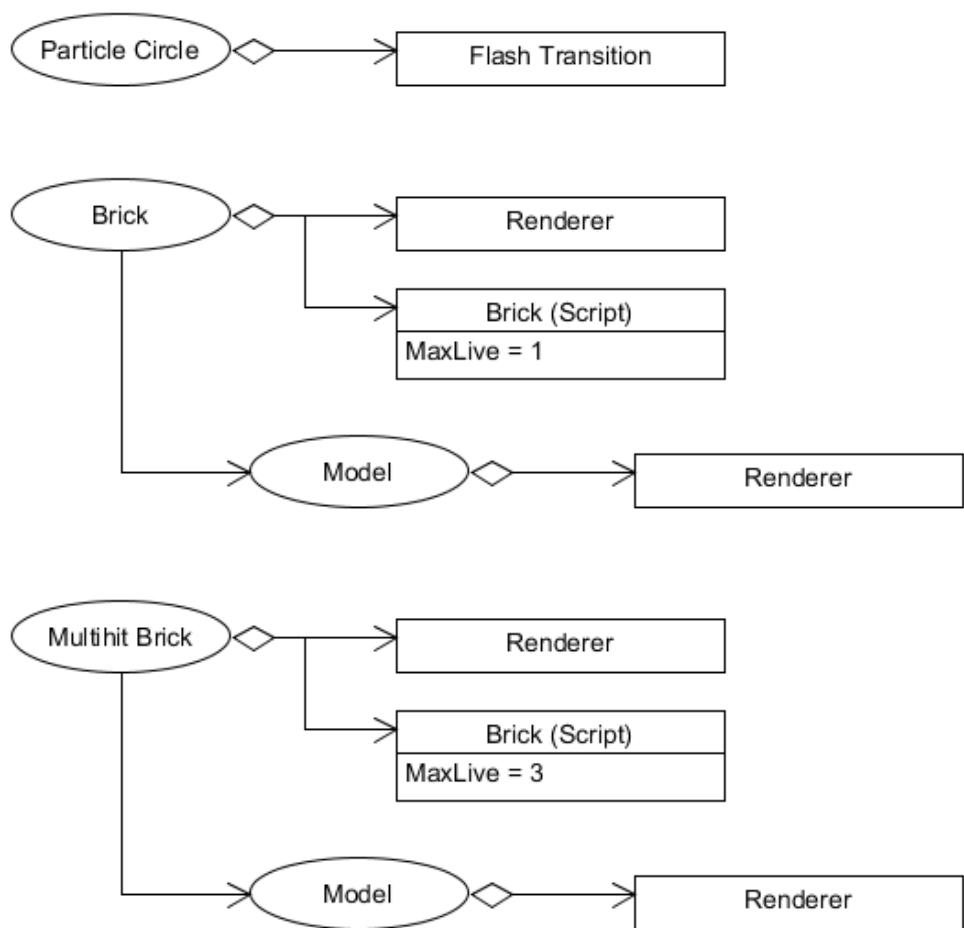


Figura 3.4: Lista de GameObjs en la escena principal

3. ARQUITECTURA

A estos GameObjects hay que añadirles una serie de Prefabs adicionales que no se encuentran en escena al inicio de la ejecución, pero son añadidos de forma dinámica cuando son necesarios. Estos prefabs son:





3. ARQUITECTURA

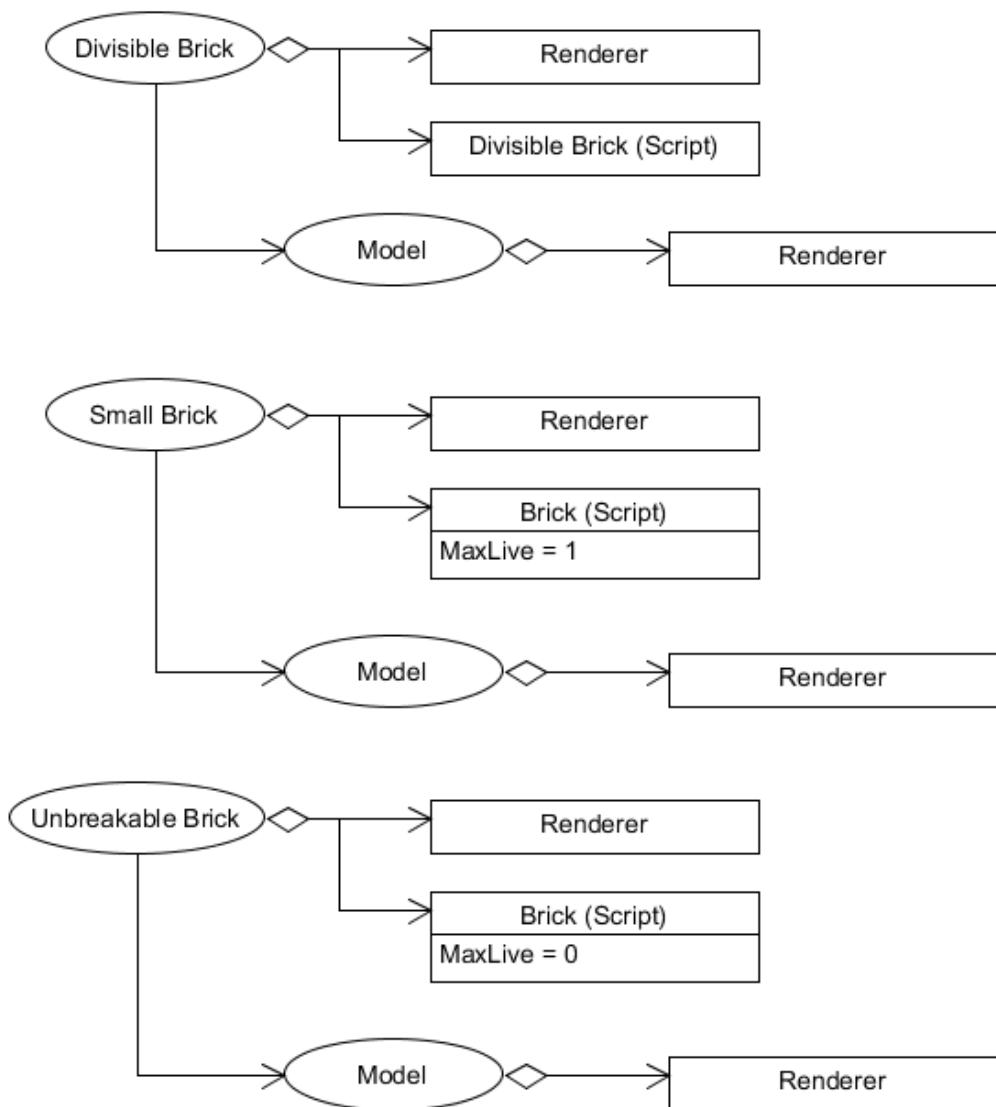


Figura 3.5: Prefabs de la escena Principal

Finalmente, la pantalla de título y final del juego cuentan con sus propios GameObjects (solo se mostrará una gráfica dado que la estructura de ambas escenas es prácticamente idéntica). Los GameObjects de estas escenas son:

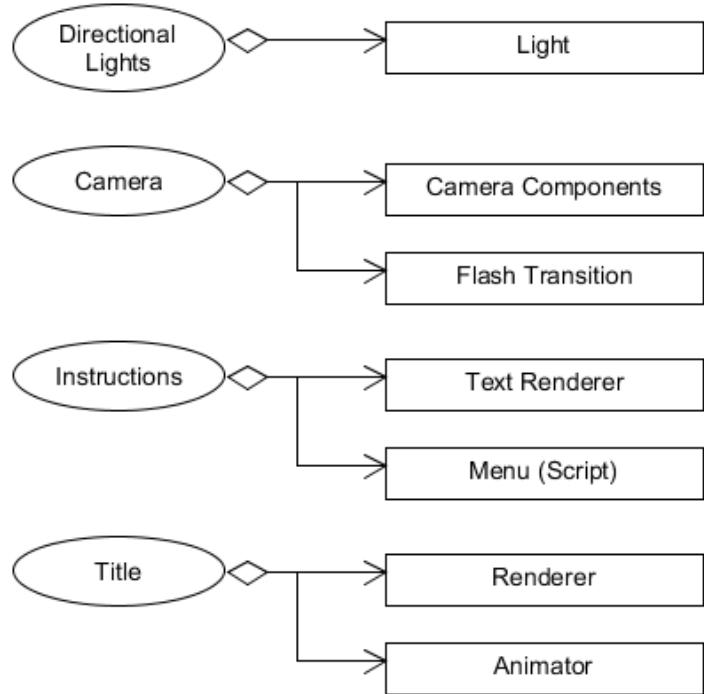


Figura 3.6: Lista de GameObjects en los menús

3.2 Carga y Estructura de Niveles

El diseño de los distintos niveles del juego fue un punto muy importante del desarrollo. Desde un principio se tenía claro que había que minimizar todo lo posible el tiempo que se requeriría para trasladar el diseño de un nivel a su complementación en Unity. Basándose en los requisitos de los niveles de este género de juego, las principales capacidades que se deseaban para el editor eran las siguientes:

- Precisión: Poder alinear los bloques con una cuadricula de forma automática, para así mantener la estética de juego “retro” deseada.
- Facilidad para añadir y eliminar niveles sin tener que modificar el código del juego.
- Posibilidad de elegir individualmente e color para cada uno de los bloques del nivel, independientemente de su comportamiento o posición. Esto nos permite decorar los niveles y hacerlos más memorables para el jugador.
- Simplicidad en el proceso de creación. En la medida de lo posible, la creación de niveles deberá poder hacerse de forma gráfica, sin escribir código.

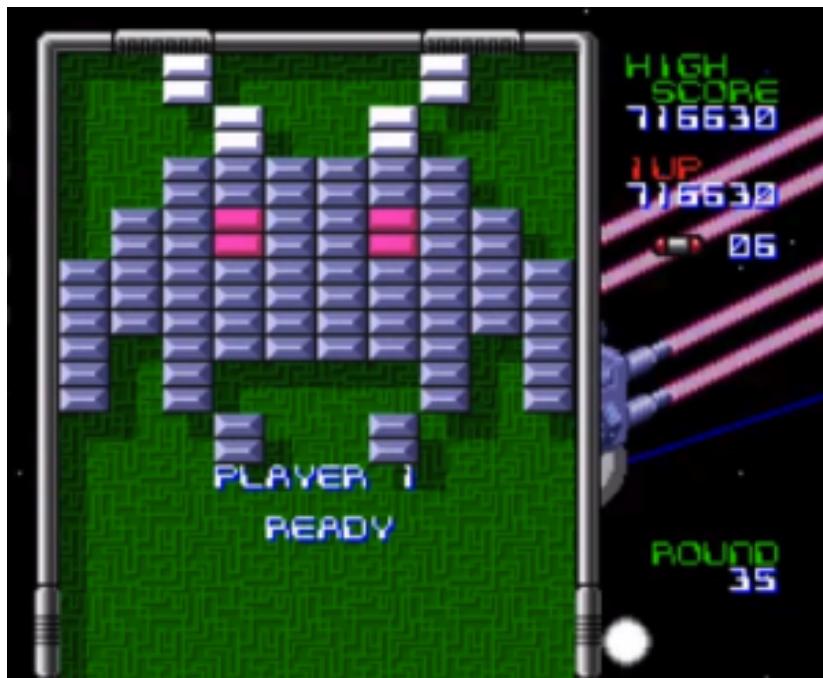


Figura 3.7: Homenaje a Space Invaders(Taito, 1978) dentro de Arkanoid: Doh it Again (Taito, 1997)

Tras una primera iteración en la que los niveles se implementaban en el propio editor de escenas de Unity, se optó por utilizar Tiled. Tiled es un editor de mapas de propósito general que permite editar mapas basados en baldosas o “Tiles”. Los mapas desarrollados generados con Tiles eran exportados al formato XML, el cual podía se

cargado en Unity. Sin embargo, la producción de niveles era demasiado lenta debido a determinadas carencias en el programa, lo que nos llevó a replantear el sistema.

El nuevo enfoque se basa en la lectura de pequeñas imágenes que contienen la información del nivel codificada en los colores de sus pixeles. En este sistema toma dos imágenes para generar el nivel. La primera imagen sirve para determinar el tipo y posición de los bloques, mientras que la segunda imagen determina los colores de cada bloque. Un archivo JSON adicional sirve para “enlazar” ambas imágenes y para contener información adicional, como la cantidad de golpes que debe recibir la puerta para abrirse o de si debe iniciarse o no un combate de jefe.

El sistema de carga de niveles está implementado en la clase LevelGenerator, un componente adherido al objeto Room. El proceso de carga de niveles funciona así:

1. Se lee el archivo JSON determinado por el índice del nivel actual. La lectura del archivo se realiza mediante la clase JsonUtility de Unity, que automáticamente genera un objeto con la estructura contenida el archivo JSON.
2. El sistema determina si se trata de un nivel de jefe. En ese caso se detiene el proceso y se empieza a preparar la batalla con el jefe. En caso contrario, se continua con el proceso.
3. Se cargan las imágenes del nivel en forma de dos texturas de la clase Texture2D. A efectos prácticos, cada textura es una matriz bidimensional de objetos Color. Llamaremos Textura de tipos a aquella que determina el tipo de los bloques y Textura de colores a aquella que determina sus colores.
4. Se recorre la textura de tipos obteniendo los valores de sus pixeles. Cada pixel se compara con una tabla de equivalencias que relaciona un tipo de bloque con un color. En caso de acierto, se instancia un bloque del tipo determinado, al cual se asigna el color del pixel de la textura de colores de la misma posición.
5. Dado que los bloques ocupan más de una “casilla”, cada vez que uno es creado se marcan las posiciones que ocupan en una “matriz de ocupación”. durante el recorrido de la textura, se ignorarán los pixeles que correspondan a posiciones ocupadas.
6. Finalmente, se asigna el número de golpes que requiere la puerta.

De esta forma es posible producir niveles rápidamente, los cuales son fáciles de modificar. La implementación del sistema está realizada de forma que es capaz de “ignorar” pequeños errores en el mapa, como asignación de colores a casillas vacías o la presencia de colores que no se corresponden con ningún tipo de bloque.

Sin embargo, este sistema no está exento de limitaciones:

- En primer lugar, solo una pequeña cantidad de estos pueden ser asignados a

3. ARQUITECTURA

tipos de bloques. Esto se debe a las discrepancias entre los formatos de colores (los componentes RGB de los colores se guardan como enteros entre 0 y 255 en las texturas, pero como números de punto flotante entre 0 y 1 en la clase Color), la cual provoca perdida de información durante la conversión y nos fuerza a utilizar valores de color que sean fracciones exactas de 256.

- En segundo lugar, se trata de un sistema poco escalable, permitiendo un tamaño único de campo de juego y poca capacidad para “personalizar” los tipos de bloque más allá de su color; por lo que sería necesario realizar cambios importantes en caso de que se necesitaran construir niveles más complejos

3.3 Control de movimiento y físicas

Los videojuegos son por definición sistemas interactivos en los que uno o más elementos de este reaccionan a las acciones del jugador. En nuestro caso, los dos elementos móviles que se encuentran a disposición del jugador son el personaje principal y la pelota.

El movimiento de estas dos entidades se implementó haciendo uso del motor de físicas incluido dentro de Unity Engine, el motor PhysX de Nvidia¹. El uso de un motor de físicas permitió agilizar el desarrollo ya que incluye mucha funcionalidad necesaria para el funcionamiento del juego, como la detección de colisiones, el movimiento con aceleración y desaceleración, la simulación de gravedad o el efecto de rebote de cuerpos en movimiento.

El comportamiento de las dos entidades se controla internamente mediante una máquina de estados finitos. El uso de este patrón de programación facilitó enormemente el desarrollo de estos objetos, ya que permitía controlar de forma precisa como iban a reaccionar las entidades en cada momento y reducía enormemente el tamaño de las lógicas de control, evitando así fallos difíciles de detectar[Nys04]. La implementación de la máquina de estados finitos se implementó mediante la librería *Simple Finite State Machine* de *Made With Monster Love*²

En los siguientes apartados describiremos en detalle la implementación de las dos entidades

3.3.1 Personaje principal

El personaje principal es el avatar del jugador en el juego, al que controla directamente mediante el teclado. Su comportamiento es bastante sencillo: el personaje se mueve cuando el jugador pulsa las flechas de dirección, y crea una “paleta” cuando se pulsa la tecla espacio. Usando la paleta, el personaje es capaz de redirigir la pelota,

¹<https://developer.nvidia.com/unity>

²<https://github.com/thefuntastic/Unity3d-Finite-State-Machine>

pero si la pelota impacta contra el directamente se quedará aturdido unos segundos. La paleta solo permanece activa unos instantes, en los cuales el personaje permanece inmóvil, por lo que presenta un desafío para el jugador, que debe saber posicionarse correctamente. El personaje reproduce también una animación al inicio de los niveles y otra al final, en la que entra y sale de la sala del juego.

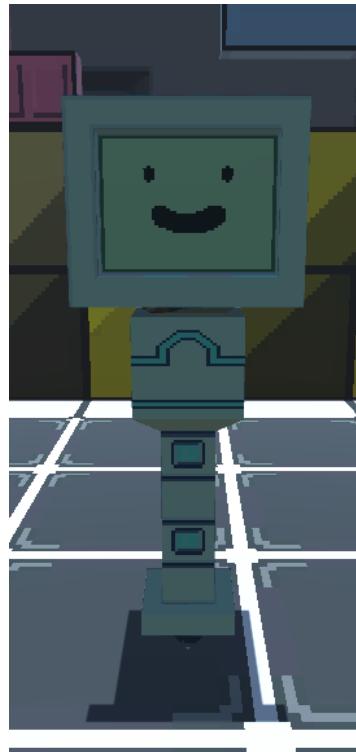


Figura 3.8: Modelo del personaje principal

Los estados que componen al personaje principal son los siguientes:

- **Enter:** Es el estado inicial. En este estado el personaje, posicionado fuera de escena, se mueve hacia el interior de la sala hasta colocarse en el centro de esta. Durante este estado, la colisión y la gravedad están desactivadas, lo que le permite entrar en la sala cerrada. Una vez terminada la animación, pasa al estado **Stand**.
- **Stand:** Se trata del estado principal. En él, el personaje espera inmóvil a recibir las órdenes del personaje. Si se pulsan las flechas de dirección, el personaje pasará al estado **Move** y si se pulsa la tecla espacio, se pasará al estado **Paddle**.
- **Move:** En este estado el personaje se mueve a velocidad constante en la dirección determinada por las flechas pulsadas. Cuando ninguna de las flechas de dirección esté siendo pulsadas, se volverá al estado **Stand**. En este estado también se puede pulsar la tecla espacio para moverse entrar en el estado **Paddle**.
- **Paddle:** En este estado el personaje crea frente a él un objeto Paleta que sirve para desviar la pelota. La paleta tiene un temporizador que hace que, al acabarse,

3. ARQUITECTURA

la paleta desaparezca y el personaje vuelve al estado **Stand**. Durante este estado el jugador no se puede mover, aunque si se entró desde el estado **Move** todavía se conservará parte de la velocidad debido a la inercia.

- **Confused:** El personaje entra en este estado cuando la pelota le golpea, independientemente de en qué estado se encuentre. En este estado, el jugador girará aturdido durante unos instantes antes de volver al estado **Stand**, como penalización para el jugador por no haber golpeado la pelota correctamente.
- **Dead:** El personaje entra en este estado cuando la pelota es destruida, sin importar el estado anterior. En este estado el personaje se desploma derrotado y se queda inmóvil, a espera de la pantalla de “fin del juego”.
- **Exit:** Cuando se completa un nivel, el personaje entra en este estado. El estado tiene dos partes: primero, el personaje permanece inmóvil a la espera de que la puerta del nivel se abra; y una vez esta esté abierta, el personaje avanza a través de ella hacia el siguiente nivel.

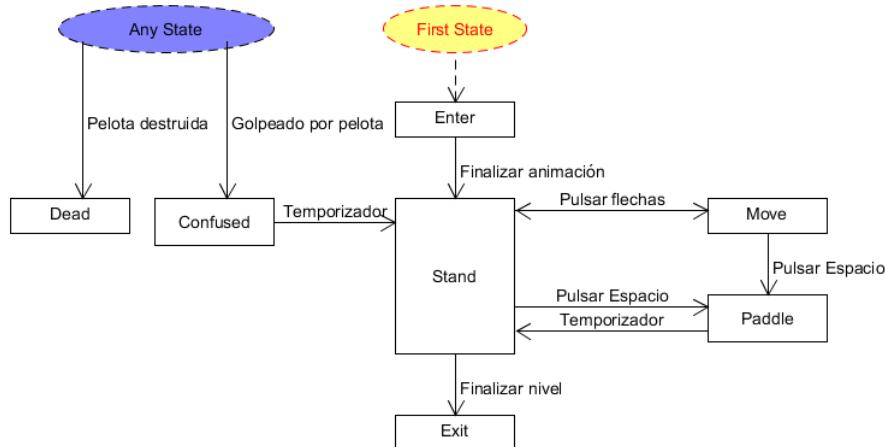


Figura 3.9: Diagrama de estados del personaje principal

Los movimientos del personaje (tanto durante el movimiento controlados por el jugador como durante las animaciones de principio y final del nivel) se implementaron mediante el uso de fuerzas del sistema de físicas. Esto hacía que el jugador se moviese de forma fluida sin cambios bruscos de velocidad. Para el movimiento controlado por el jugador, se utilizaron fuerzas muy grandes en conjunción con un factor de fricción también elevado, lo que permitía controlar al personaje con precisión. Esto se combina con una reducción de la fricción en el estado **Paddle** para que el personaje conservase parte de su velocidad al sacar la paleta, lo que servía para aumentar el margen de error del jugador a la hora de intentar golpear la pelota.

Las animaciones del personaje se realizaron mediante el motor de animaciones de Unity. Este motor permite animar los elementos del juego alterando sus propiedades

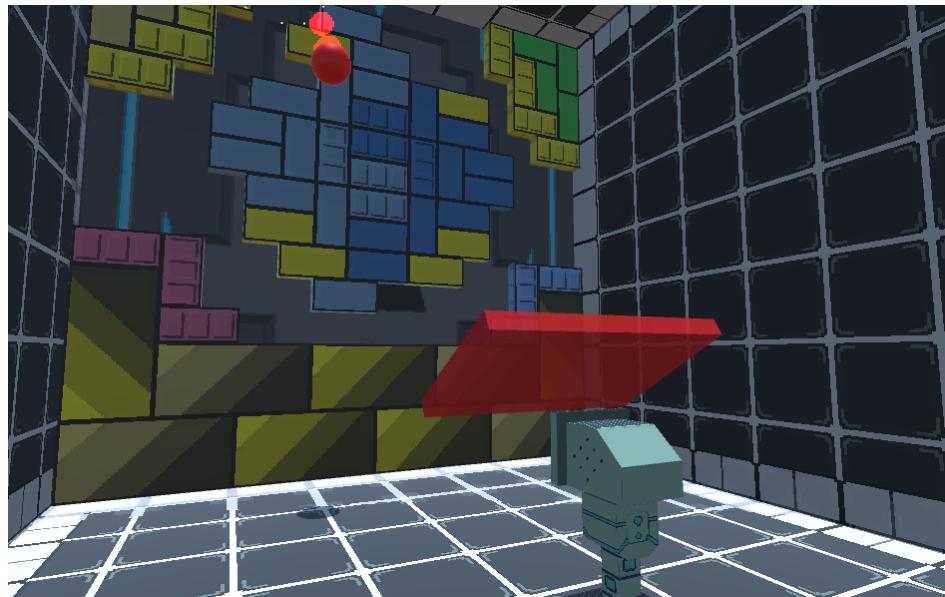


Figura 3.10: Personaje principal activando la paleta

(como la posición, rotación y escala de los modelos) durante intervalos de tiempo. El sistema funciona como una máquina de estados que permite cambiar entre distintos “clips” de animación dependiendo de variables de control de los objetos. Las animaciones que se crearon para el personaje son muy sencillas (cambios de escala y rotación) por lo que no hicieron uso de las propiedades más avanzadas del motor de animaciones (como los arboles de mezcla de animación³ o la cinemática inversa⁴)

3.3.2 Pelota

Podríamos considerar la pelota como el ”arma” del personaje principal, el medio por el que el jugador interacciona con los objetos del juego. La pelota viaja en trayectoria rectilínea, rebotando en las paredes y el techo de la sala del juego. Al golpear la puerta, o los bloques que la cubren, estos reciben daño y la pelota rebota, pero si la pelota golpea el suelo de la sala es la pelota la que recibe daño. Si la pelota golpea el suelo tres veces, se acaba la partida. El jugador debe intentar golpear la pelota con la paleta para redirigirla hacia la puerta, evitando que toque el suelo.

La máquina de estados de la pelota es significativamente más simple que la del personaje jugador, contando solo con tres estados: **Normal** (su comportamiento normal de movimiento), **Destroy** (animación de destrucción de la pelota que se reproduce al final del juego) y **Locked** (estado especial en el que la pelota permanece quieta e invisible, para la cinemática de inicio y final del nivel). Sin embargo, la tabla de interacciones de la pelota es mucho más extensa, debido a que tiene una reacción distinta para cada objeto del juego con el que colisione. Las posibles reacciones son:

³<https://docs.unity3d.com/Manual/class-BlendTree.html>

⁴<https://docs.unity3d.com/Manual/InverseKinematics.html>

3. ARQUITECTURA

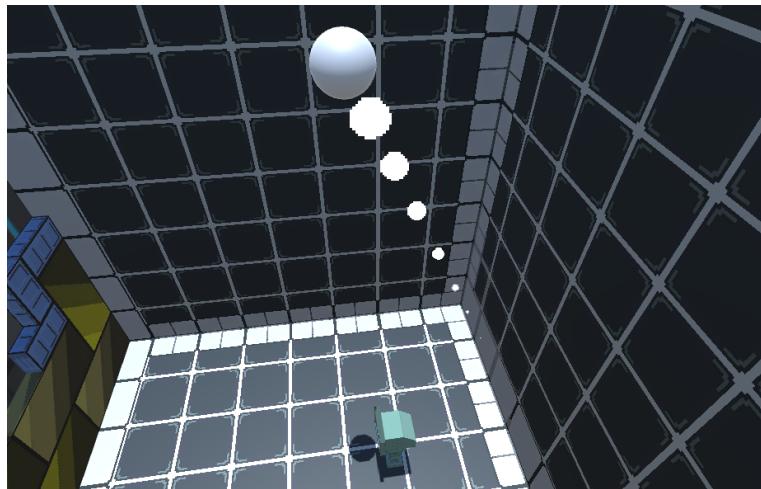


Figura 3.11: Pelota moviéndose

- Al golpear la **Puerta** de la sala los **Bloques** que la cubren, estos sufrirán un punto de daño, y la pelota rebotará. Al iniciarse el rebote, la pelota empezará a caer en dirección al suelo de la sala.
- Al golpear las **Paredes** o el **Techo** de la sala, la pelota rebotará de forma natural.
- Al golpear el **Suelo** de la sala, la pelota recibirá un punto de daño. Además, la pelota rebotará, pero no de forma natural sino de forma que vaya en dirección a la puerta. Esto sirve principalmente para facilitar la tarea del jugador, “regalándole” un golpe a la puerta. La pelota también pierde la fuerza de la gravedad.
- Al golpear al **jugador**, este se quedará aturdido unos instantes y la pelota rebotará de forma natural.
- Al golpear la paleta del jugador, la pelota rebotará y perderá la fuerza de la gravedad. La dirección en la que rebotará la pelota dependerá del punto de la paleta en el que golpeó la pelota, siendo la dirección normal al plano de la paleta si se golpea justo en el centro.

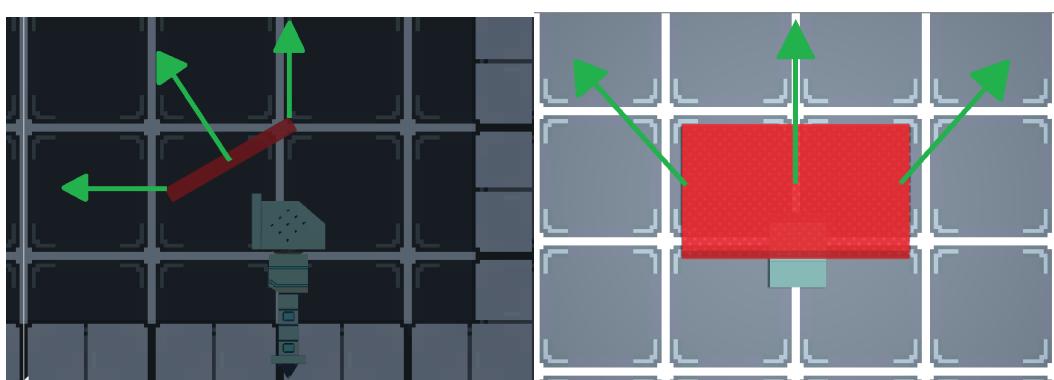


Figura 3.12: Direcciones que tomaría la pelota dependiendo del punto de la paleta que golpee (Aproximada)

El movimiento de la pelota se basa íntegramente el motor de físicas de Unity. Al entrar en su estado **Normal**, la pelota recibe un impulso en una dirección dada y a partir de ahí se moverá sin fricción de forma perpetua. Los movimientos “antinaturales” mencionados, el cambio de dirección al golpear el suelo o la paleta se realizan cambiando el vector de la velocidad de la pelota por un vector nuevo generado manualmente por código.

Aunque la pelota no tiene animación en si misma, a su comportamiento se le han añadido unos efectos de partículas creados mediante el motor de partículas de Unity⁵, que sirven tanto para embellecer el juego como para crear guías visuales para el jugador. La pelota cuenta con tres efectos animados de partículas: una “cola” que marca la trayectoria que siguió la bola; un efecto circular que aparece cuando la bola colisiona con las paredes, el techo y el suelo, que sirven para marcar el punto de impacto de la bola y un efecto de explosión final.

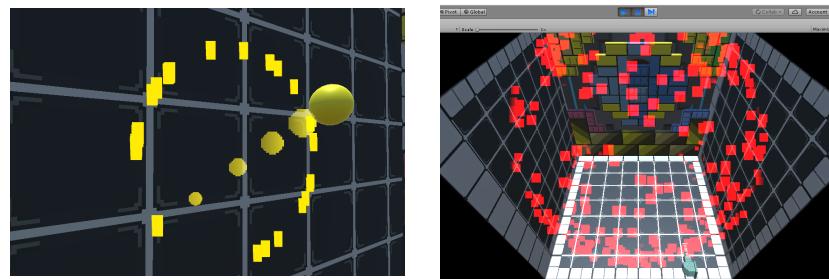


Figura 3.13: Efectos de partículas: impacto en muro (izquierda) y explosión (derecha)

3.4 Jefe Final

Un jefe final, o monstruo jefe, es un enemigo poderoso al que los jugadores deben enfrentarse para poder alcanzar algún objetivo dentro del juego [Bjo]. Los jefes sirven principalmente para tres propósitos: dar clausura a una sección del juego, al colocarse al final de una sección, nivel o fase; ofrecen variedad al juego, al suponer una variación con respecto al juego corriente y finalmente sirven como “test” para el jugador, al tratarse de un desafío mucho mayor que los anteriores.

Para este juego, el jefe final es el único “enemigo” al que se enfrenta el jugador, que aparece en el nivel 11 del juego. Este jefe se comporta como un “clon” o “rival” del jugador: su objetivo es impedir que la pelota golpee la puerta, moviéndose y redirigiéndola de forma similar a como lo haría el jugador. El jefe se mueve a velocidad constante, siempre pegado a la pared, en dirección a la pelota cuando detecta que esta se mueve hacia la pared. Para evitar que este enemigo sea imbatible, su velocidad se reduce cada vez que consigue la pelota, lo que provoca que se vuelva incapaz de seguirle el ritmo a esta después de un tiempo.

⁵<https://docs.unity3d.com/Manual/ParticleSystems.html>

3. ARQUITECTURA

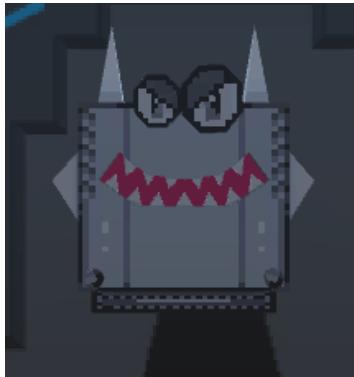


Figura 3.14: Modelo del Jefe.

La forma de derrotar a este jefe es idéntica a como se supera cualquier fase: Golpeando la puerta suficientes veces.

3.4.1 Implementación

La implementación de este jefe se realizó usando el sistema de Prefabs y Componentes de Unity. El jefe es un Prefab al que el sistema de carga de niveles instancia en los niveles marcados como niveles de jefe. Este Prefab está formado por dos GameObjects: el primero, que se encarga de determinar el comportamiento del jefe y un GameObject hijo que contiene el modelo del jefe. Se implementa de esta para permitir cambios en el modelo del jefe sin que se tenga que ajustar el comportamiento y también para facilitar la animación. El GameObject principal, por otro lado, incluye una serie de componentes que se encargan de modelar distintas partes su comportamiento. La mayor parte de estos son componentes por defecto de Unity:

- **Transform:** Almacena la posición, rotación y escala del jefe, información básica de todo objeto del juego.
- **Collider:** Se utiliza en la detección de colisiones.
- **Rigidbody:** Otorga propiedades físicas al GameObject, como velocidad, masa o aceleración. Simplifica enormemente la implementación del movimiento del jefe.
- **Animator:** Permite añadir animaciones al GameObject, las cuales alteran las propiedades de sus componentes basándose en el tiempo. Se utiliza para crear dotar al jefe de animaciones sencillas de modo que no resulte demasiado estático.
- **Audio Source:** Sirve para emitir sonidos y música. El jefe lo utiliza para emitir sonidos en situaciones concretas, como al golpear la pelota o al ser derrotado.

Para controlar el comportamiento del jefe se implementaron también dos scripts: **BossController** y **BossAI**. **BossController** Contiene la información y las funciones necesarias para facilitar el movimiento del jefe. El método principal de esta clase, *MoveDirection()*, permite mover al jefe a velocidad constante en el ángulo suministrado,

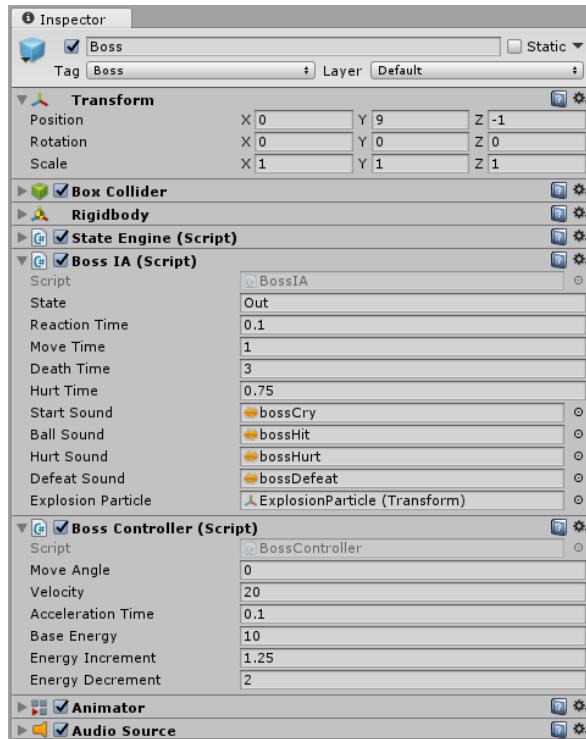


Figura 3.15: Inspector mostrando los componentes del Jefe.

siempre en relación con el plano de la puerta. El movimiento se implementa a través del Componente Rigidbody, aplicando fuerza constante en la dirección indicada, lo que crea un movimiento con una ligera aceleración que resulta más natural que un cambio instantáneo de velocidad. El script implementa también un sistema de energía: cuanto mayor sea la energía, más rápido se mueve.

Este método es utilizado por el segundo script, **BossAI**, para mover al jefe. El script **BossAI** codifica el comportamiento del jefe en forma de una máquina de estados finitos que decide la forma en la que el jefe se mueve dependiendo de la dirección de la bola. También se encarga de aumentar y reducir la energía del jefe y de cambiar la animación. El comportamiento se compone de los siguientes estados:

- **Out:** El jefe espera fuera del área de juego, inmóvil, a la espera de la entrada del jugador en la sala.
- **Intro:** El jefe se mueve a su posición inicial y realiza su animación inicial. Este movimiento se implementa mediante el uso de animaciones en lugar de mediante el BossController dado que debe atravesar las paredes de la sala.
- **Wait:** En este estado, el jefe permanece inmóvil a la espera de un estímulo que le haga cambiar iniciar el movimiento, en concreto, que se detecte que la pelota se aproxima a la puerta.
- **Move:** El jefe se mueve en dirección a la pelota utilizando los métodos de BossController. El movimiento se detendrá si se consigue golpear la pelota o si la

3. ARQUITECTURA

pelota golpea la puerta.

- **Hurt:** En este estado el jefe permanece inmóvil mientras se reproduce una animación de daño. Tras esta breve pausa, el jefe retorna al estado Wait.
- **Death:** Cuando la puerta se abre, el jefe realiza una animación de destrucción, en la que el jefe se vuelve invisible y emite una explosión de partículas.

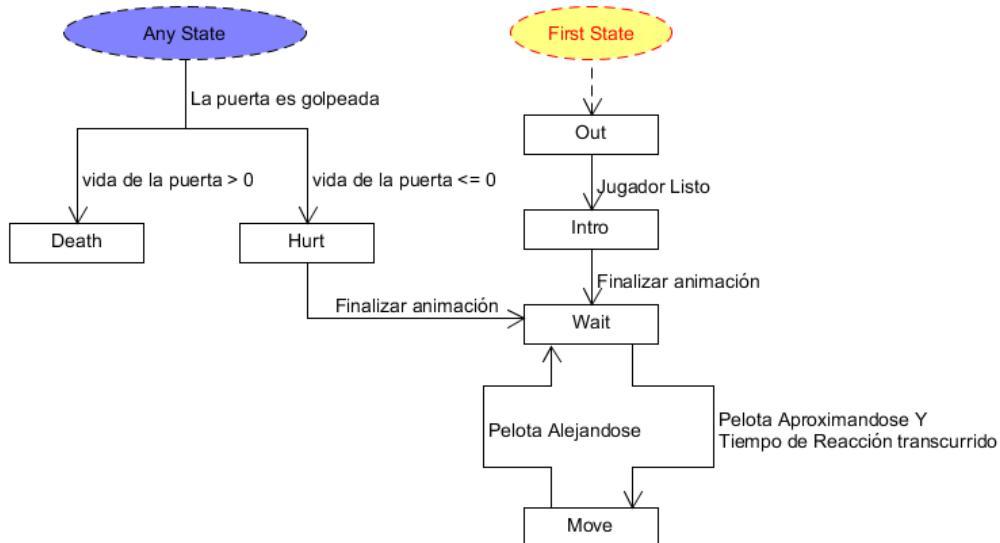


Figura 3.16: Diagrama de estados del jefe.

Cuando la pelota golpea al jefe, este cambia la dirección de la pelota. Esta redirección no sigue el ángulo “natural” que seguiría la pelota si “rebotase” contra el jefe, sino que es redirigido hacia el jugador. El propósito de este comportamiento es crear una situación en la que el jefe y el jugador se intercambian la pelota como si de un partido de tenis se tratase. Cada vez que la pelota golpea al jefe, BossIA reduce la energía de BossController.

Por otro lado, si el jefe falla al golpear la pelota, y esta golpea la puerta, BossIA restaurará la energía y aumentará por encima del máximo inicial, incrementando la velocidad. Esto provoca un incremento progresivo de la dificultad a medida que avanza el combate.

El objetivo de este diseño en dos componentes del comportamiento del jefe nos permite mantener una separación entre la implementación del movimiento y el comportamiento propiamente dicho. De esta forma, podremos remplazar, por ejemplo, la inteligencia artificial por un controlador manual para implementar un modo de dos jugadores, o usar el módulo de la inteligencia artificial en otro jefe distinto cuyo comportamiento se base también en la posición de la bola.

Capítulo 4

Conclusiones

4.1 Resultados



Figura 4.1: Relación entre desafío y la ansiedad/aburrimiento.

El desarrollo del juego tuvo éxito en implementar las características básicas del juego. Algunas de estas características son:

- El control de personaje principal fue desarrollado completamente, de forma que es preciso y fácil de controlar. Se pudo implementar unas pequeñas cinemáticas para su entrada y salida de la sala.
- El movimiento de la pelota y su interacción con el resto de los elementos también fue implementado con éxito, incluyendo las modificaciones a las trayectorias de rebote naturales que mejoraban la experiencia de juego.
- La funcionalidad deseada para el sistema de carga de niveles pudo ser implementada a la perfección, con posibilidad de fácil ampliación en caso de necesidad.

4. CONCLUSIONES

- Relacionado con los niveles, se pudieron implementar los cinco tipos de bloques descritos en el diseño.
- El jefe final pudo ser implementado.
- Las pantallas de título y fin del juego pudieron ser implementadas y decoradas con textos animados. En adición a estas dos primeras pantallas, se creó una pantalla de victoria.



Figura 4.2: Relación entre desafío y la ansiedad/aburrimiento.

El estilo artístico del juego pudo ser implementado completamente. Todos los elementos del juego cuentan con su modelo y textura propios, gracias a la simpleza del estilo gráfico. También se añadieron pequeñas animaciones simples al personaje principal, a la puerta, al jefe y a los textos de los menús. Las partículas que se utilizan para enfatizar determinadas acciones, como el movimiento de la pelota o la destrucción de bloques, se implementaron utilizando texturas sólidas de baja resolución, lo que conjuga con el resto del estilo.

El juego se encuentra alojado tanto en GameJolt(<https://gamejolt.com/games/virus-breaker/316764>) como en Itch.io(<https://pedro-romero.itch.io/virus-breaker>). En ambas páginas, el juego puede ser descargado gratuitamente para la plataforma Windows.

4.2 Mejoras Futuras

Debido a las limitaciones de tiempo y alcance de este proyecto, el juego resultante está lejos de los estándares de calidad de la industria actual. En caso de querer retomar este proyecto por un estudio profesional, el juego debería ser modificado para añadir

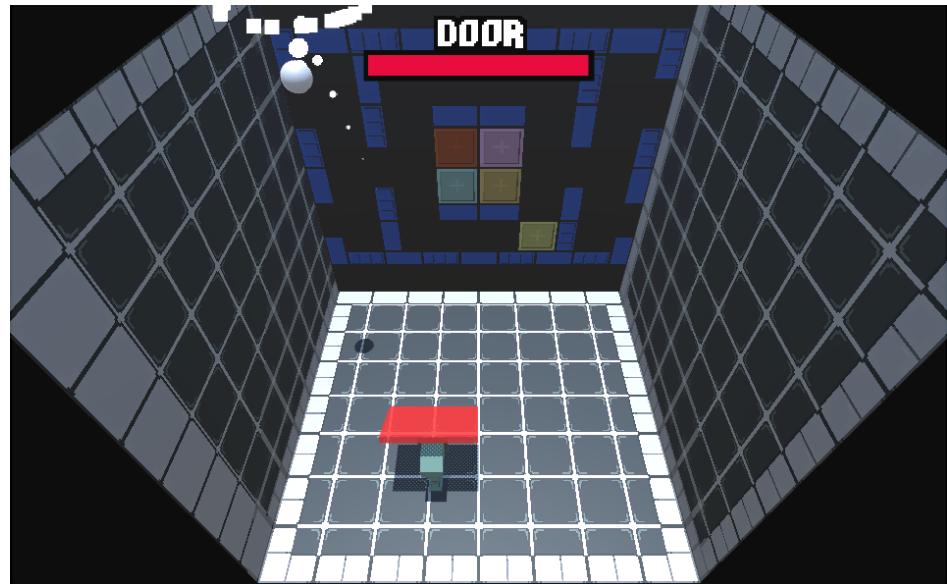


Figura 4.3: Relación entre desafío y la ansiedad/aburrimiento.

gráficos y sonidos de mejor calidad, aumentar su duración con más niveles y limpiar y optimizar su código.

A continuación, hablaremos de algunos de los elementos que podrían ser añadidos en versiones futuras del juego. Muchos de ellos son elementos que se idearon originalmente para formar parte de esta versión del proyecto, pero fueron descartados por falta de tiempo y recursos.

4.2.1 Escenarios

Lo primero que se añadiría al juego serían más niveles de juego. Al aumentar el número de niveles supondrá un incremento de la duración del juego, lo que resultará más atractivo para el jugador. Dado que los juegos de este género pueden superar el centenar de niveles (por ejemplo, LEGO Bricktopia¹ cuenta con 150 niveles) por lo que sería necesario implementar un número mayor de niveles para poder competir con ellos.

La cantidad de niveles que se pueden realizar no es infinita: solo alternando la configuración de los bloques que existe actualmente se llegará pronto a un punto en el que no se podrán crear más niveles sin caer en la repetición. La solución a este problema sería la implementación de más elementos de juego que dotaran de variedad a los niveles.

La creación de más tipos de bloques sería una de las primeras soluciones, ya que podrían integrarse con relativa facilidad al sistema actual de codificación de niveles. Entre los posibles bloques estarían los bloques explosivos (destruyen bloques cercanos al ser destruidos), bloques móviles que pudiesen programarse para crear formaciones

¹<https://www.bigfishgames.com/games/1252/legobricktopia/>

4. CONCLUSIONES

móviles, o bloques con propiedades físicas distintas que alteraran el movimiento de la pelota (aumentando su velocidad o haciendo que rebote en ángulos extraños).

Otro posible elemento por implementar serían los niveles móviles. En estos la cámara se movería por una sala mucho más grande que la sala convencional. En estos niveles el objetivo del jugador sería más luchar contra el movimiento de la cámara para evitar quedar fuera del encuadre que el de abrir la puerta. Los principales problemas para implementar estos niveles serian:

1. Se debería modificar la forma en la que los niveles se guardan para dar soporte a la nueva información necesaria para estos niveles (principalmente, tamaño y velocidad de movimiento).
2. Habría que cambiar la estructura de la sala base para dar soporte al tamaño variable.
3. Convendría implementar un sistema que gestionase la creación de bloques, el cual fuese creando los bloques según aparecen en el encuadre, y los elimine según se salgan de este, para optimizar el juego.

4.2.2 Enemigos

Los enemigos son elementos del juego que dificultan activamente el progreso al jugador, proveyendo al jugador de desafíos que superar[Bjo]. Los enemigos son comunes en este género de juegos (apareciendo incluso en títulos pioneros como Arkanoid (1986, Taito)) y permiten dar variedad e interactividad a los niveles del juego. El juego final incluye un único enemigo, el jefe final, aunque durante el desarrollo se barajó la opción de crear algunos enemigos, pero fue descartada para reducir el coste de su programación.

Los dos enemigos diseñados eran los siguientes:

- **El Trepador:** Se trataba de un enemigo con forma de gusano que reptaba a velocidad constante por la sala, pudiendo incluso trepar por el techo y paredes. Podía aturdir al personaje principal si este lo tocaba, por lo que el jugador debía esquivarlos mientras se movían por la sala.
- **El Replicador:** Este enemigo se colocaría en la puerta e imitaría el aspecto de los bloques. Cada cierto tiempo, este enemigo crearía una copia de si mismo en un espacio libre cercano, por lo que el jugador debería darse prisa o acabarían reparando"la pared. Estos enemigos podían ser destruidos si se les golpeaba con la pelota.

También sería necesario la adicción de jefes finales, para dar variedad a los desafíos finales. Durante el desarrollo se barajaron varios jefes finales distintos, pero fueron descartados en favor del jefe actual dado que solo se tenía tiempo para implementar

uno. Estos diseños podrían recuperarse, se tendría que tener en cuenta el coste de implementar un jefe, cuyo comportamiento es bastante más complejo.

4.2.3 Sistemas Complementarios

Fuera del diseño de niveles, el juego necesita de ciertos sistemas complementarios que sirvan para prolongar su vida útil.

El primero de estos sistemas sería una tabla de puntuaciones. Las tablas de puntuaciones son unas listas que almacenan la puntuación que han alcanzado los jugadores al terminar el juego con el propósito de que puedan ser comparadas. La implementación de estas tablas, junto con la de un sistema de puntuación (que podría estar basado en factores como el tiempo de juego, cantidad de bloques destruidos o número de niveles superados) aumentaría enormemente la vida útil del juego ya que alentaría al jugador a que volviera a jugar con el propósito de superar su propia puntuación o la de otro jugador. Idealmente, la tabla de puntuaciones sería online, compartida entre todos los jugadores, pero eso supondría un coste adicional debido a la infraestructura necesaria.

El segundo sistema que podría implementarse para aumentar la vida útil del juego es un sistema de generación procedimental de niveles. Este sistema podría producir una cantidad de niveles infinita, lo que solucionaría el problema de la corta duración y reduciría el tiempo de desarrollo dedicado a la generación manual de niveles. Además, basándose en este sistema podrían implementarse otros sistemas complementarios, como un sistema de “niveles diarios” en la que todos los jugadores jugarían a un mismo grupo de niveles (sistema presente en juegos como *The binding of Issac: Rebirth*(Nicalis, 2014) o *Leap Day*(Nitrome, 2016).

ANEXOS

Bibliografía

- [Bar] Sean Baron. *Cognitive Flow: The Psychology of Great Game Design*. URL: https://www.gamasutra.com/view/feature/166972/cognitive_flow_the_psychology.php.
- [Bat04] Bob Bates. *Game Design*. Boston: Thomson Course Technology, 2004.
- [Bet03] Erik Bethke. *Game Development and Production*. Plano, Texas: Wordware Publishing, 2003.
- [Bjo] Staffan Bjork. *Game Design Patterns*. URL: http://virt10.itu.chalmers.se/index.php/Main_Page.
- [BS12] Brenda Brathwaite y Ian Schreiber. *Breaking into the Game Industry: Advice for a Successful Career from Those Who Have Done It*. Boston: Course Technology, 2012.
- [Dav15] David Villa David Vallejo Carlos González. *Desarrollo de Videojuegos: Un Enfoque Práctico*. Ciudad Real, España, 2015.
- [DEV17] Desarrollo Español del Videojuego DEV, ed. *Libro Blanco del Desarrollo Español del Videojuego*. Madrid, España, 2017.
- [Gre09] Jason Gregory. *Game Engine Architecture*. Wellesley, Massachusetts: A K Peters, Ltd., 2009.
- [Lyd] Bill Lydon. *Industry 4.0 - Only One-Tenth of Germany's High-Tech Strategy*. URL: <https://www.automation.com/automation-news/article/industry-40-only-one-tenth-of-germanys-high-tech-strategy>.
- [Nys04] Robert Nystrom. *Game Programming Patterns*. Geneber Benning, 2004.
- [Sam59] Arthur L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. En: *IBM Journal of Research and Development* (1959), págs. 210-229.
- [Sid14] James D. McCalley Siddhartha Kumar Khaitan. “Design Techniques and Applications of Cyber Physical Systems: A Survey”. En: *IEEE Systems Journal* 9.2 (2014), págs. 350-365. DOI: <https://doi.org/10.1109/JST.2014.2322503>.
- [Tec] Unity Technologies, ed. *Unity Manual*. URL: <https://docs.unity3d.com/Manual/UnityManual.html>.

BIBLIOGRAFÍA

- [Tur53] Alan M. Turing. “Digital computers applied to games”. En: *Faster than thought* (1953), pág. 101.
- [War] Jeff Ward. *What is a Game Engine?* URL: https://www.gamecareerguide.com/features/529/what_is_a_game_.php.
- [Yan12] Georgios N. Yannakakis. “Game AI Revisited”. En: *Proceedings of the 9th conference on Computing Frontiers* (2012), págs. 285-292.
- [YT18] Georgios N. Yannakakis y Julian Togelius. *Artificial Intelligence and Games*. <http://gameaibook.org>. Springer, 2018.

Videojuegos

- [Cap91] Capcom. *Street Fighter II: The World Warrior*. 6 de feb. de 1991.
- [Chu86] Chunsoft. *Dragon Quest*. 27 de mayo de 1986.
- [Dou52] Alexander S. Douglas. *OXO*. 1952.
- [Eni16] Square Enix. *Final Fantasy XV*. 29 de nov. de 2016.
- [Ent04] Blizzard Entertainment. *World of Warcraft*. 23 de nov. de 2004. URL: <https://worldofwarcraft.com>.
- [Gam09] Riot Games. *League of Legends*. 27 de oct. de 2009. URL: <https://leagueoflegends.com>.
- [Max00] Maxis. *The Sims*. 4 de feb. de 2000.
- [Meg98] Epic MegaGames. *Unreal*. 22 de mayo de 1998.
- [Moj11] Mojang. *Minecraft*. 18 de nov. de 2011.
- [Nin81] Nintendo. *Donkey Kong*. 9 de jul. de 1981.
- [Nin85] Nintendo. *Super Mario Bros*. 13 de sep. de 1985.
- [Nin86] Nintendo. *The Legend of Zelda*. 21 de feb. de 1986.
- [Sof93] id Software. *Doom*. 10 de dic. de 1993.
- [Sof94] Raven Software. *Heretic*. 23 de dic. de 1994.
- [Sof96] id Software. *Quake*. 22 de jun. de 1996.
- [Sof99] id Software. *Quake III Arena*. 2 de dic. de 1999.
- [Squ95] Square. *Chrono Trigger*. 11 de mar. de 1995.
- [Val07] Valve. *Team Fortress 2*. 10 de oct. de 2007. URL: <https://teamfortress.com>.

VIDEOJUEGOS

Este documento fue editado y tipografiado con L^AT_EX empleando la clase **esi-tfg** (versión 0.20180319) que se puede encontrar en:
https://bitbucket.org/arco_group/esi-tfg

[respeta esta atribución al autor]

