UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

CREACIÓN Y DESARROLLO DE UN VIDEOJUEGO BASADO EN UN SISTEMA DE MAGIA SINTÁCTICO

Trabajo Final de Grado

FACULTAD DE INFORMÁTICA DE BARCELONA (FIB)

Especialización en Ingeniería del Software

Autor: Germán Alejandre Campos

Director: Manuel Rello Saltor

Ponente: Ernest Teniente López

21 de junio de 2021

RESUMEN

Para innovar en las posibilidades de los conjuros del jugador de los videojuegos, este proyecto propone crear un sistema de magia basado en la estructura de un nuevo lenguaje. En él, los hechizos hacen el papel de oración y, como tal, están formados por la unión de diversos sintagmas (que determinan su comportamiento) regidos por una gramática que les aporta significado.

Para poder probar este modelo en acción, se ha programado también un juego prototipo que lo implementa. Esto implica plantear ciertos aspectos, como por ejemplo los apartados de visualización y de audio, que tendrán que configurarse en tiempo real.

RESUM

Per innovar en les possibilitats dels conjurs del jugador en els videojocs, aquest projecte proposa crear un sistema de màgia basat en l'estructura d'un nou llenguatge. En ell, els encantaments fan el paper d'oració i, com a tal, estan formats per la unió de diversos sintagmes (que determinen el seu comportament) regits per una gramàtica que els hi aporta significat.

Per poder provar aquest model en acció, s'ha programat també un joc prototip que l'implementa. Això implica plantejar alguns aspectes, com per exemple els apartats de visualització y d'àudio, que s'hauran de configurar en temps real.

ABSTRACT

In order to be innovative in the possibilities of sorcery for the videogame's player, this project proposes the creation of a magic system based on the structure of a new language. As such, spells play the role of phrase and are formed by the union of various syntagms (which determine their behaviour) while being governed by a grammar that gives them meaning.

As a means to test this model in action, a prototype game based around the model has also been programmed. This implies raising certain aspects, such as the visualization and audio sections, which will have to be configured in real time.

AGRADECIMIENTOS

En primer lugar, me gustaría retribuir tanto al profesor Manuel como al profesor Ernest. Ambos tutores han contribuido a la producción de este proyecto y han proporcionado vastas cantidades de *feedback* durante la producción del mismo. Gracias a la realización de este trabajo he podido conocer a dos grandes maestros de la ingeniería del Software que me han ayudado a consolidar los conocimientos aprendidos durante estos cuatro años de carrera.

Por otro lado, agradezco también a mis padres y a mi hermana por apoyarme y escucharme durante la evolución del proyecto. Su consejo e interés ha sido muy importante en la realización de este documento.

En última instancia, pero no por ello menos relevante, me complacería dar las gracias a todos mis amigos por estar a mi lado y ayudarme a concebir los requisitos de este trabajo. Las largas noches “virtuales” de estos últimos meses de pandemia no hubieran sido igual sin vosotros.

ÍNDICE

CAPÍTULO 1: INTRODUCCIÓN

El trabajo de fin de grado “Creación y Desarrollo de un Videojuego basado en un sistema de magia sintáctico” consiste en un proyecto perteneciente a los estudios de Grado en Ingeniería Informática de la Facultat d’Informàtica de Barcelona (Universitat Politècnica de Catalunya), concretamente en la especialidad de Ingeniería del Software.

1.1. CONTEXTO

En el presente, nadie puede negar que el mundo de los videojuegos es uno de los mercados que más rápido se está expandiendo dentro del sector multimedia. Estos productos ya superan en ingresos a otras industrias como la del cine, y ha sido uno de los pocos negocios que ha crecido durante este año 2020 con la crisis del coronavirus. Según [1], los videojuegos han conseguido facturar la friolera de 175 mil millones de dólares si se combinan las ventas de software y de hardware (consolas), que supone un incremento del 20% respecto al año anterior.

No obstante, esto no ha sido siempre así. El fenómeno no empezó hasta finales de la década de los 70, momento en el que exponentes como Pong o la primera NES salen a la venta. En aquellos años, el mercado de los videojuegos era inexistente y es desde entonces que los mismos se van poco a poco introduciendo en la vida cotidiana de las personas. Esto se debe de forma fundamental a la reducción del coste de venta del hardware necesario, el fenómeno de Internet y su facilidad de uso.

Para ejemplificarlo, uno puede viajar veinte años atrás en el tiempo. A finales de la década de los noventa y a principios de siglo, cualquier niño (que podría ser perfectamente yo) al cual le gustara jugar a videojuegos era considerado por sus compañeros como el friki de la clase en la mayoría de casos. Es muy fácil generalizar, aunque la situación era así en la mayoría de casos. Sin embargo, por fortuna la situación actual es radicalmente distinta a la que nos podíamos encontrar en las escuelas españolas de antaño, pudiendo perfectamente disfrutarlos abiertamente con otros compañeros y ser igualmente reconocido como un niño más de la clase. A ello han influido también factores externos, entre los cuales destacan la aparición de las redes sociales. Con ellas, existen también nuevas celebridades que viven del contenido que crean a partir de estos programas.

Gracias a los avances en el desarrollo de software los videojuegos también han evolucionado. Desde los primeros de plataformas hasta los últimos *shooters* multijugador o de estrategia, nuevos géneros han ido surgiendo con el paso de los años. Con ello, muchos títulos se han ido desarrollando a lo largo de la historia. Algunos logran popularizarse dentro de la comunidad (otros no tanto), logrando así crear nuevas tendencias que pueden perdurar durante varios años.

Paralelo al software, el hardware también ha sido fundamental a la hora de popularizar esta nueva forma de ocio. Aunque el fenómeno empezó con las máquinas arcade de los ya clásicos locales de recreativas de los años ochenta, las consolas y el PC se encargaron en su día de montar una base estable de consumidores. Hoy por hoy, los dispositivos móviles han ampliado masivamente el número de jugadores gracias la invención y accesibilidad del smartphone, tal y como podemos ver en la Figura 1.

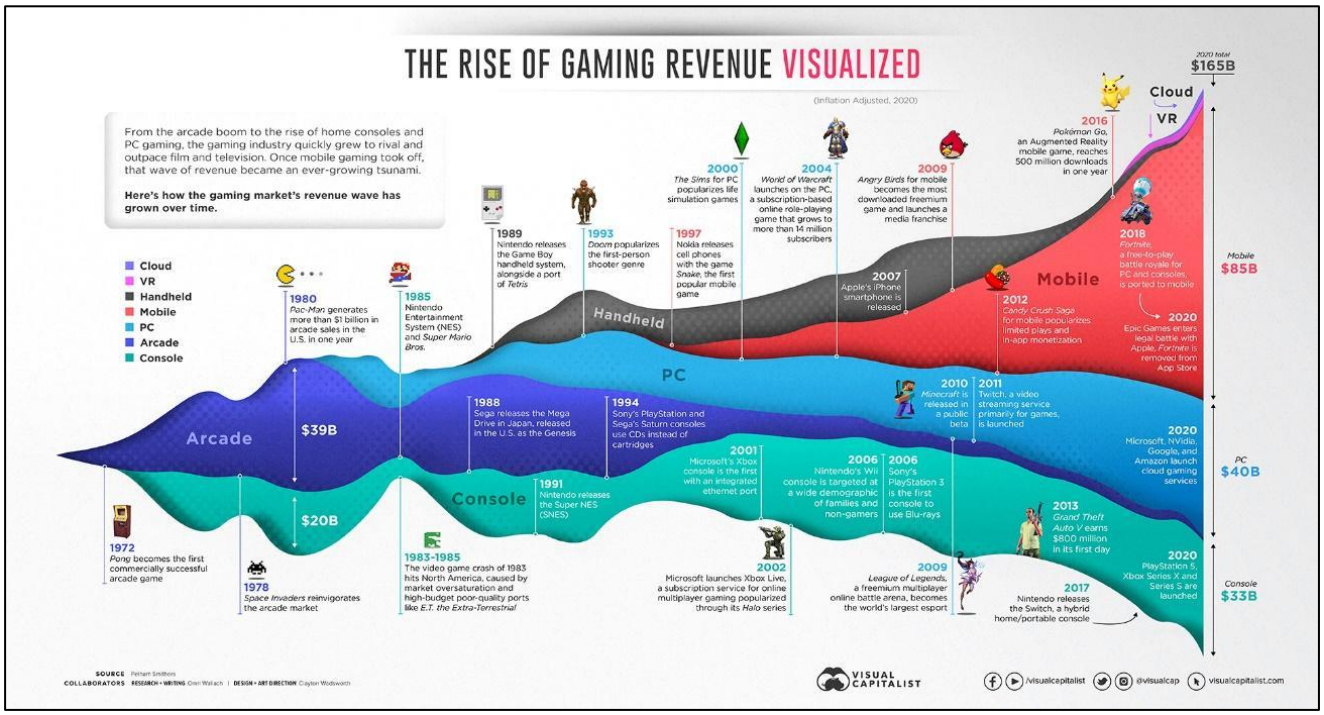


Figura 1: Histórico de las ganancias del mercado de los videojuegos por plataforma. Fuente: [2]

Es importante hacer especial hincapié en que los videojuegos no dejan de ser programas diseñados con la finalidad de entretener e inducir una sensación de satisfacción al usuario. Como tal, los mismos han experimentado todos los cambios y revoluciones que se han producido en la creación de programas, entre los cuales quiero destacar la Ingeniería del Software.

Un indicador de estos cambios es el tamaño en disco que ocupan. Por ejemplo, el primer Super Mario estaba programado a mano en lenguaje de ensamblador que soportaba la consola, una tarea loable pero relativamente asequible teniendo en consideración que el juego solamente pesaba 31 KB. Casi cuarenta años más tarde es muy común encontrarse juegos "triple A" o de gran presupuesto que pueden llegar a ocupar casi una décima parte de la unidad de almacenamiento.

A lo que se quiere llegar con esto es que la escala del videojuego ha aumentado en todos sus aspectos y tanto la cantidad de líneas de código como su complejidad no es ninguna excepción. La mejora del hardware ha tenido mucho que ver, pero también debe destacarse la importancia de procesos como la ingeniería de requisitos o la arquitectura del software.

1.2. MOTIVACIÓN

En primer lugar, me gustaría destacar el curioso origen de la idea de este trabajo de final de grado. De forma inicial, en otoño mostraba bastante indiferencia respecto a la temática del proyecto siempre y cuando pudiera realizar un TFG de modalidad B en cualquier empresa de software. No obstante, después de enviar varios currículums a diversas empresas y otros meses de descontrol de la pandemia finalmente no pudo ser así.

En consecuencia, tuve que buscar una alternativa al plan que tenía pensado originalmente. Puestos a elegir, preferí escoger una temática que me atrajera desde el principio. Fue en este momento cuando pensé en realizar un trabajo relacionado con los videojuegos. Es un ámbito que me gusta mucho y considero muy interesante puesto que ha estado presente una buena parte de mi vida y en el cual, para bien o para mal, no se hace demasiado hincapié en estos durante la carrera. Sin embargo, tampoco quería hacer un videojuego como tal ya que hay ciertos aspectos (diseño artístico, modelaje, creación de texturas, etc.) que no son mi fuerte y tampoco están relacionadas con el grado. En este sentido, mi idea original era poder crear alguna mecánica o elemento de *gameplay* (que preferiblemente fuera innovador) capaz de integrarse en un proyecto real y que se pudiera posteriormente enseñar en una pequeña demo jugable. Al fin y al cabo, los videojuegos entran por los ojos y su éxito depende enormemente de ello.

Con esto en mente y después de haberle dado varias vueltas, finalmente decidí contactar con el profesor Manuel Rello, mi actual director del trabajo de fin de grado. Estuvimos discutiendo durante un par de días sobre varias posibilidades de *gameplay* poco exploradas dentro del mundo de los videojuegos. Entre todas ellas, finalmente surgió la idea que ha inspirado todo el trabajo realizado durante estos últimos meses: el diseño de una gramática como sistema de creación de hechizos integrada en un videojuego.

De entrada, he de admitir que la idea nos gustó tanto al profesor Manuel como a mí. Y es que cumple todos los prerrequisitos que había fijado en el inicio: se puede aplicar a cualquier juego de corte fantástico, es innovadora puesto que pocos videojuegos han tratado de explorar algo similar y es fácilmente contextualizarlo dentro del departamento de ESSI, ya que presenta un problema capaz de resolverse mediante el planteamiento de una arquitectura y el uso de patrones de diseño. Además, también presentaba la oportunidad de aprender conceptos de intérpretes y compiladores, dos áreas totalmente desconocidas para mí.

1.3. FORMULACIÓN DEL PROBLEMA

Como ya se ha mencionado en la introducción, en ella se comentan los diversos géneros y tendencias que han surgido durante toda la historia de los videojuegos. Entre todos ellos, el género RPG es una de las categorías más exitosas de la colección. Así lo justifican títulos como Diablo III (30M de unidades vendidas) [3], The Elder Scrolls V: Skyrim (30M de copias) [4] y The Witcher 3 (28M de ventas) [5]. Al ser títulos del mismo género, estos tres ejemplos comparten una serie de características comunes entre las cuales destaca el sistema de magia y los conjuros que los componen.

Antes de continuar, se considerará hechizo (o conjuro) toda acción con carácter mágico o sobrenatural que realice una entidad. En general se crea con el deseo de producir algún tipo de beneficio o perjuicio sobre un objetivo. Este último puede ser un NPC (personaje no jugable) o el propio jugador, aunque también pueden enfocarse en el entorno. En múltiples sagas, la magia suele ser un elemento más del trasfondo del universo y acostumbra tener características propias para favorecer la inmersión del usuario.

En concreto, existen dos tipos de sistemas de magia presentes en los videojuegos: un primero basado en hechizos ya predefinidos (que suele ser el más común, presente en conocidas sagas como “Final Fantasy” o “World of Warcraft”) y un segundo basado en conjuros configurables con opciones preestablecidas (véase la serie Elder Scrolls). Estos sistemas han funcionado muy bien a lo largo de estos años, pero su fórmula no ha presentado ningún avance considerable durante las últimas dos décadas.

Por ello, el proyecto quiere presentar una evolución algo más innovadora. A fin de aumentar la variedad de hechizos, este TFG propone profundizar y dar al usuario las herramientas, llamadas sintagmas, necesarios para crearlo. Este nombre no se ha escogido a la ligera, ya que para que el conjuro tenga éxito y funcione, el jugador tendrá que combinar elementos sintácticos de manera que el resultado final tenga sentido. Dicho de otra forma, el proyecto se propone crear un lenguaje formal similar a cualquier otro, donde los hechizos hagan el papel de oración. Como tal, está formado por diversos sintagmas regidos por una gramática que les aporta sentido y coherencia.

Si la oración escrita por el usuario cumple la norma a rajatabla, el conjuro se realizará exactamente tal y como el jugador lo haya definido en su frase. La gramática admitirá tanto alocuciones simples y rápidas como complejas y potentes de forma que el sistema sea fácil de aprender y difícil de dominar. De esta manera se animará al jugador a tratar de controlar la gramática.

Esta lingüística será implementable en cualquier plataforma objetivo. No obstante, el autor ha decidido que el proyecto tratará de enfocar su aplicación a dispositivos móviles dado el continuo crecimiento de su mercado en estos últimos diez años.

En definitiva, el objetivo de este trabajo es presentar una alternativa al sistema de magia clásico, adaptable a cualquier juego de rol de género fantástico, que fomente la creatividad y la experimentación. De esta manera, se espera poder inducir una sensación de satisfacción, realización e inmersión en el jugador. Programar esta gramática en un videojuego implica también replantearse otros aspectos, como por ejemplo los apartados de visualización y de audio. Estos se tendrán que adaptar de forma que el *software* sea capaz de asignar los efectos visuales y sonoros adecuados a cada conjuro para que el jugador reciba el *feedback* satisfactorio.

CAPÍTULO 2: ESTADO DEL ARTE

Tras haber definido el objetivo del TFG, se comentarán algunos ejemplos que han servido de inspiración o bien pueden beneficiarse de la inclusión de nuestro proyecto en el suyo. De esta manera se destacarán qué elementos han conseguido agradar a su público objetivo, ver las posibles carencias que se deben evitar y, por último (pero no por ello menos importante), tener una visión clara de los posibles competidores con los que nuestro proyecto tenga que compartir audiencia.

2.1. ARX FATALIS

Arx Fatalis es un videojuego de temática RPG en primera persona producido por la compañía de Arkane Studios, famosos por títulos como la saga Dishonored. Fue lanzado en PC el 12 de noviembre de 2002 y posteriormente el 23 de diciembre de 2003 para Xbox [6].

La acción del videojuego tiene lugar en un mundo donde el sol no existe y el ser humano debe refugiarse en cuevas para protegerse de los monstruos que ahora habitan en la superficie. A diferencia de otros ejemplos, Arx Fatalis introdujo nuevas mecánicas como la habilidad de conjurar hechizos mediante el dibujo de trazados con el ratón, apreciable en la Figura 2.

Aunque el juego no gozó de mucho éxito comercial, fue bien recibido por la crítica. En Metacritic [7], este consiguió agenciarse un 77/100 y un 7’8/10 en Meristation [8].

Figura 2: Sistema de entrada de hechizos de Arx Fatalis. Fuente: [9]

2.2. THE ELDER SCROLLS III Y IV

El tercer episodio de la saga The Elder Scrolls (Morrowind) es un videojuego de rol desarrollado por Bethesda Game Studios y publicado en Windows el 1 de mayo de 2002. Un mes después se lanzó en Xbox (6 de junio), aunque su lanzamiento en Europa se retrasó hasta el 22 de noviembre de ese mismo año [10]. Por su parte, la cuarta entrega (The Elder Scrolls IV: Oblivion), se estrenaría en PC, Xbox 360 y PlayStation 3 el 20 de marzo de 2006 [11].

Aunque la acción y la historia transcurre en lugares diferentes (países vecinos), estos juegos comparten una serie de características comunes. Entre ellos, es interesante resaltar el sistema de creación de conjuros. Tanto en la tercera como en la cuarta entrega, el jugador puede obtener efectos mágicos mediante la adquisición de hechizos a los diversos comerciantes que habitan el mundo. Las características de estos efectos (magnitud, duración, etc.) pueden editarse y ser combinados con otros para formar conjuros más potentes.

La saga de Bethesda es, sin lugar a dudas, la más popular y aclamada de esta sección. Estos dos títulos han conseguido valoraciones de 89 y 94 respectivamente en Metacritic [12] [13].

2.3. LOST MAGIC

Lost Magic es un videojuego RPG de estrategia producido por Taito y lanzado para Nintendo DS el 19 de enero de 2006 en Japón. Su lanzamiento en el resto del mundo se produjo tres meses después, en concreto el 25 de abril de ese mismo año [14].

El videojuego implementó un sistema de magia basado en la combinación de runas de distintos elementos (fuego, hielo, aire…) con un orden determinado. Los dibujos se conjuran de forma similar a Arx Fatalis (Figura 3) pero adaptado a la nueva Nintendo DS (con el trazado de los símbolos en la pantalla inferior de la consola mediante el lápiz táctil). El jugador puede aumentar la potencia de sus conjuros con la fusión de runas del mismo elemento o el aumento de la similitud del trazado su referencia. No obstante, hacer esto requiere exponerse a los enemigos cercanos.

Pese a todo, Lost Magic pasó desapercibido ante los ojos de muchos. La crítica profesional no favoreció al juego, recibiendo un 66 por parte de Metacritic [15]. Aun así, logró gustar a muchos de sus usuarios.

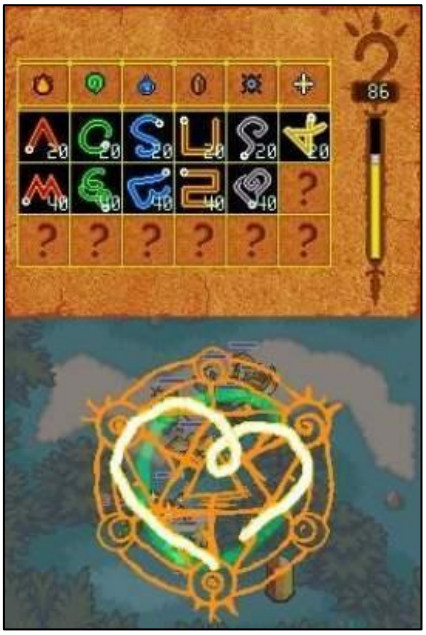


Figura 3: Sistema de runas de Lost Magic. Fuente [16]

2.4. ERAGON (NINTENDO DS)

De forma similar a Lost Magic, Eragon es un videojuego de aventuras desarrollado por Amaze Entertainment y lanzado en Nintendo DS, Game Boy Advance y PlayStation Portable el 14 de noviembre de 2006 [17]. Su estreno en Europa se realizaría diez días más tarde.

De todas sus versiones, la versión de DS fue sin duda la más interesante. La desarrolladora trató de aprovechar el panel táctil que incluía la consola, con la implementación de un sistema de control mixto que combinaba el uso del panel táctil con los botones más tradicionales. Estos servían junto a la cruceta para realizar acciones como moverse por la escena y atacar cuerpo a cuerpo mientras que el puntero podía utilizarse en tareas auxiliares como conjurar hechizos o utilizar objetos curativos. Estas acciones se activaban mediante el dibujo de símbolos (Figura 4), que se hacían más complejos cuanto más poderoso era el hechizo o el objeto utilizado.

Eragon tuvo también versiones para las consolas de sobremesa, aunque la recepción de estas resultó ser generalmente negativa. La versión de Nintendo, sin embargo, consiguió agenciarse un 63 por parte de Metacritic [18].

Figura 4: Sistema de hechizos en Eragon. Como podemos ver, el jugador dibuja un rayo para lanzar un conjuro eléctrico. Fuente: [19]

2.5. TWO WORLDS II

El segundo episodio de Two Worlds es un videojuego RPG estrenado en las plataformas de PC, Xbox 360 y Playstation 3 en noviembre de 2010. Fue desarrollado por la empresa polaca Reality Pump [20].

Este juego es sin lugar a duda el que más se parece a la idea de nuestro proyecto. Cabe destacar que fue bien valorado por introducir uno de los sistemas de magia más complejos e innovadores jamás construidos en la historia de los videojuegos. En él, los hechizos se crean mediante la combinación de cartas de diversos tipos, donde cada una aporta diversas propiedades y efectos al conjuro (daño, velocidad, comportamiento, etc.). Una vez realizada esta configuración, el jugador deberá pagar una cantidad específica de oro para poder quedarse con el hechizo. A lo largo del transcurso del juego, la efectividad de los conjuros mediante la acumulación de naipes de un mismo tipo.

No obstante, el juego tuvo una recepción mediocre debido a su trama poco atractiva y un control algo anticuado para la época. Sin embargo, Two Worlds II consiguió obtener un notable (75) en Metacritic [21].

2.6. OUTWARD

Outward es un videojuego estrenado en PC, Xbox One y Playstation 4 el 26 de marzo de 2019 [22]. Producido por el pequeño estudio canadiense Nine Dots Studio, el proyecto trata de combinar los géneros de rol y de supervivencia.

La magia de Outward está bastante condicionado por el sistema de habilidades del mismo. En él, el jugador puede aprender nuevas ventajas y movimientos comprándolos a varios maestros repartidos por el mapa. Cada entrenador tiene aptitudes únicas, por lo que el usuario deberá ir a su localización para poder adquirirlas. Entre todos los tutores, existe uno que se especializa en el uso de runas (Figura 5), un tipo de estilo de combate que se basa en la combinación entre ellas. Por ejemplo, una runa A se combina con otra B realiza un hechizo C. El sistema es sensible al orden, por lo que si hacemos la combinación BA se ejecuta otro conjuro D.

La acogida fue mixta, pues ganó una puntuación de 67 en Metacritic [23]. No obstante, el juego ha conseguido mantener una pequeña comunidad de nicho. Gracias a ello, la desarrolladora ha continuado trabajando en él, lanzando dos expansiones descargables durante estos últimos años. Por último y como dato adicional, Outward fue desarrollado en Unity. Más adelante se mostrará el porqué.

Figura 5: Ficha de combinaciones de runas en Outward. Fuente: [24]

2.7. VALHEIM

Por último, Valheim es un juego de mundo abierto y rol producido por Iron Gate Studio prelanzado (o puesto a la venta) en PC el 2 de febrero de 2021 [25]. Este es un *early access*, es decir, todavía no ha finalizado su etapa de desarrollo y está sujeto a cambios.

Siendo el videojuego más novedoso de la lista, en esta aventura el jugador no tiene ningún objetivo principal. Su fin es sobrevivir al entorno y obtener poco a poco objetos que le faciliten cumplir este propósito. Aunque la acción tiene lugar en el mundo místico de Valheim (Figura 6), el juego no incluye por el momento ningún hechizo accesible por el usuario. Dadas sus características, quizás sería muy interesante añadir al *gameplay* actual un sistema de magia como el que propone el TFG. De esta manera, se aprovecharía su trasfondo y las mecánicas de fabricación ya existentes.

Para ser un juego con pocos meses de vida, la desarrolladora escandinava ha conseguido vender ya cinco millones de copias y es uno de los títulos más jugados en la actualidad. Al no haber finalizado su etapa de producción, Valheim todavía no ha sido evaluado en Metacritic. Aun así, el 96% de las reseñas de Steam [26] (la principal plataforma distribuidora) son positivas.

Figura 6: Panorámica de Valheim. Como podemos ver, la estética mística admitiría la inclusión de la idea del TFG. Fuente: [27]

2.8. CONCLUSIÓN

El estado del arte podría resumirse en que estos títulos han provocado una reacción mixta en el público jugador: unos como la saga The Elder Scrolls o Valheim han tenido un éxito sin precedentes mientras que otros han pasado mucho más desapercibidos. No obstante, los siete títulos han conseguido de una forma u otra fascinar a sus jugadores con sus innovaciones en el sector gracias a las mecánicas introducidas.

Estas pueden ser muy variadas, desde la edición de hechizos hasta poder dibujar conjuros en la pantalla aprovechando los periféricos de la consola objetivo. De esta manera, el usuario consigue ponerse en la piel del protagonista que controla y sentir que está haciendo algo más que pulsar un botón.

Por otro lado, es importante destacar la relevancia de la facilidad de uso del sistema en esta clase de proyectos. La magia de Two Worlds II puede parecer muy atractivo sobre el papel, pero en la práctica estas mecánicas fueron demasiado complejas y complicadas para sus jugadores. Además, la desarrolladora Reality Pump no incluyó apenas ningún tipo de tutorial. En consecuencia, la mayoría de personas que jugó al videojuego optó por utilizar otras mecánicas como el combate cuerpo a cuerpo o el uso del sigilo, que eran mucho más sencillas e igualmente efectivas.

En conclusión, este proyecto tratará de aportar un nuevo enfoque al *gameplay* existente mediante la creación de un sistema de magia con una variedad de hechizos nunca vista en ningún videojuego con la intención de aprovechar al máximo las posibilidades de la plataforma objetivo. De esta manera, se intentará innovar en el campo y conseguir un nicho de jugadores que tenga el mismo tipo de inquietudes que las planteadas en este proyecto.

CAPÍTULO 3: SCOPE

3.1. OBJETIVOS

Como objetivo principal, el TFG quiere presentar una alternativa al sistema de magia clásico, adaptable a cualquier juego de rol de género fantástico, que fomente la creatividad y la experimentación del jugador. Para ello, los subobjetivos se han clasificado en dos ramas.

Por una parte, se han definido una serie de efectos y de comportamientos base fundamentándose en diversos conjuros sencillos clásicos. Estos no suponen nada por sí mismos, sino que cobrarán significado una vez se unan a otros elementos. Es por ello que, como primer subobjetivo, los efectos deben de ser modulares y combinables entre ellos. De esta manera, se transmite una gran sensación de variedad que se quiere obtener en el resultado final.

Para definir las normas de asociatividad entre estos comportamientos, el TFG pretende implementar una gramática (o *parser*) que se encargue de recibir la información del jugador y la materialice en un objeto que constituirá nuestro hechizo/conjuro. Esta gramática deberá permitir cierta flexibilidad y conforme un sistema *easy to learn, hard to master*. Es decir, que cualquier usuario pueda aprender rápidamente su funcionamiento, pero admita cierta complejidad y profundidad que poder explorar. Un resumen puede verse en la Figura 7.

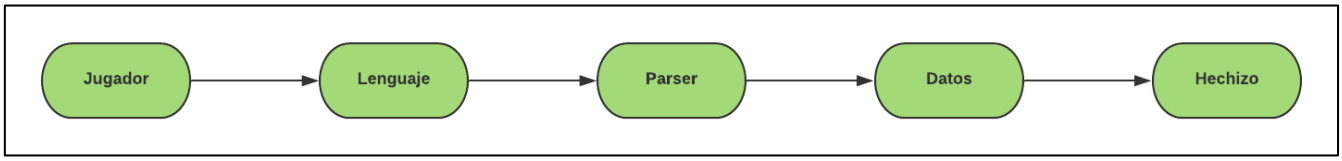


Figura 7: Esquema general de funcionamiento. Fuente: Elaboración propia

En segundo lugar, el proyecto ha desarrollado un pequeño videojuego, que será referenciado como FPP (First Playable Prototype), en uno de los motores de videojuegos más populares: Unity. De esta manera se pueden probar los efectos y la gramática comentados en el primer objetivo. Este prototipo cumplirá una serie de subobjetivos específicos descritos a continuación.

- Diseño y producción de un nivel principal poblado de distintos objetos, personajes y animales donde el jugador pueda crear y probar sus combinaciones de hechizos personalizadas.
- Creación de un sistema dinámico de partículas y de sonido que sea capaz de adaptarse y de generar los efectos visuales adecuados para todos los hechizos, puesto que sería imposible asignar un efecto a cada combinación.
- Opcionalmente, si se dispone de tiempo suficiente, el TFG plantea otros aspectos:
    - Un pequeño sistema multijugador de hasta cuatro personas para maximizar el grado de entretenimiento que nos puede dar el juego.
    - Un sistema de misiones y objetivos que recompensen al jugador por experimentar y le dé un motivo con el que seguir jugando (y retenerlo para que siga jugando).
    - Por último, crear una mecánica de jefes para animar al jugador a conocer todas las posibilidades de la gramática y a cooperar entre varias personas.

3.2. RIESGOS Y OBSTÁCULOS

Por desgracia, el TFG ha contado desde el inicio con una serie de riesgos y obstáculos que se han debido tener en cuenta para asegurar el éxito del trabajo. Estos problemas se han tratado de manera inmediata con el objetivo de asegurarse que no desemboquen en factores críticos más graves.

- **Período de tiempo limitado:** al tratarse de una asignatura más del grado, el proyecto conlleva una fecha límite de entrega, por lo que su desarrollo ha estado inevitablemente condicionado por la misma. La gestión del tiempo ha sido vital para garantizar el buen desarrollo del trabajo. En caso de contratiempos, ha tocado priorizar y enfocarse en aquellas tareas que sean realmente importantes.
- **Inexperiencia en algunas de las tecnologías utilizadas:** aunque el desarrollador ya ha tenido experiencias previas con ciertas herramientas utilizadas para la producción del proyecto, hay otras que le han sido totalmente desconocidas y, por lo tanto, ha tenido que invertir cierta cantidad de tiempo para dominarlas.

- **Compatibilidad entre plataformas:** desarrollar el FPP para Android ha acarreado de manera inevitable problemas a la hora de realizar tests, diseñar la interfaz y asegurar la compatibilidad de todos los plugins y librerías que se han utilizado en el proyecto.

3.3. METODOLOGÍA

Debido a las importantes limitaciones que tiene el proyecto (margen de tiempo limitado e inexperiencia en ciertas tecnologías) y el número reducido del equipo desarrollador (se consta solamente de una persona), se ha optado por seguir una metodología ágil basada en múltiples iteraciones de trabajo. En concreto, se ha tomado Scrum (Figura 8) como marco de referencia al cual se han realizado algunas modificaciones para adaptarlo a la forma de trabajar del equipo.

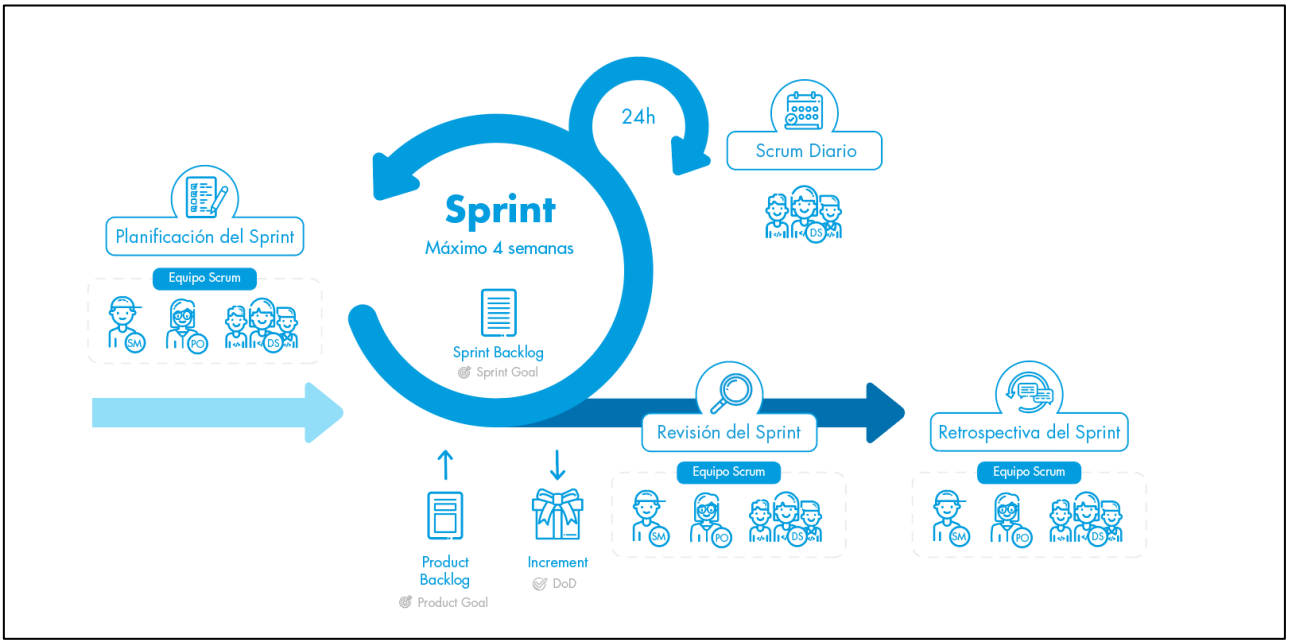


Figura 8: Diagrama de la metodología Scrum original. Fuente: [28]

La duración de cada sprint o iteración no es fija, pero se puede establecer en una semana por defecto. De esta manera, el proceso de producción se ha hecho más flexible, haciéndolo más adaptable a los contratiempos que puedan surgir a lo largo del desarrollo del proyecto.

En específico, las primeras semanas han conformado la fase inicial o *inception* del trabajo. En ellas, se definen todas las características, requisitos fundamentales y el *scope* del proyecto. Esta fase corresponde con el curso de Gestión de Proyectos que finalizó el 24 de marzo de este año. Una vez acabada, el trabajo se divide en pequeños *sprints* semanales acordados y verificados por el director. Cada una de estas iteraciones se ha agrupado temáticamente en el capítulo de Planificación para facilitar su presentación. Además, el autor ha redactado este documento en paralelo para evitar una acumulación de tareas en la etapa de finalización del proyecto.

En cuanto a los roles de la metodología, se ha adoptado una vista muy simplificada del Scrum original. El papel de *Product Owner* (PO) del proyecto lo ha llevado a cabo el director del mismo. Él ha sido el representante de los intereses del cliente, definido sus funcionalidades y comunicado su progreso al ponente. Por otro lado, el rol del equipo desarrollador (DT) lo ha representado el propio autor del proyecto, puesto que ha sido la persona encargada de implementar las funcionalidades del mismo. El Scrum Master no ha figurado en el TFG ya que su presencia no tiene mucho sentido en un equipo de dos personas.

Al final de cada sprint, el *Product Owner* y el desarrollador se reúnen en una sala de Meet para comentar todas las tareas que se han realizado en retrospectiva (*sprint review*). Su fecha y hora no son fijas, aunque por defecto la reunión se estableció cada viernes a las 17:00h.

En ella se identifica aquello que se puede dar por finalizado y todas las funcionalidades que no se han logrado realizar o pueden mejorarse. El director también trata resolver las posibles dudas que haya podido tener el desarrollador durante la iteración. En caso de que exista un problema de pequeña envergadura o que convenga resolverlo de manera urgente, siempre existe un canal de Google Hangouts. Gracias a él, los miembros de todos los roles pueden establecer contacto de forma casi inmediata.

Por último, teniendo en consideración el desarrollo del último *sprint*, se traza una nueva planificación de la próxima fase (*sprint backlog*). Esta siempre se ha realizado en la misma reunión que la anterior para aprovechar el tiempo y mantener la inercia de trabajo.

CAPÍTULO 4: ANÁLISIS DE REQUISITOS

4.1. ACTORES IMPLICADOS

Como cualquier otro proyecto de envergadura académica o comercial, se pueden identificar varios colectivos interesados en él en mayor o menor medida, también conocidos como *stakeholders*. El TFG tiene que asegurarse su apoyo mediante el cumplimiento de los objetivos. Esto se debe a que repercuten de forma directa en el resultado final del proyecto y de su éxito. Por ello, conviene identificarlos antes de iniciar las etapas de diseño e implementación (Figura 9).

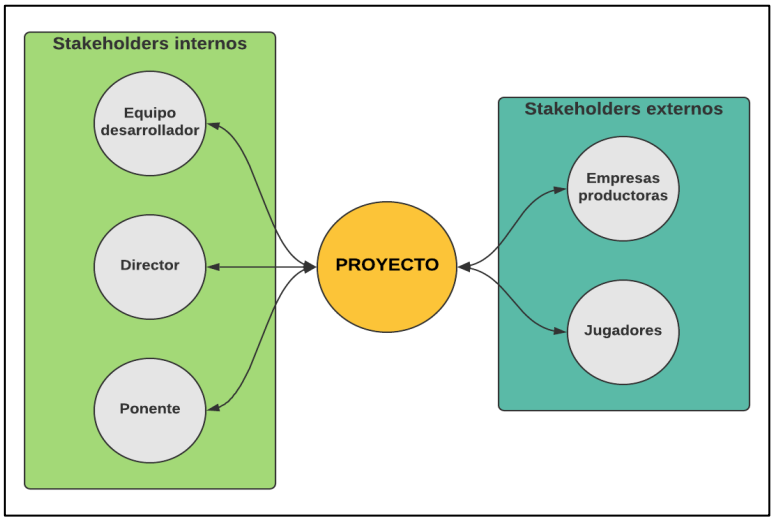


Figura 9: Clasificación de los distintos stakeholders del proyecto. Fuente: Elaboración propia

Jugadores

Aunque de implicación bastante baja en el desarrollo, los jugadores son de vital importancia para el proyecto puesto que, en caso de no existir, este carecería de sentido. Además, los jugadores son el cliente final y, por lo tanto, los que determinan el éxito de la aplicación. El público comprende a todos los aficionados a juegos RPG de los sectores *teens*, *young adults* y *adults* (14 años o más).

El FPP que se ha realizado en demostración de la gramática está desarrollado de forma exclusiva para Android (debido a la escasez de tiempo), que determina de cierta manera el público objetivo anterior. No obstante, la gramática es universal y es totalmente independiente de la tecnología escogida. En otras palabras, esta es implementable de forma fácil en cualquier plataforma (PC o consolas), por lo que podemos ampliarlas de forma sencilla.

Empresas productoras

Si bien las empresas productoras de videojuegos tienen poca influencia en el proyecto, estas estarían muy interesadas en su éxito. Esto se debe a que les abriría una nueva vía de innovación en futuros videojuegos. En concreto, la gramática interesaría especialmente a empresas desarrolladoras y distribuidoras que tendrían una idea sobre la que basar su próximo videojuego, sin importar la plataforma final de lanzamiento.

Por otro lado, es por ello que es importante entender que el TFG no considera al resto de empresas como competencia, sino como un posible socio con el que poder colaborar y establecer una relación de beneficio mutuo.

Equipo desarrollador

En cualquier software, el equipo desarrollador es uno de los actores más importantes en su despliegue puesto que es el encargado de dar a luz la idea del *Product Owner*. Al fin y al cabo, ha sido el encomendado de diseñar, programar, testear y documentar la totalidad del proyecto.

Tratándose de un trabajo de fin de carrera, el equipo desarrollador está formado por único programador: el autor mismo. En este caso específico, existe también el objetivo personal en que el trabajo haya funcionado de forma exitosa puesto que se reflejará más adelante en su expediente.

Director del proyecto

Manuel Rello Saltor, como profesor de la Universidad Politécnica de Catalunya, también ha tenido una implicación directa en el desarrollo del proyecto. Como ya hemos mencionado antes, el director ha llevado a cabo el papel de *Product Owner* en el trabajo, encargándose de supervisar y tutorizar al desarrollador para asegurar su correcta producción. Sin él, el trabajo no hubiera sido posible o, mínimamente no se hubiera efectuado de la misma manera.

Además, como ingeniero jefe de proyectos en Gameloft, su experiencia en la empresa ha sido vital en la toma de decisiones más enfocadas al mundo de los videojuegos. Algunos ejemplos son todas aquellas relacionadas con el diseño del FPP: planteamiento de la arquitectura, elección de patrones de diseño, técnicas de iluminación, etc.

Ponente

Por otra parte, Ernest Teniente López se ha encargado de garantizar que el trabajo cumpla todos los requisitos de la normativa y evaluar los objetivos en colaboración con el Director. Como Catedrático de la misma universidad, su consejo ha sido en especial importante en la concepción y diseño de la gramática de los próximos capítulos.

4.2. REQUISITOS FUNCIONALES

En este apartado veremos el listado de funcionalidades que el proyecto ha implementado y que define su uso. Estas se agruparán en varios subapartados en función del área a la cual pertenezcan. Los requisitos funcionales se mostrarán de forma detallada en formato de ticket o historias de usuario utilizando el modelo INVEST [29].

Crear hechizo

Como jugador del FPP, quiero ser capaz de introducir una combinación de sintagmas para obtener un hechizo que poder utilizar después.

Criterios de aceptación:

- El jugador no puede introducir una combinación vacía.
- El jugador no puede encontrarse en combate en ese mismo momento.

Sistema de partículas

Como jugador del FPP, quiero poder ver en la pantalla los efectos visuales correctos para saber el hechizo exacto que he realizado.

Criterios de aceptación:

- El hechizo debe de estar definido en la escena.
- Todas las propiedades del conjuro deben de estar definidas.

Sistema de sonido

Como jugador del FPP, quiero escuchar un sonido conveniente para cada evento que suceda en la escena para aumentar el grado de inmersión en la acción.

Criterios de aceptación:

- Existe un sonido para el evento ocurrido.
- El jugador se encuentra a una distancia dentro del rango audible de la fuente de sonido a reproducir.

Sistema de animaciones

Como jugador del FPP, quiero que las acciones que realice el protagonista y las entidades correspondan con su movimiento para obtener el *feedback* adecuado.

Criterios de aceptación:

- La entidad animada ha realizado una acción.
- Si la entidad estaba realizando ya una acción, esta es cancelable.

Mover protagonista

Como jugador del FPP, quiero ser capaz de cambiar mi posición en el nivel para poder explorarlo e interaccionar con las entidades que lo habitan.

Criterios de aceptación:

- El jugador debe de haber pulsado la pantalla táctil de su dispositivo.
- El jugador no ha pulsado ningún botón.
- La posición de destino debe de ser accesible.

Conjurar hechizo

Como jugador del FPP, quiero poder dibujar la representación de un hechizo para instanciarlo en la escena y así producir un efecto en cualquier personaje de la misma.

Criterios de aceptación:

- El jugador ha pulsado previamente el botón para conjurar hechizos.
- El hechizo ha sido definido de forma previa.
- El resultado tiene un parecido mínimo al dibujo del jugador.

Modificar vida

Como jugador del FPP, quiero tener la posibilidad de cambiar el estado de vida de una entidad para poder defenderme o curarla de las posibles heridas que pueda recibir.

Criterios de aceptación:

- La entidad tiene un efecto activo de vida.

Modificar resistencia

Como jugador del FPP, quiero tener la posibilidad de cambiar la resistencia a un elemento de una entidad para hacerla más o menos resistente a los efectos basados en el mismo.

Criterios de aceptación:

- La entidad tiene un efecto activo de resistencia.

Modificar velocidad

Como jugador del FPP, quiero tener la posibilidad de cambiar la velocidad de movimiento de una entidad para esquivar proyectiles con mayor facilidad o tener más tiempo de huida.

Criterios de aceptación:

- La entidad tiene un efecto activo de velocidad.

Perder partida

Como jugador del FPP, quiero ser capaz de perder combates con otras entidades para así ayudarme a mejorar en el diseño de hechizos y en la toma de decisiones.

Criterios de aceptación:

- La vida del jugador es menor o igual a 0.

Actividad de las entidades (NPCs)

Como jugador del FPP, quiero que los NPCs (personajes no jugables) de la escena tengan un comportamiento determinado para tener alguien con quien poder interactuar.

Criterios de aceptación:

- La escena está cargada y ejecutándose en tiempo real.

Respawn de NPCs

Como jugador del FPP, quiero que algunas entidades sean capaces de reaparecer para poder seguir interaccionando con ellas en caso de que desaparezcan.

Criterios de aceptación:

- La cantidad de copias de un NPC es inferior al máximo establecido.
- El punto de creación no está en el campo de visión del jugador.
- El jugador se encuentra a una distancia mínima de 30 metros.

Salir del juego

Como jugador del FPP, quiero ser capaz de abandonar la partida en cualquier momento para dejar de jugar.

Criterios de aceptación:

- El menú principal se encuentra abierto.

Modo de bajo consumo

Como jugador del FPP, quiero poder bajar el límite de fotogramas del juego para reducir la energía que la aplicación consume en mi dispositivo y alargar la duración de la batería del móvil.

Criterios de aceptación:

- El menú principal se encuentra abierto.

4.3. REQUISITOS NO FUNCIONALES

Por otro lado, el proyecto también cuenta con unos requisitos no funcionales que definen la calidad general del videojuego. Estos requerimientos son igualmente importantes a las cláusulas definidas en el apartado anterior, y por ello serán especificados con la terminología de Volere [30]:

Requisito de apariencia (Volere 10a)

- ***Descripción:*** el FPP debe de tener una iluminación y estilo artístico atractivos, aunque no tienen por qué ser extremadamente detallados.
- ***Justificación:*** la primera impresión que obtiene un jugador respecto un videojuego llega a través de la visual del mismo. Por ello, el juego tiene que entrar por los ojos para asegurar una buena percepción inicial del mismo.

Requisito de estilo (Volere 10b)

- ***Descripción:*** el FPP tiene que ser divertido de jugar y no resultar estresante para el usuario.
- ***Justificación:*** si el FPP no es divertido, los esfuerzos del proyecto habrán sido en vano ya que el jugador no va a valorar el trabajo que hay detrás del sistema de creación de hechizos. Por otra parte, las posibles empresas colaboradoras tampoco verán el potencial máximo que puede ofrecer esta gramática.

Requisito de usabilidad (Volere 11a)

- ***Descripción:*** el FPP debe de tener una *User Experience* correcta, lograda con un sistema de movimiento cómodo y una interfaz poco saturada con la menor cantidad de botones posible.
- ***Justificación:*** otro de los símbolos de calidad de un videojuego es la fluidez del *gameplay* y la forma en que se enseña la información al usuario. Es por ello importante hacer especial hincapié en este apartado.

Requisito de aprendizaje (Volere 11c)

- ***Descripción:*** el sistema de magia tiene que ser intuitivo y fácil de aprender por parte del usuario.
- ***Justificación:*** la gramática que rige los hechizos va a ser compleja, por lo que se deberá prestar una ayuda al jugador para que aprenda a usarlo y no se frustre, vía un manual de usuario o vídeos explicativos integrados dentro del juego.

Requisito de rendimiento (Volere 12a)

- ***Descripción:*** el producto final tiene que ser capaz de ejecutarse utilizando la menor cantidad de recursos posible del sistema.
- ***Justificación:*** el FPP será ejecutado en un dispositivo móvil, por lo que deberá consumir poca energía para que la duración de su batería se vea lo menor mermada posible.

Requisito de escalabilidad (Volere 12g)

- ***Descripción:*** la gramática debe de ser fácilmente escalable y ampliable con nuevos elementos, efectos y comportamientos.
- ***Justificación:*** uno de los puntos fuertes del proyecto es el número de combinaciones posibles de conjuros que puede llegar a crear el jugador. Si se quiere exportar este sistema a otros juegos, la gramática necesitará adaptarse a muchos entornos.

Requisito de adaptabilidad (Volere 14c)

- ***Descripción:*** el sistema de magia tiene que ser capaz de implementarse en cualquier videojuego en producción.
- ***Justificación:*** si la idea es poder colaborar con el máximo número de estudios desarrolladores posible, la gramática debe de ser universal y no puede depender de a nivel tecnológico de la implementación.

CAPÍTULO 5: CONCEPTOS PREVIOS

Antes de entrar en profundidad con el diseño y la implementación del sistema, es interesante repasar algunas ideas fundamentales que se repetirán de forma continua a lo largo del trabajo. Se hará especial hincapié en qué consiste cada una y cómo afectan al desarrollo del TFG.

Gramática y sintaxis

Según la tercera entrada del diccionario de la Real Academia Española [31], la gramática es la parte de la lingüística que estudia los elementos de una lengua y su forma en que estos se organizan y se combinan. En concreto, la sintaxis es la parte de la gramática que estudia el modo en que se combinan las palabras y los grupos que estas forman para expresar significados, así como las relaciones que se establecen entre todas esas unidades [32].

Si se trasladan estos dos conceptos al proyecto, la gramática y la sintaxis constituirán el conjunto de leyes y de normas que el jugador deberá seguir para fabricar un hechizo que tenga sentido. La gramática del trabajo es tan necesaria como en la lengua misma puesto que, de no existir, el software no es capaz de interpretar de ninguna manera la entrada que recibe del usuario.

Sintagma

También un término lingüístico (según la propia RAE) [33], un sintagma es una palabra o conjunto de palabras que se articula en torno a un núcleo y que puede ejercer alguna función sintáctica o, dicho de otra manera, que pueden expresar significados. Esta última parte es la que interesa, ya que las propiedades de los conjuros se conformarán a partir de ellos. De forma similar a cualquier idioma, se han creado diversos tipos de sintagmas que expresan cualidades distintas. Esto, sin embargo, pertenece a otro capítulo y se explicará más adelante.

Intérprete

Un intérprete es un programa informático encargado de ejecutar líneas de código (escritas en alto nivel) directamente sin tener que ser traducidas/compiladas a lenguaje máquina de forma previa.

Son muy utilizados en tareas de virtualización y de emulación, donde se necesitan varias copias del mismo programa o en plataformas diferentes a la originalmente planificada [34]. Los intérpretes suelen realizar varias tareas entre las cuales se destacarán tres que forman parte del proceso de interpretación [35].

- **Análisis léxico:** inicialmente el *lexer*, que es la parte del programa que se encarga de analizar el código fuente de entrada, rompe este código en cadenas de texto más pequeñas. A partir de ellas, se procede a extraer unidades mínimas de información, también conocidas como tokens. En cierta forma, se parecen a las palabras de cualquier idioma como por ejemplo el castellano.
- **Análisis sintáctico:** ejecutada a continuación del análisis léxico, en esta fase se aplica la sintaxis del lenguaje para comprobar si los tokens generados en la primera fase forman una expresión con sentido pleno [36]. Si se ejecuta con éxito, el *parser* codifica la información en lo que se conoce como un Abstract Syntactic Tree (AST), como se muestra en la Figura 10.

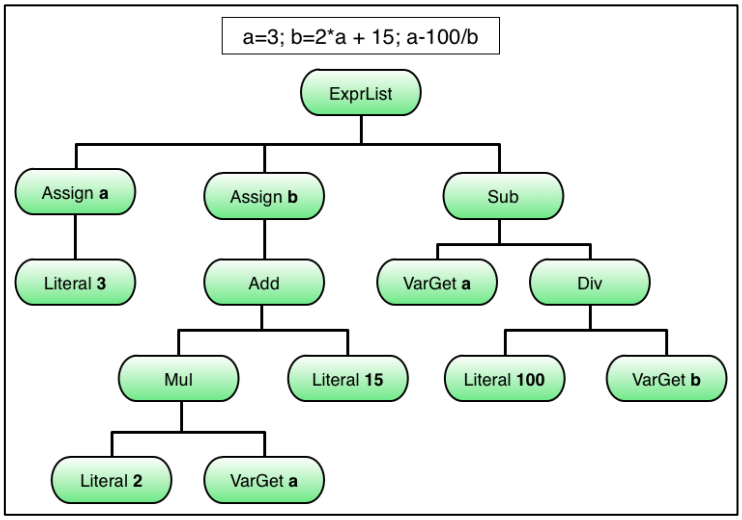


Figura 10: Ejemplo de estructura de un AST. Fuente: [37]

- **Análisis semántico:** finalmente, en esta parte se comprueba si hay algún error en el AST y se determina el significado real del código que teníamos inicialmente. A efectos prácticos, es en esta fase donde el intérprete puede traducir las instrucciones a cualquier otro lenguaje que soporte.

Game Loop

Es el componente central de todo videojuego. Gracias a él, este es capaz de funcionar incluso en casos en los que no existan eventos externos. A diferencia de otros programas, los juegos tienen que ser capaces de ejecutarse de forma independiente a la entrada del usuario.

Para cada fotograma que renderiza la tarjeta gráfica, el procesador debe encargarse de captar el estado de mando y actualizar la escena. Esto se realiza mediante cálculos pertenecientes a diferentes áreas: físicas, inteligencia artificial, sonidos, etc. Todo el proceso se muestra en un esquema simplificado en la Figura 11.

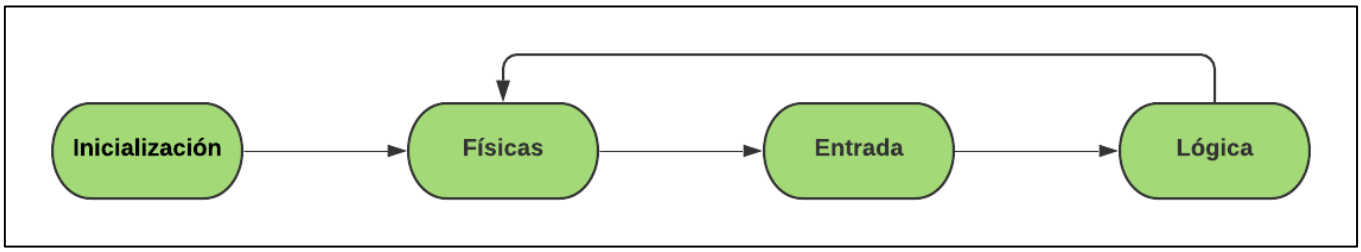


Figura 11: Diagrama simplificado de las fases del Game Loop. Fuente: Elaboración propia con datos de [38]

En la mayoría de *game engines* se implementan métodos polimórficos que permiten especificar el código que se ejecutará dentro del bucle. En el caso de Unity, este genera de manera automática en cada clase las funciones *Start()* y *Update()* que se ejecutan antes de renderizar y durante cada fotograma, respectivamente.

CAPÍTULO 6: MODELO CONCEPTUAL

Antes de crear la gramática que se ha ido mencionando a lo largo del documento, se tiene que especificar las partes que la conforman. Para ello, en este apartado se describirán sus componentes y más adelante los representaremos de forma gráfica mediante la creación de un esquema conceptual o diagrama de clases.

El diagrama que se puede visualizar en la Figura 13 no es propiamente un modelo, en el sentido clásico del término, sino un metamodelo o, valga la redundancia, un modelo de modelos [39]. En otras palabras, en vez de representar arquetipos concretos (como haría un modelo estándar), este tipo de diagramas permite hacer especial hincapié en la lógica y propiedades de estas ideas. De esta manera, uno puede elaborar un diseño más abstracto que permita representar cualquier tipo de concepto referente al dominio del metamodelo.

Las instancias de este esquema corresponden al modelo de hechizos concreto creado para un juego determinado, pero las formas desarrolladas para manipular y animar estos conjuros se han definido a nivel de metamodelo. De esta manera, la solución aportada es aplicable a todos los posibles modelos concretos instanciados posteriormente.

Al prestar uno atención, este verá que una gramática cumple con una función muy parecida a esta clase de esquemas. Si por ejemplo se considera la oración "El perro es entrenado por su dueño" (Figura 12), esta idea no se especifica con algo en concreto como podría ser el círculo '○'. Más bien, se construye una oración con sentido completo que resulta la combinación de varias propiedades o sintagmas.

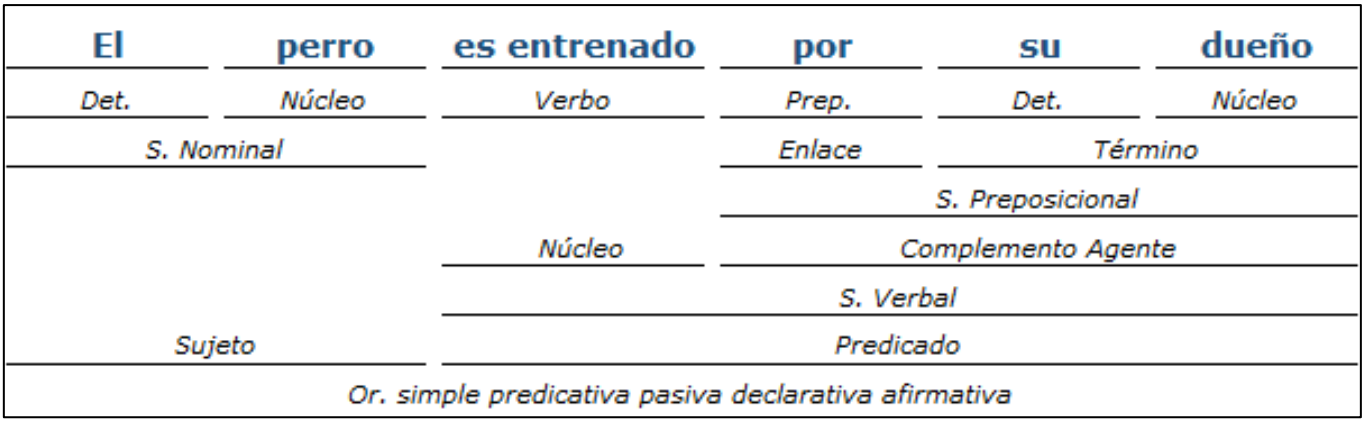


Figura 12: Análisis morfosintáctico de la oración. Fuente: [40]

La gramática que quiere implementar este proyecto persigue la misma ambición. En vez de programar cada hechizo por separado, se pretende implementar por separado las propiedades que lo componen para al final poder combinarlos. No obstante, de forma similar al lenguaje real, cualquier combinación de sintagmas no tiene porqué dar lugar a una oración. El programa no puede recibir una entrada arbitraria, sino que esta debe cumplir una serie de normas impuestas.

Con el objetivo de facilitar la entrada de nuevos jugadores, la gramática tiene que ser capaz de admitir la omisión de ciertos campos y asumir valores por defecto. Por ejemplo, la entrada:

"FROZEN SPEED DEBUFF IN PROJECTILE"

Daría lugar a un proyectil helado que ralentiza a cualquier entidad que colisione con él. El sistema asume magnitudes estándar para todos sus componentes, se mueve en línea recta y se destruye al impactar. De esta forma, el usuario puede ir poco a poco aprendiendo los nuevos efectos añadiéndolos progresivamente en nuevos hechizos. En este otro ejemplo:

"STORMFUL GREAT EVERLASTING RESISTANCE BUFF IN SELF"

Permitiría aumentar enormemente y durante mucho tiempo la resistencia eléctrica del hechicero. La gramática deberá asumir que este conjuro debe incluir el componente *Child*. Para los jugadores más experimentados, se les permite indagar más en las propiedades del sistema. Si el sistema recibiera una entrada del tipo:

"FIRESOME WISE PITTY SPARKING LIFE DEBUFF IN TINY PROJECTILE AS RUNNING IMPULSE AND SPARKING DESTROY"

Se generaría un hechizo con las siguientes características:

- **FIRESOME:** ígneo
- **WISE:** el efecto del hechizo no afecta al jugador.
- **PITTY SPARKING LIFE DEBUFF:** el conjuro causa daño menor a la entidad receptora.
- **TINY PROJECTILE:** proyectil diminuto.
- **RUNNING IMPULSE:** movimiento recto con velocidad estándar.
- **SPARKING DESTROY:** autodestrucción instantánea del hechizo al impactar.

6.1. DIAGRAMA DE CLASES

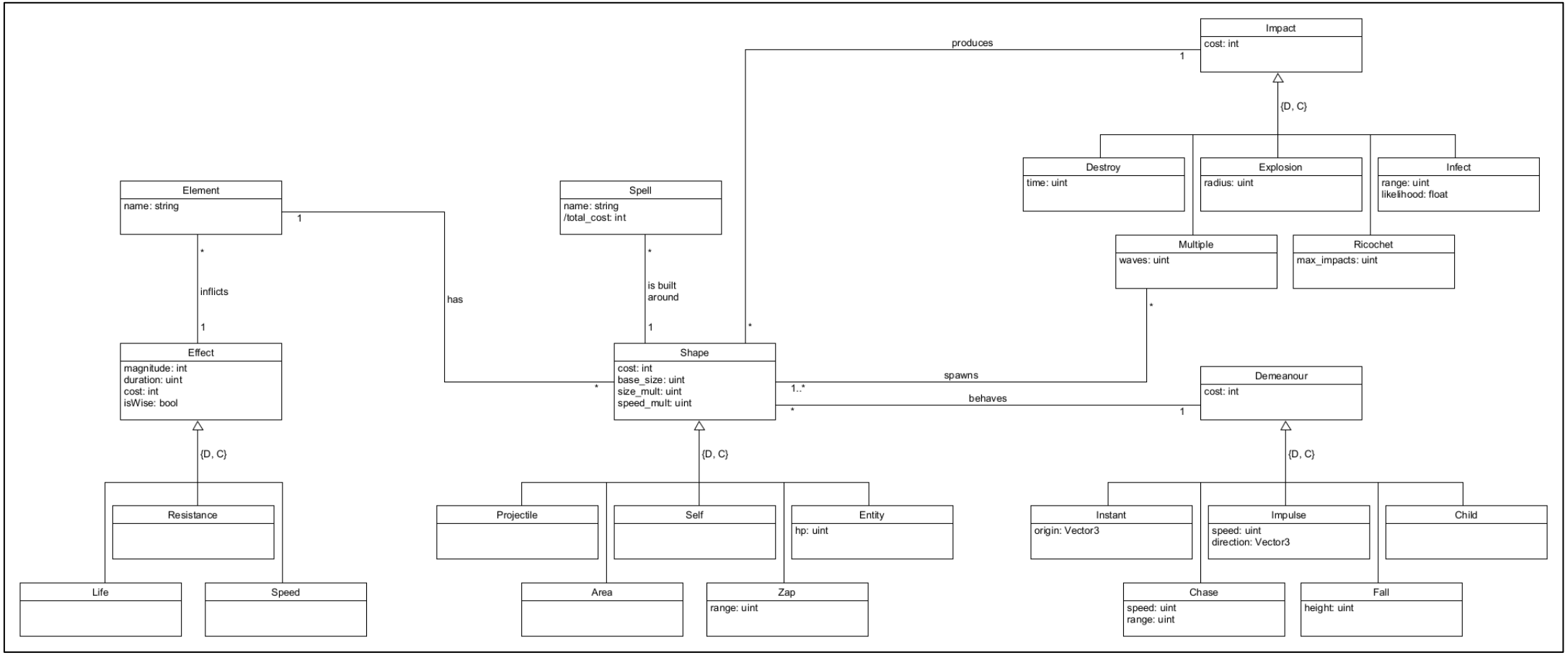


Figura 13: Diagrama de clases del metamodelo. Fuente: Elaboración propia

Restricciones textuales del modelo

- La forma *Self* siempre debe asociarse con el comportamiento *Child*.
- El molde *Zap* no puede relacionarse con *Instant*.
- *Entity* sólo admite la conducta *Instant*.
- El atributo *likelihood* contenido en la clase *Infect* debe de tener un valor perteneciente al intervalo [0, 1].

6.2. DESCRIPCIÓN DE LAS CLASES

El diagrama de la Figura 13 trata de recoger todo esto en varias clases agrupables en tres grupos distintos. Nótese que el diagrama contiene múltiples clases vacías (sin atributos) porque se espera que más adelante implementen funcionalidades distintas.

- **Parte central:** formada por *Spell* y *Shape*, se centra en describir los atributos físicos del hechizo (su tamaño, colisión, molde, etc.).
- **Parte izquierda:** constituida por *Element* y *Effect*, este conjunto contiene los modificadores aplicables al colisionar con el jugador o cualquier NPC.
- **Parte derecha:** por último, este módulo define el comportamiento (*Demeanour* e *Impact*) de la entidad que conforma el conjuro. Algún ejemplo de ello es el patrón de movimiento o la reacción ante otros objetos de la escena.

Spell

Es el concepto que engloba todas las propiedades del hechizo pertenecientes a él. Contiene información general, como su nombre, que lo identifica o el coste mágico total, que es la suma del gasto de cada modificador integrado en el conjuro, además de suponer un acceso centralizado a todos ellos.

Shape

Situándose como la clase central del diagrama, la shape (o forma) es el molde o representación gráfica y espacial de un hechizo. Cada una tiene características únicas que modifican parámetros físicos como la velocidad o el tamaño. Supone el nivel superior de una jerarquía que define sus varios tipos. Gracias a ella el desarrollador puede añadir nuevas configuraciones de forma fácil si así lo desea.

- ***Projectile:*** simula un cuerpo en general móvil o arrojadizo, de forma arbitraria (esférica, de lanza o de roca, etc.) y con un tamaño estándar.

- ***Area:*** figura circular que tiene la peculiaridad de recibir un bonus a su volumen a cambio de una reducción de su velocidad. Suele ser muy útil para aplicar una misma alteración a varias entidades cercanas entre sí.
- ***Self:*** esta forma actúa como *placeholder* de un efecto que se aplica directamente al usuario que instancia el hechizo. Por lo tanto, no se puede mover y su tamaño es cero.
- ***Zap:*** con un estilo similar a Projectile, esta estructura tiene la característica principal de impactar de manera inmediata con el blanco. Como contrapartida, tiene un rango limitado. Ideal para jugadores novatos (*easy to use*).
- ***Entity:*** a diferencia de las categorías anteriores, esta crea un NPC aliado que ayudará al jugador a superar sus retos.

Element

Un elemento define la composición física de un hechizo (de qué está hecho). Efectúan el mismo rol que aquellos presentes en la filosofía griega: agua, aire, fuego y tierra. Define algunos elementos visuales del mismo y repercute en la magnitud final del modificador en una entidad en función de su resistencia.

Effect

Es el resultado o el fin que produce un conjuro en la entidad que lo recibe. Tiene una magnitud que define su potencia y una duración finita. De manera similar a *Shape*, la clase supone el nivel superior de una jerarquía que define varios tipos.

- ***Life:*** afecta a la cantidad de vida del receptor. Nótese que la magnitud es un entero (positivo o negativo), por lo que este efecto puede sumar o restar vida.
- ***Resistance:*** modifica el grado de fortaleza del destinatario hacia un elemento. Este se determina por el mismo que contiene el hechizo. Un factor de resistencia inferior a 1 reduce la magnitud de todos los efectos de ese mismo elemento, mientras que uno superior la aumenta. Es importante percatarse que esto ocurre independientemente de si la finalidad del conjuro es positiva o negativa.
- ***Speed:*** este efecto aumenta o disminuye la velocidad de movimiento del actor.

Demeanour

Esta clase se encarga de definir el comportamiento físico del hechizo una vez este ha sido instanciado. En general, involucra a su velocidad y la trayectoria que realiza para alcanzar su blanco.

- ***Instant:*** se crea un conjuro en la posición indicada. Si no se le añade ninguna clase más, este se quedará estático hasta que se destruya.
- ***Chase:*** instancia el objeto justo delante del invocador. Justo después buscará y perseguirá al rival más cercano a él que esté dentro de su radio de acción.
- ***Impulse:*** en cierta manera similar a *Chase*, esta clase empuja al hechizo en cierta dirección fija hasta alcanzar velocidades elevadas.
- ***Fall:*** permite dejar caer un conjuro desde una determinada altura. Útil para alcanzar blancos que estén situados detrás de una pared, aunque no se puede usar en interiores.
- ***Child:*** vincula la forma al brujo, de modo que esta replica todos sus movimientos.

Impact

Con un parecido tangible a *Demeanour*, este componente determina la conducta de un hechizo justo al impactar con otro elemento de la escena, ya sea por ejemplo una entidad o un muro de piedra. Como muchas otras clases anteriores, constituye una jerarquía que le da flexibilidad a la hora de añadir nuevas funcionalidades.

- ***Destroy:*** temporizador básico que destruye el conjuro asociado después del tiempo especificado.
- ***Multiple:*** crea copias de sí mismo que permite multiplicar sus efectos si estas logran impactar en otras entidades. Para evitar un posible bucle infinito, las nuevas imitaciones no heredan este componente.
- ***Explosion:*** el hechizo explota al impactar, tratando de extender su efecto en un área grande a su alrededor con una magnitud reducida.
- ***Ricochet:*** da a la forma la capacidad de rebotar sobre diversos obstáculos hasta un máximo de ocasiones especificado por la propia clase.
- ***Infect:*** funcionalmente similar a *Explosion*, este componente permite extender los efectos del conjuro a varias entidades cercanas al punto de impacto. Tiene un alcance menor a cambio de una mayor efectividad.

CAPÍTULO 7: VISIÓN GENERAL DEL SISTEMA

Una vez vistos los requisitos del proyecto y el modelo conceptual, este apartado expondrá todas las partes que lo componen y sus roles que interpretan dentro del mismo. También mostrará cómo se integra la gramática especificada en el capítulo anterior en el FPP.

7.1. ARQUITECTURA DEL FPP

El proyecto ha utilizado el motor de juego Unity como base para montar el FPP. Esta elección tiene una repercusión importante a la hora de decidir la arquitectura, puesto que este *game engine* apuesta fuertemente por el diseño basado en componentes.

Según el manual del desarrollador [41], estos definen la conducta del objeto de la escena al cual están asociados. Por sí mismo, un *GameObject* no es capaz de realizar ninguna funcionalidad por sí mismo, sino que actúa como recipiente para los componentes. Por ejemplo, cualquier contenedor está obligado a tener un *Transform* (Figura 14), el elemento encargado de definir la posición, rotación y escala de un objeto en el ambiente.

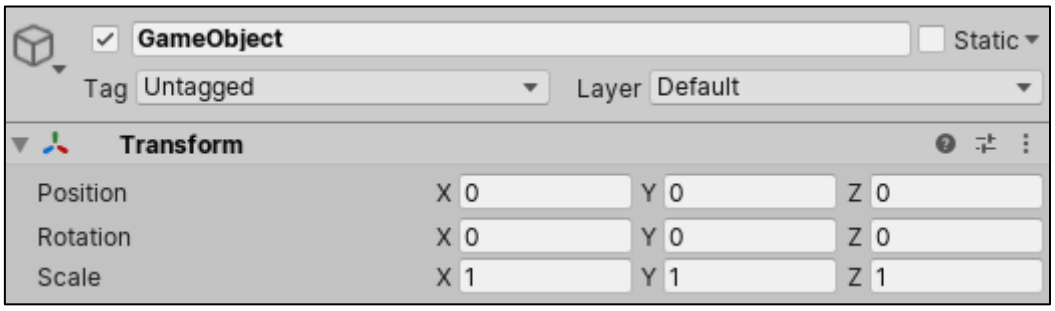


Figura 14: Componente Transform asociado a un GameObject. Fuente: [41]

Unity ofrece de manera predeterminada muchos componentes que realizan funciones comunes en todos los juegos. Cuerpos de colisión, emisores de luz y sonido o *renderers* que dibujan los triángulos que conforman los modelos son modelos de algunos de ellos. No obstante, por ellos mismos no son suficientes ya que no permite al desarrollador definir nuevos comportamientos personalizados. Por esta razón, el motor permite crear nuevas clases que actuarán como componentes siempre y cuando estas hereden de la clase base *MonoBehaviour* (que a su vez sucede a *Component*).

Dada esta característica, se ha decidido diseñar el código del videojuego entorno a una arquitectura en cuatro capas interconectadas entre ellas.

Visualización

Siendo el nivel superior de la jerarquía y suministrada por Unity, se encarga de mostrar al usuario información de su situación en el juego. Contiene también los elementos de la interfaz que comunican sus eventos a objetos de la siguiente capa inferior.

Componentes

Aquí se han definido todas las clases que están asociadas a algún *GameObject* de la escena. En la mayoría de casos, realizan sus acciones al inicio del programa mediante los métodos *Awake()* y *Start()*. No obstante, también son capaces de ejecutar código si reciben un evento. La naturaleza de este es muy variable, ya que puede ser generado por una colisión entre dos objetos o por el mismo jugador pulsando un botón. Tienen en común que todas sus clases heredan de *MonoBehaviour*.

Son componentes todos aquellos que determinan todas las mecánicas visibles por el usuario como el sistema de vida o la IA de los enemigos.

Lógica

Como capa intermedia en la arquitectura, en este nivel se localizan las clases que ejecutan procesos independientes que no necesitan información del *game engine* para funcionar. Tienen una estructura más tradicional (constructores, *getters*, *setters*, etc.) y son utilizadas por los componentes para delegar los procesos más complejos del videojuego. A diferencia de los componentes, todos los objetos de lógica no heredan de *MonoBehaviour*. El patrón ECS, explicado a continuación, no se aplica en esta capa.

En el proyecto en particular, aquí se encuentran todas las clases relacionadas con la implementación de la gramática, así como el código C# generado automáticamente por ANTLR.

Definiciones

Por último, en esta capa se ubican todos aquellos ficheros que, como ya indica su categoría, almacenan los datos del juego. Como característica principal, todas las clases que pertenecen aquí heredan del padre *ScriptableObject*. Gracias a ello, el desarrollador puede crear en la carpeta del juego ficheros de estadísticas que pueden ser importadas de forma fácil por otras clases. Las definiciones pueden ejecutar funciones, aunque no se pueden añadir a ningún *GameObject*.

Por ejemplo, en este último nivel se encuentra toda la información relevante al jugador y a los NPCs (vida, resistencias elementales y velocidad de movimiento, entre otros). También se encuentran los datos referentes a las propiedades de los conjuros como la magnitud o la duración.

Es importante mencionar que estas definiciones son, en ocasiones, referenciadas de forma directa desde algunos componentes. Esto se hace por rapidez y por sencillez ya que si se siguiera una estructura más rígida existirían clases puente que carecerían de funcionalidad. De esta manera, el proyecto sigue una arquitectura dividida en cuatro capas, una rígida (Visualización) y tres relajadas, representado en la Figura 15.

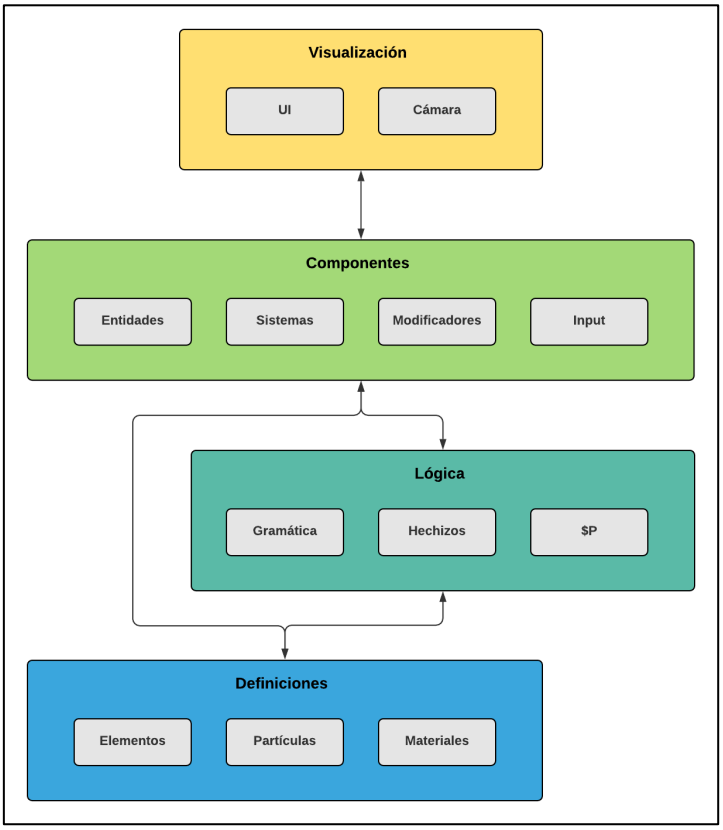


Figura 15: Diagrama de capas del sistema. Fuente: Elaboración propia.

7.2. ECS

El modelo ECS (*Entity – Component – System*) es un patrón arquitectónico muy utilizado en el desarrollo de videojuegos basado en el concepto de composición sobre herencia. Este principio se basa en la obtención de funcionalidades polimórficas mediante la inclusión de otras clases que contengan el cometido deseado, descartando herencias de un objeto padre. De esta manera se consigue un diseño muy flexible que favorece la reutilización del código [42].

El sistema de herencias no tiene porqué ser siempre la solución ideal. Si se quiere representar diferentes tipos de entidades, puede darse la situación de que ocurran hechos como el apreciable en la Figura 16. En ella, uno puede percatarse de la existencia de múltiples repeticiones de atributos. Además, en un caso más que probable, este hecho implicará la duplicidad de las funciones encargadas de tratar estos datos (por ejemplo, el método encargado de actualizar la vida).

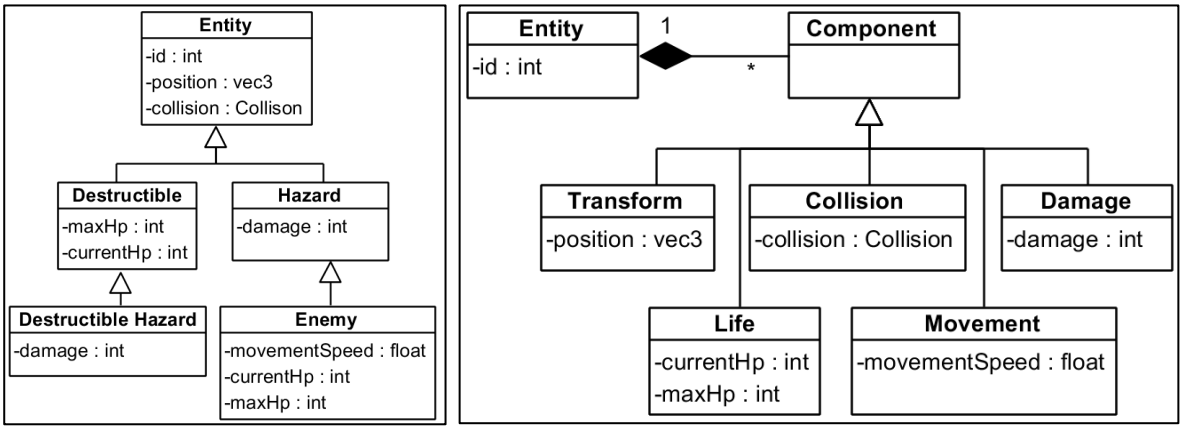


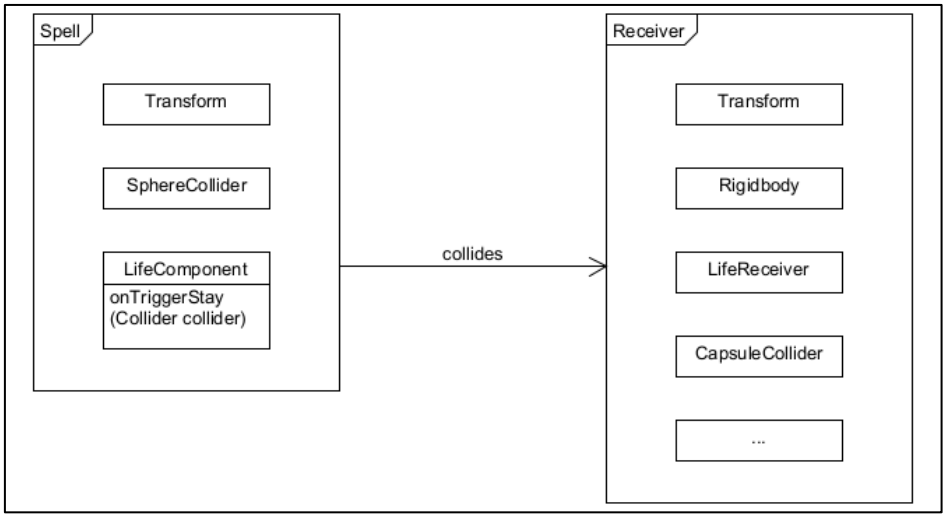
Figura 16: Ejemplo de polimorfismo mediante herencia (izquierda) y mediante componentes (derecha). Fuente: [43]

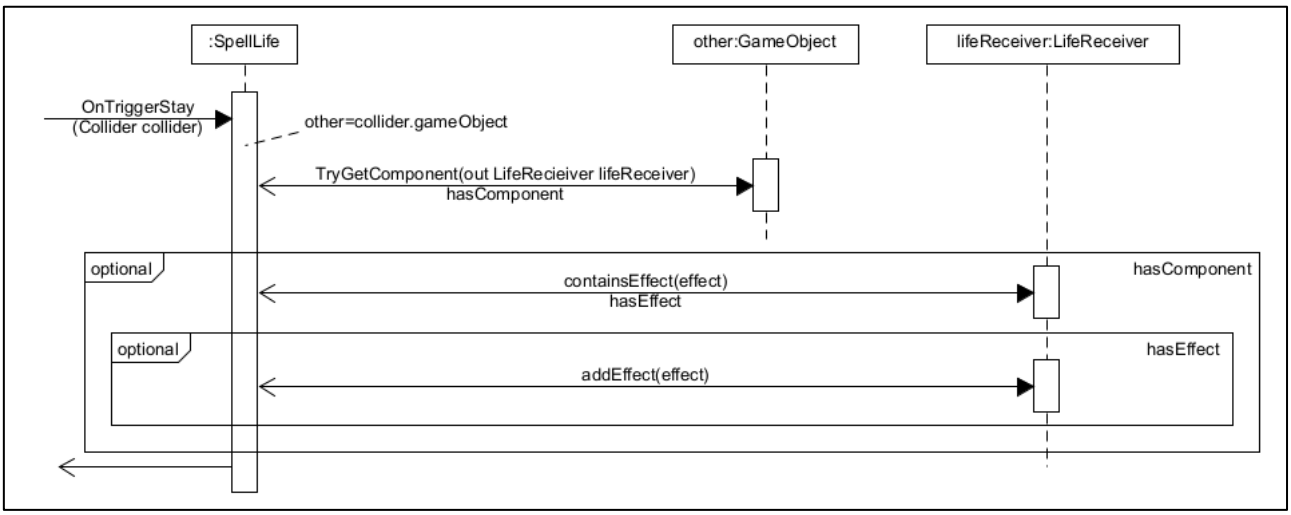
Como solución a este problema, el patrón ECS pretende montar un diseño que gira alrededor de tres estructuras fundamentales.

- **Entidades:** a este grupo se encuentran todos los objetos que pertenecen a la escena. Muchas veces, sólo almacenan su ID de referencia. En el proyecto de Unity, representarían los *GameObjects* que pueblan el entorno.
- **Componentes:** sus clases contienen la información e interacción de una característica concreta del videojuego. Establecen una relación de composición con las entidades que lo necesiten.
- **Sistemas:** independientes del resto, existe un sistema por cada tipo de componente que exista en el proyecto. Cada uno ejecutan acciones y cambios en todas las entidades que tengan este mismo asociado a ellas.

A fecha de redacción de este documento, Unity ya dispone de un *framework* ECS sobre el que realizar un videojuego. Apoyado sobre otras técnicas como el C# Job System o el último Unity Burst Compiler, el *game engine* es capaz de tratar de forma muy eficiente muchos más *GameObjects* de forma simultánea. Esto puede ser interesante, por ejemplo, en juegos de estrategia en tiempo real que deben de calcular las acciones de muchas unidades en batalla.

No obstante, todavía se encuentra en continuo desarrollo y no existe ninguna versión estable sobre la que producir un proyecto serio como un TFG. Aun así, ECS ofrece ventajas de reutilización de código muy interesantes para un sistema de magia modular como el que especifica este trabajo. Por ello, se ha diseñado una solución híbrida como la que se especifica en la Figura 17. En ella, se propone interpretar al sistema *LifeReceiver* como otro componente más. Cuando ocurra el evento que requiera una acción (en este caso, una colisión), este comprobará si la entidad contiene el componente que ha de tratar (Figura 18). En caso de que sea cierto, se realizarán los cambios que sean convenientes.





Figuras 17 y 18: Visión simplificada de la colisión de dos GameObjects distintos y diagrama de secuencia de OnTriggerStay(), respectivamente. Fuente: Elaboración propia

La arquitectura de tres capas descrita en el apartado anterior es totalmente compatible con la visión ECS y, combinados, darán al FPP unos buenos cimientos sobre los cuales construir el código que le dé vida.

7.3. PATRONES DE DISEÑO

Paralelo a la arquitectura, a lo largo del desarrollo se han ido encontrado numerosas situaciones que, como trabajo inscrito en Ingeniería del Software, se han resuelto mediante el uso de patrones de diseño. De esta manera, se consigue una solución fiable, eficaz y flexible a la hora de añadir código nuevo.

Patrón factoría

El modelo factoría provee un método para encapsular la creación de uno o más objetos relacionados sin tener que especificarlo.

Permite desvincular la clase que lo requiere y permite cambiar la instanciación al margen suyo. Esto es conveniente en la creación de objetos complejos, como es el caso de la clase *Spell* en la capa de Lógica.

Patrón visitante

Este método consiste en separar un algoritmo de la estructura de un objeto. Permite añadir nuevas funcionalidades sin modificar la estructura de los mismos.

El ejemplo más representativo del proyecto es la clase *SpellCreator* que hereda de *SpellCreatorBaseListener*, una de las clases autogeneradas por el plugin de ANTLR. En este caso, este patrón permite definir una función para tratar cada nodo del árbol generado por la gramática sin cambiar el código del mismo.

Patrón singleton (Variante de Unity)

Esta pauta permite a un desarrollador acceder directamente a la instancia de una clase siempre y cuando esta sea la única que existe en el contexto de ejecución. Es útil en particular en ocasiones donde un objeto esté encargado de coordinar las acciones de todo el sistema.

Su principal diferencia con cualquier otro singleton estándar radica en su inicialización. Al ser componentes (y por lo tanto hereda de *MonoBehaviour*), el objeto no puede instanciarse mediante el constructor. En su defecto, el patrón utiliza la función *Awake()* ejecutada al iniciar el programa. El resto de características y funcionalidades son iguales al modelo estándar.

En el TFG, se ha utilizado el singleton en clases como el *EntityManager* (que guarda las referencias de todas las entidades de la escena) o el *RuneRecognizer* (interpreta la entrada del usuario). Un ejemplo de su invocación lo podemos encontrar en *SpellChase* tal y como muestra el diagrama de la Figura 19.

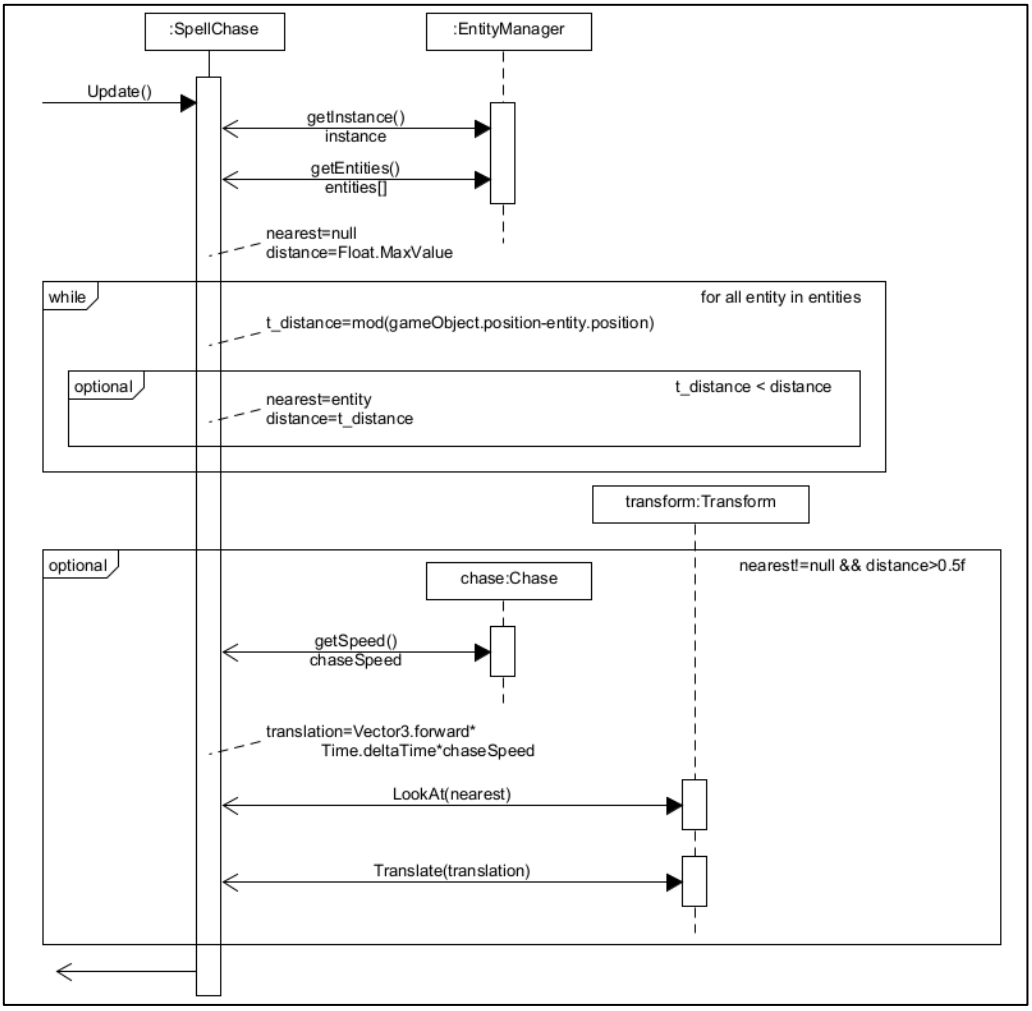


Figura 19: Patrón singleton. Fuente: Elaboración propia

Patrón observador

Por último, este modelo permite establecer una dependencia entre uno o muchos objetos. De esta forma, cuando uno cambia de estado o recibe un evento, este notifica al resto que están asociados a él.

Es, sin lugar a dudas, el patrón más utilizado en todo el proyecto. Es el encargado de cambiar los estados de los árboles de animación de las entidades del juego o de comunicar la colisión entre dos objetos a sus componentes. Esta acción es realizada de forma automática por Unity y el desarrollador puede decidir cómo la trata (Figura 18 del apartado anterior).

CAPÍTULO 8: IMPLEMENTACIÓN

Una vez llegados a este punto, este apartado tratará de exponer todos los detalles de la solución definida en los capítulos anteriores. Se conocerán los detalles de todas las tecnologías utilizadas, la gramática de ANTLR y el FPP. Dentro del tercero, se mostrarán las decisiones de desarrollo en todos los apartados (animaciones, partículas, sonido, etc.).

8.1. TECNOLOGÍAS UTILIZADAS

Unity

Como ya se ha adelantado en el apartado de Objetivos, el FPP se realizará mediante el motor gráfico Unity, en concreto en su versión 2020.2.1f1. Hay varios motores disponibles con licencia gratuita, entre los cuales destacan Unity y Unreal Engine. Aunque el segundo es realmente una opción igual de válida que el primero, al final se ha decidido escoger Unity principalmente por su mayor popularidad (y por lo tanto mayor facilidad a la hora de resolver problemas) y por el hecho de haber tenido ya algunas experiencias previas con este. El *game engine* también ofrece otras características importantes como la capacidad de poder testear el proyecto en Android de forma directa mediante depuración USB o la capacidad de monitorizar el rendimiento según las tareas realizadas entre fotograma y fotograma.

A la hora de trabajar con el motor de juego, se ha establecido una jerarquía específica de los *assets* o archivos utilizables en el proyecto con el objetivo de facilitar la navegación del entorno de trabajo. En concreto, se ha decidido estos ítems en diversas subcarpetas.

- **Arte:** en este directorio se guardarán todos los archivos relacionados con la parte visual del proyecto. Materiales, partículas, texturas o modelos son algunos ejemplos de ello. Cada uno de estos elementos se organizan en más subcarpetas en función de su categoría y/o objeto al cual pertenezcan.
- **Recursos:** en esta ruta se encuentran todos los datos de los distintos objetos del proyecto: estadísticas del jugador, símbolos predefinidos y la configuración de los sistemas de partículas, entre otros. Suelen ser instancias concretas de *Scriptable Objects* que veremos en detalle más adelante.

- **Scripts:** por último, en este directorio se encuentran todas las clases y jerarquías que determinan el funcionamiento del videojuego y de la gramática. Por su parte, scripts se divide fundamentalmente en más subcarpetas (componentes, lógica y definiciones) cuya especificación se realizará en un futuro apartado dedicado a la arquitectura.

Finalmente, a la hora de guardar las revisiones del proyecto, se ha decidido utilizar la herramienta ya integrada denominada Collab, que permite la compartición de código, *assets* (modelos, texturas, sonidos…), escenas y librerías entre múltiples personas. El plan gratuito admite hasta un máximo de tres personas y 2 GBs de almacenamiento que será suficiente para el desarrollo.

Figura 20: Logo de Unity.

ANTLR

ANTLR (o *ANother Tool for Language Recognition*) es un generador de *parsers* utilizado para leer, procesar, ejecutar o traducir un texto estructurado de entrada [44] a partir de una gramática previamente definida por el programador. Su funcionamiento es muy simple: el desarrollador determina la gramática en un archivo de extensión ".g4" y pasa el archivo por parámetro al ejecutable Java de ANTLR, disponible para la descarga en su página oficial. En concreto, se ha utilizado su versión 4.9.1. Este ejecutable se encarga de generar código en el lenguaje especificado (C#), capaz de integrarse fácilmente en nuestro proyecto.

Figura 21: Logo de ANTLR.

Visual Studio Code (VSC)

Aunque Unity ofrece soporte de forma nativa para realizar *debugging* con Visual Studio, se ha optado por utilizar en su lugar Visual Studio Code fundamentalmente por el plugin de soporte [45] para la gramática de ANTLR. Gracias a esta extensión, el IDE es capaz de reconocer errores de compilación en tiempo real y ofrecer recomendaciones de autocompletado que permiten hacer el trabajo más eficiente. Por otro lado, VSC también tiene disponible otro plugin para poder "debugar" el código en Unity [46], razón por la cual no se pierde ninguna funcionalidad.

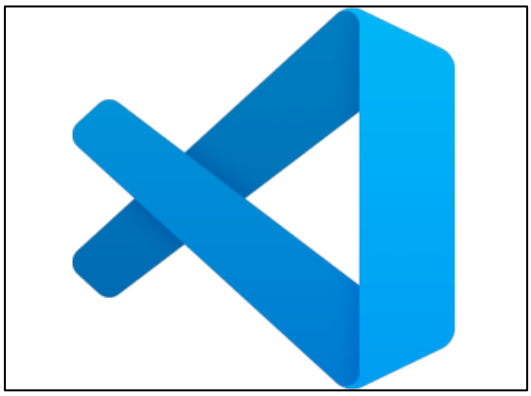


Figura 22: Logo de Visual Studio Code.

8.2. COMPONENTES DE LA GRAMÁTICA

Una vez realizada la configuración inicial del proyecto, el autor comenzó implementando los efectos mágicos en el proyecto de Unity. No se puede construir una casa (o un hechizo en este caso) sin tener primero los ladrillos con los que construirla. Por esta razón, este proceso significó la primera fase del trabajo.

Al empezar a desarrollar las funcionalidades, el equipo se dio cuenta que necesitaban información adicional para ciertas clases. El comportamiento de *Chase*, por ejemplo, no tiene ningún sentido si no tiene a alguien a quien perseguir. *Ricochet* necesitaba también un atributo adicional que se encargará de contar las ocasiones en las que el hechizo había rebotado. Sin embargo, no tenía mucho sentido incluir estas nuevas variables en el modelo conceptual puesto que no son relevantes a la hora de crear el hechizo.

Por esta razón, se tenía que diseñar una solución que fuera capaz de aceptar la inclusión de nuevos atributos sin entorpecer el esquema visto en el Capítulo 5 (Figura 13). Gracias a la arquitectura en tres capas y el modelo ECS, se desarrolló el modelo representado en la Figura 23.

En ella, se han separado los elementos fundamentales definidos en la gramática en un objeto perteneciente a la capa de lógica. Esta información es accesible desde el componente *SpellRicochet*, que incluye todo el código relacionado con la conducta del hechizo y sus variables auxiliares. Esta estructura se mantiene para el resto de *Effects*, *Demeanours* e *Impacts*.

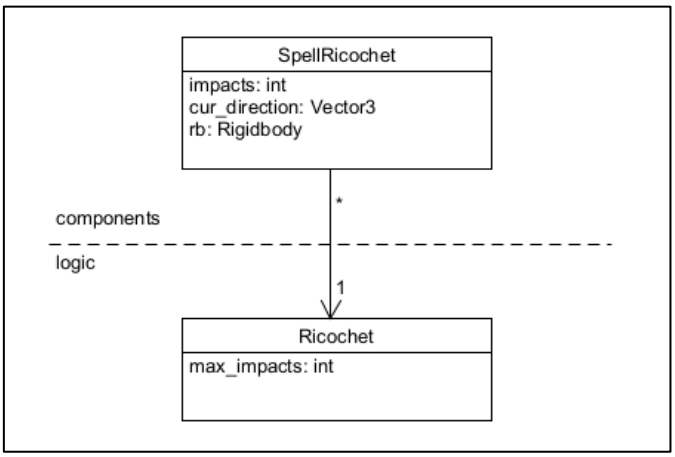


Figura 23: Implementación de Ricochet en el FPP. Fuente: Elaboración propia

Según la funcionalidad que implemente el componente, su código será invocado por el método *Update()* del que hemos hablado antes o desde toda función que marque un evento como *OnTriggerEnter()*. Sus diagramas de secuencia ya se han mostrado de forma previa en las Figuras 17 y 18 del Capítulo 7.

En cuanto a *Shapes* se refiere, estas no definen ningún tipo de comportamiento. Sus atributos sólo se utilizan para definir las propiedades del *GameObject* que se instancia al crear un hechizo. Algunos ejemplos de ellas son el tipo y tamaño de la malla de colisión, los efectos visuales e información sobre los componentes a agregar al objeto, entre otros. Por estas razones, es suficiente definir sus clases únicamente en la capa de Lógica.

8.3. HECHIZO COMO GAMEOBJECT

Como ya se ha adelantado, basándose en el esquema ECS, los conjuros representan una entidad en la escena cargada. Estos objetos actúan como contenedores de otras clases que determinan su conducta y el efecto que el hechizo produce al impactar con el resto de NPCs del mapa. Esto puede verse en la vista simplificada de la Figura 24. En la imagen se encuentra la información visible de los componentes que lo constituyen (*Transform*, *SpellLife*, *SpellImpulse*, etc.).

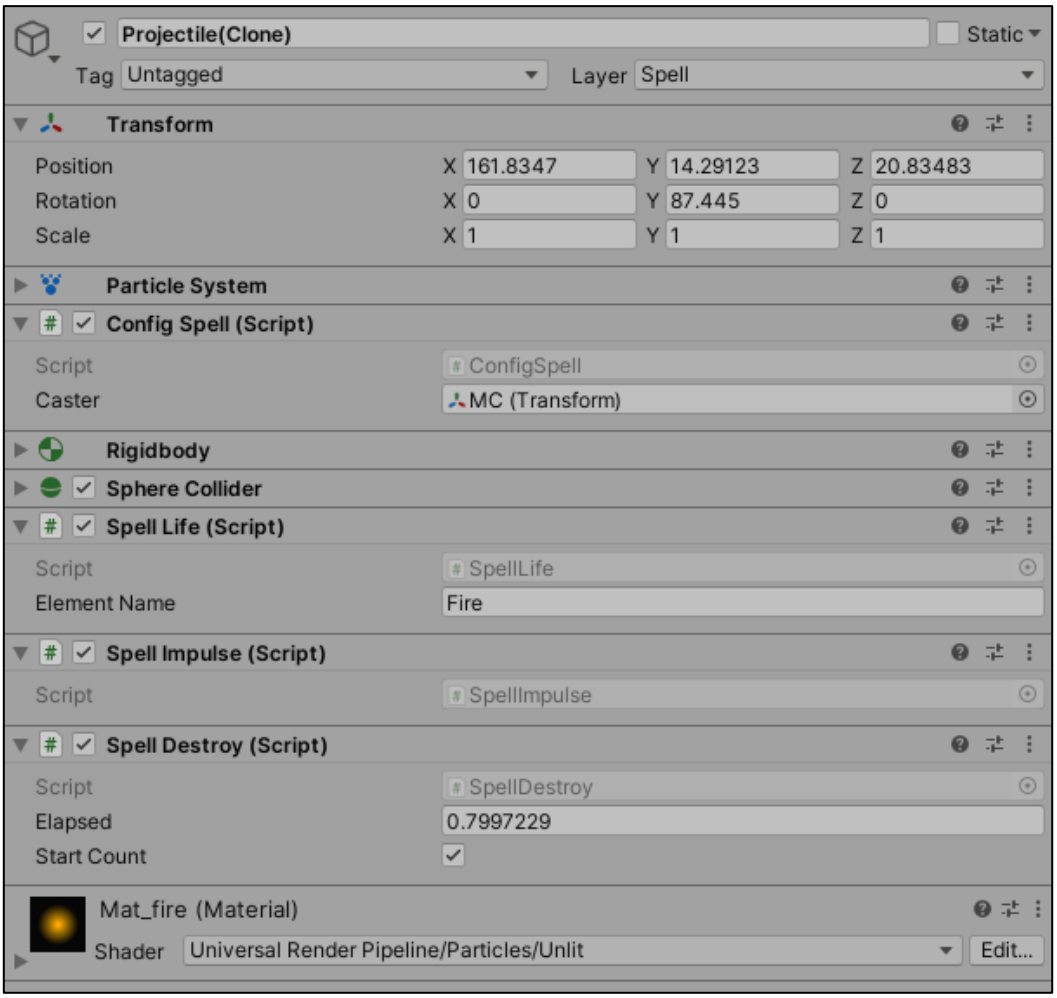


Figura 24: Visión de un hechizo en el editor de Unity. Fuente: Elaboración propia

A partir de este punto, el desarrollador ya tiene todas las piezas para crear el hechizo que le plazca. No obstante, si se considera lo que se ha implementado hasta este punto, el proceso sólo puede realizarse de forma manual. Aún falta por ver qué parte del sistema se encarga de generar los conjuros de manera automática.

8.4. GRAMÁTICA EN ANTLR

A partir de un texto de entrada y un conjunto de normas definido de forma previa, ANTLR es capaz de ordenar el input (en este caso, un hechizo) en un árbol jerárquico y generar el código de las clases encargadas de recorrerlo. La gramática se encuentra definida en el fichero contenido en Scripts/Logic llamado SpellCreator.g4.

Dentro de este archivo, se pueden encontrar dos tipos de declaraciones que hacen referencia a las diferentes tareas del proceso de interpretación [47].

Reglas del *Lexer*

Son aquellas que se encargan de encontrar unidades que contengan algún significado dentro de la entrada textual. También conocidas como *tokens*, estas normas se encargan de identificar y separar las palabras del hechizo. Realizan una función similar a la del análisis morfológico de una oración escrita en catalán, castellano o inglés.

Todos estos elementos fundamentales pueden verse en el *Snippet* 1 mostrado más adelante. En él, se pueden reconocer las palabras referentes a las distintas clases del modelo conceptual del Capítulo 6. Además, se han añadido una serie de *tokens* adicionales para representar las magnitudes de los distintos componentes para hacer del lenguaje algo más natural. Es más inmersivo para el jugador representar un proyectil como “MAJESTIC PROJECTILE” que mediante el uso de números “PROJECTILE 10”.

Reglas del *Parser*

En contraste, estas normas tienen la función de buscar y reconocer las relaciones que se establecen entre los tokens generados. Se ejecutan, por lo tanto, después de realizar el análisis léxico (*Lexer*). Su cometido es idéntico al del análisis sintáctico de frases escritas en cualquier lenguaje natural. Como tal, un sintagma es capaz de contener otros sintagmas.

Las reglas que reconoce la gramática del proyecto se exponen en el *Snippet* 2. ANTLR permite incluir varios sufijos para determinar la multiplicidad de cualquier fragmento. En este caso, el carácter “?” especifica que el token admite 0..1 instancias de sí mismo. De esta manera, el sistema puede seguir identificando hechizos y efectos sin la presencia de esa palabra. El objetivo final es crear una gramática adaptable a los escasos conocimientos de los jugadores en los momentos iniciales de juego.

Aunque la gramática de ANTLR permite añadir algunas funcionalidades en el mismo fichero de extensión “.g4”, se ha decidido realizar todo el tratamiento de la entrada en un fichero aparte. Con ello, se evita mezclar el tratamiento del hechizo con su dicha especificación, de forma que se obtiene una estructura final mucho más limpia y clara.

```
///// LEXER RULES /////
// ELEMENTS
FIRESOME: 'FIRESOME';
FROZEN: 'FROZEN';
STORMFUL: 'STORMFUL';

// EFFECTS
LIFE: 'LIFE';
RESISTANCE: 'RESISTANCE';
SPEED: 'SPEED';

// SHAPES
PROJECTILE: 'PROJECTILE';
AREA: 'AREA';
SELF: 'SELF';

// DEMEANOURS
INSTANT: 'INSTANT';
IMPULSE: 'IMPULSE';
CHASE: 'CHASE';
FALL: 'FALL';
PARENT: 'PARENT';

// IMPACTS
DESTROY: 'DESTROY';
RICOCHET: 'RICOCHET';
MULTIPLE: 'MULTIPLE';

// ADJECTIVES
fragment SMALL: ('PITTY' | 'TINY' | 'SPARKING' | 'WALKING');
fragment MEDIUM: ('COMMON' | 'LARGE' | 'ENDURING' | 'RUNNING');
fragment LARGE: ('GREAT' | 'MAJESTIC' | 'EVERLASTING' | 'FLYING');
ADJECTIVE: (SMALL | MEDIUM | LARGE);

// MAGNITUDE SIGN
fragment BUFF: 'BUFF';
fragment DEBUFF: 'DEBUFF';
SIGN: (BUFF | DEBUFF);

// AFFECTS CASTER?
WISE: 'WISE';

// NON FUNCTIONAL
WS: [ \t\r\n] -> skip;
CONNECTOR: ('IN' | 'AS' | 'AND');
```

Snippet 1: Reglas del Lexer de SpellCreator.g4. Fuente: Elaboración propia

```
///// PARSER RULES /////
// SPELL
spell: element shape EOF?;

// ELEMENTS
element: (fire | ice | shock) effect;
fire: FIRESOME;
ice: FROZEN;
shock: STORMFUL;

// EFFECTS
effect: (life | resistance | speed);
life: WISE? ADJECTIVE? ADJECTIVE? LIFE SIGN;
resistance: WISE? ADJECTIVE? ADJECTIVE? RESISTANCE SIGN;
speed: WISE? ADJECTIVE? ADJECTIVE? SPEED SIGN;

// SHAPES
shape: connector (projectile | area | self) demeanour? impact?;
projectile: ADJECTIVE? PROJECTILE;
area: ADJECTIVE? AREA;
self: SELF;

// DEMEANOURS
demeanour: connector (instant | impulse | chase | fall | parent);
instant: INSTANT;
impulse: ADJECTIVE? IMPULSE;
chase: ADJECTIVE? CHASE;
fall: ADJECTIVE? FALL;
parent: PARENT;

// IMPACTS
impact: connector (destroy | ricochet | multiple);
destroy: ADJECTIVE? DESTROY;
ricochet: ADJECTIVE? RICOCHET;
multiple: ADJECTIVE? MULTIPLE;

// OTHER
connector: CONNECTOR;
```

Snippet 2: Reglas del Parser de SpellCreator.g4. Fuente: Elaboración propia

8.5. CÓDIGO GENERADO POR ANTLR

Una vez definida la gramática, el desarrollador puede proceder a ejecutar el plugin de ANTLR para obtener las clases utilizadas para el procesado del árbol. Se han creado una serie de aliases para simplificar y acelerar el trabajo con este ejecutable Java.

- **antlr4:** produce el código utilizable en el proyecto. Considera por defecto la plataforma objetivo de Java, aunque es posible modificarla mediante el parámetro "-Dlanguage".

```
java -classpath "%ANTLR4_HOME%\antlr-4.9.1-complete.jar";. org.antlr.v4.Tool %*
```

Snippet 3: Comando completo del alias antlr4. Fuente: Elaboración propia

- **compile:** permite codificar las clases Java en archivos binarios producidas con el comando anterior. Este alias es prescindible para la ejecución del videojuego, pero es necesario para el siguiente paso.

```
javac -classpath "%ANTLR4_HOME%\antlr-4.9.1-complete.jar";. %*
```

Snippet 4: Comando completo del alias compile. Fuente: Elaboración propia

- **grun:** muestra en formato texto el AST para una regla y entrada específicas. La opción de ejecución "-gui" permite mostrarlo de forma más gráfica más fácil de comprender. Un posible resultado de este programa es el que se muestra en las Figuras 25 y 26.

```
java -classpath "%ANTLR4_HOME%\antlr-4.9.1-complete.jar";. org.antlr.v4.gui.TestRig %*
```

Snippet 5: Comando completo del alias grun. Fuente: Elaboración propia

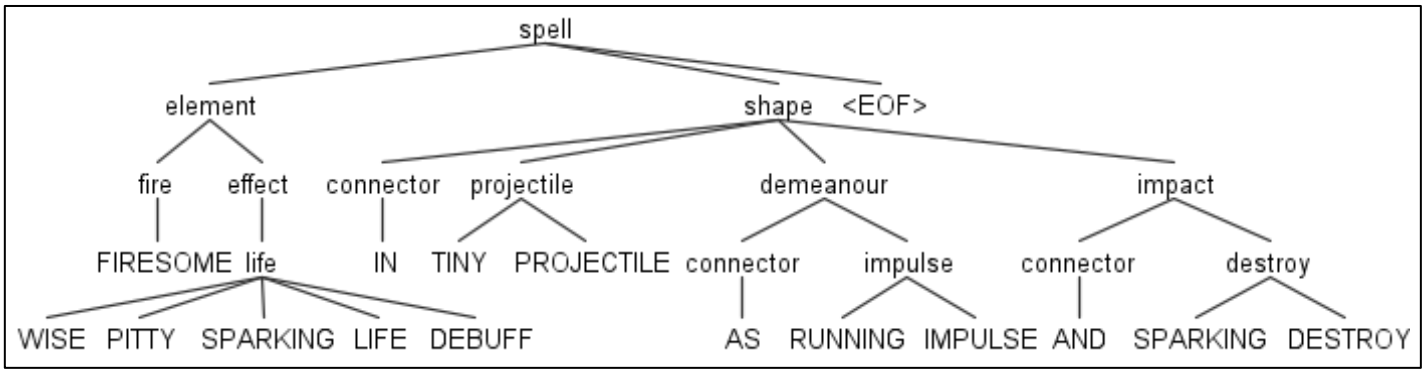


Figura 25: AST resultante de un hechizo. Fuente: Autogenerado por ANTLR

INPUT:

FIRESOME	WISE PITTY SPARKING LIFE DEBUFF	IN	TINY PROJECTILE	AS	RUNNING IMPULSE	AND	SPARKING DESTROY
fire	life	*	projectile	*	impulse	*	destroy
	effect				demeanour		impact
	element				shape		
			spell				

(*) = connector

Figura 26: Análisis morfosintáctico de un hechizo. Fuente: Elaboración propia

Cada vez que se ejecuta *antlr4*, se crean una serie de archivos necesarios para el intérprete. Entre todos ellos destacan dos interfaces que realizan la misma tarea (recorrer el árbol), aunque de formas algo distintas: el *Listener* y el *Visitor*. Su principal diferencia radica en la forma de recorrer los nodos: el primero se realiza de forma automática por ANTLR mientras que el segundo accede a los siguientes niveles con llamadas explícitas. Si se ignoran, se omitirá la rama del árbol correspondiente. El *Visitor* también es capaz de devolver resultados en sus métodos.

Aun así, en el proyecto se ha decidido utilizar el *Listener* (implementándolo la clase *SpellCreator*) debido a su buen rendimiento y facilidad de uso. La información se guarda en variables locales que se leen una vez procesado el AST. Para leer las reglas del *parser* que aparecen en el árbol (los nodos en minúscula de la Figura 25), la interfaz define dos métodos que se ejecutan a la entrada y salida de cada vértice respectivamente. El listado de estos se realiza en preorden, tal y como muestra la Figura 27. Para los nodos que no se necesiten leer (como es el caso de los conectores), la función que le corresponda puede dejarse vacía.

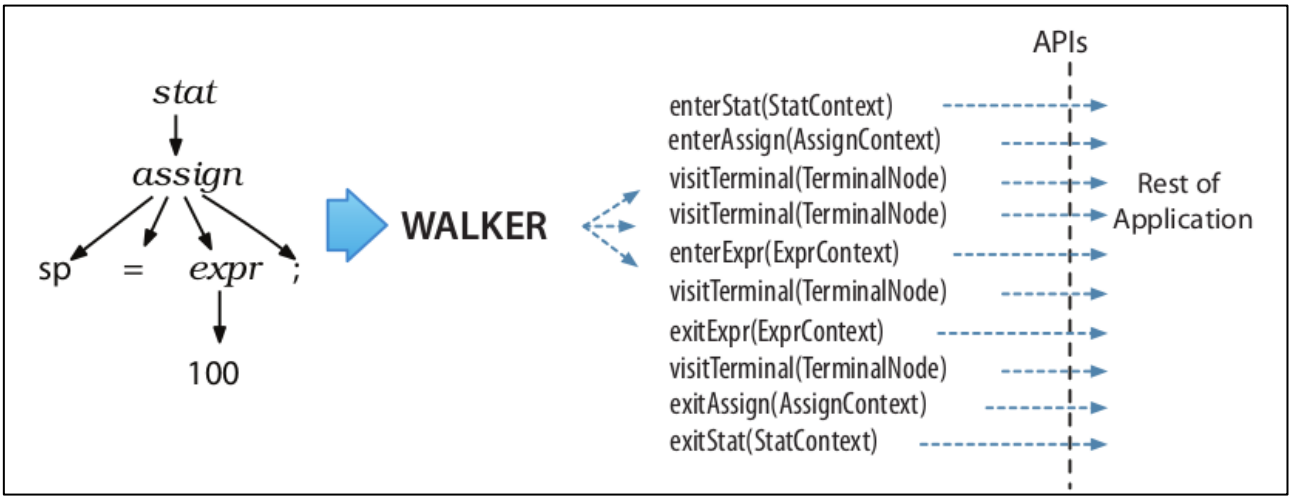


Figura 27: Lectura de un árbol en preorden. Fuente: [48]

La clase que implementa el *Listener* también contiene los diccionarios necesarios para traducir los adjetivos del hechizo a valores numéricos compatibles con los comportamientos ya creados. Es mejor realizar la traducción en *SpellCreator* en vez de en los constructores de los objetos puesto que de esta manera se evitan duplicidades.

8.6. LA FIGURA DEL SPELLCASTER

A partir de este punto, el TFG ya casi dispone de todas las herramientas necesarias para empezar a generar conjuros. Sólo falta crear una última clase desde dónde poder leer un hechizo y procesarlo mediante el Visitor creado en el apartado anterior.

Como su funcionalidad va a tener que ser invocada desde el FPP, el objeto SpellCaster debe de ser un componente. El diseño ECS permitiría generar magia a cualquier entidad que lo contenga. Por el momento sólo el *GameObject* que controla el jugador lo posee, aunque podría añadirse a NPCs si la IA pudiera soportarlo.

8.7. TRATAMIENTO DE EFECTOS

Como último aspecto del sistema de magia, esta necesitará transmitir sus modificadores, ya sean positivos o negativos, a otras entidades de la escena. En este punto de la implementación, el jugador puede crear multitud de hechizos, aunque este notará que ni él ni ningún personaje puede recibir sus efectos.

Para ello, se han diseñado nuevos componentes que actúan como sistemas asociados a los sujetos que reciben los efectos de los conjuros. En concreto, se ha añadido una nueva clase para cada modificador que existe en la gramática: *LifeReceiver*, *ResistanceReceiver* y *SpeedReceiver*. Todas ellas heredan del padre *BaseReceiver* una serie de atributos y funciones que comparten entre ellas. Sus valores se inicializan mediante una instancia específica para cada personaje de *EntityDef*, *ScriptableObject* perteneciente a la capa de definiciones.

Los efectos que reciben los sistemas se añaden a una lista propia de cada uno. Los valores del mismo se actualizan en cada segundo que pasa hasta que se agota la duración de cada efecto. Para evitar *polling* en cada *frame* (lo cual disminuiría el rendimiento del FPP), este proceso se aísla en una *coroutine* que aplica el modificador que corresponda y permanece dormida durante un segundo.

8.8. VISUAL DE UN HECHIZO

Una vez que ya se pueden generar conjuros, es necesario trabajar también su parte gráfica para tratar de hacer el sistema de magia lo más atractivo posible. En parte de ello depende la relevancia final que pueda causar el proyecto.

Debido a la escasez de tiempo y de recursos, el equipo desarrollador no ha podido permitirse crear una visual para cada combinación de efectos. En vez de ello, se ha concebido un sistema modular definido por las características de un hechizo. Más en concreto, el aspecto viene determinado por dos propiedades distintas.

- ***Element:*** almacenados en un *ScriptableObject* de clase *ElementDef*, este se encarga de describir el color del objeto. Un hechizo de fuego tendrá un aspecto rojizo en comparación al blanco característico del hielo.
- ***Shape:*** especificado por *ShapeDef*, el objeto guarda los datos relativos al modelo físico del conjuro y los emisores de partículas (tamaño, forma del foco, velocidad, etc.).
- ***Effect:*** contenido en la clase *EffectDef*, el *ScriptableObject* tiene una referencia al material (y la textura) utilizado como partícula. Por ejemplo, el efecto de vida contiene la imagen de un corazón monocromo (Figura 28).

Figura 28: Corazón monocromo del LifeEffect. Fuente: [49]

Gracias a toda esta información, el componente *SpellCreator* puede saber con exactitud las propiedades y componentes adicionales que ha de añadir al *GameObject*. El resultado todavía no es final ya que se mejorará considerablemente con los efectos lumínicos añadidos en el siguiente apartado.

8.9. GRIMORIO

Con el sistema ya montado, esta sección mostrará un listado de distintos ejemplos que el jugador puede especificar en el FPP implementado. En cada uno, se analizan sus características más destacables y el aspecto visual final.

Hechizo 1 (Figura 29)

"FROZEN SPEED DEBUFF IN PROJECTILE"

Características	
• **Elemento:** Hielo • **Efecto:** Velocidad • **Magnitud:** -50 • **Duración:** 5 segundos	• **Forma:** Proyectil de radio 2 metros • **Comportamiento:** Impulso de 15 metros por segundo • **Impacto:** Se destruye al instante

Figura 29: Visualización del Hechizo 1. Fuente: Elaboración propia

Hechizo 2 (Figura 30)

"STORMFUL GREAT EVERLASTING RESISTANCE BUFF IN SELF"

Características	
• **Elemento:** Tormenta • **Efecto:** Resistencia • **Magntitud:** 100 puntos • **Duración:** 10 segundos	• **Forma:** Aplicado a uno mismo • **Comportamiento:** Sigue al usuario • **Impacto:** Se destruye al instante

Figura 30: Visualización del Hechizo 2. Fuente: Elaboración propia

Hechizo 3 (Figura 31)

"FIRESOME WISE PITTY SPARKING LIFE DEBUFF IN TINY PROJECTILE AS RUNNING IMPULSE AND SPARKING RICOCHET"

Características	
• **Elemento:** Fuego • **Efecto:** Vida • **Magntitud:** -5 puntos (no afecta al jugador) • **Duración:** 1 segundo	• **Forma:** Proyectil de radio 1 metro • **Comportamiento:** Impulso de 15 metros por segundo • **Impacto:** Rebota un máximo de 2 impactos

Figura 31: Visualización del Hechizo 3. Fuente: Elaboración propia

Hechizo 3 (Figura 32)

"SHOCK RESISTANCE BUFF IN AREA AS CHASE AND EVERLASTING DESTROY"

Características	
• **Elemento:** Tormenta • **Efecto:** Resistencia • **Magntitud:** 50 puntos • **Duración:** 5 segundos	• **Forma:** Área de radio 5 metros • **Comportamiento:** Persigue a una entidad a 10 metros por segundo • **Impacto:** Se destruye a los 10 segundos

Figura 32: Visualización del Hechizo 3. Fuente: Elaboración propia

8.10. SISTEMA DE MAGIA

Para finalizar con el sistema de magia, este apartado incluye un esquema general de él y su listado de restricciones textuales que hay que respetar para asegurar la integridad correcta del *gameplay* del FPP.

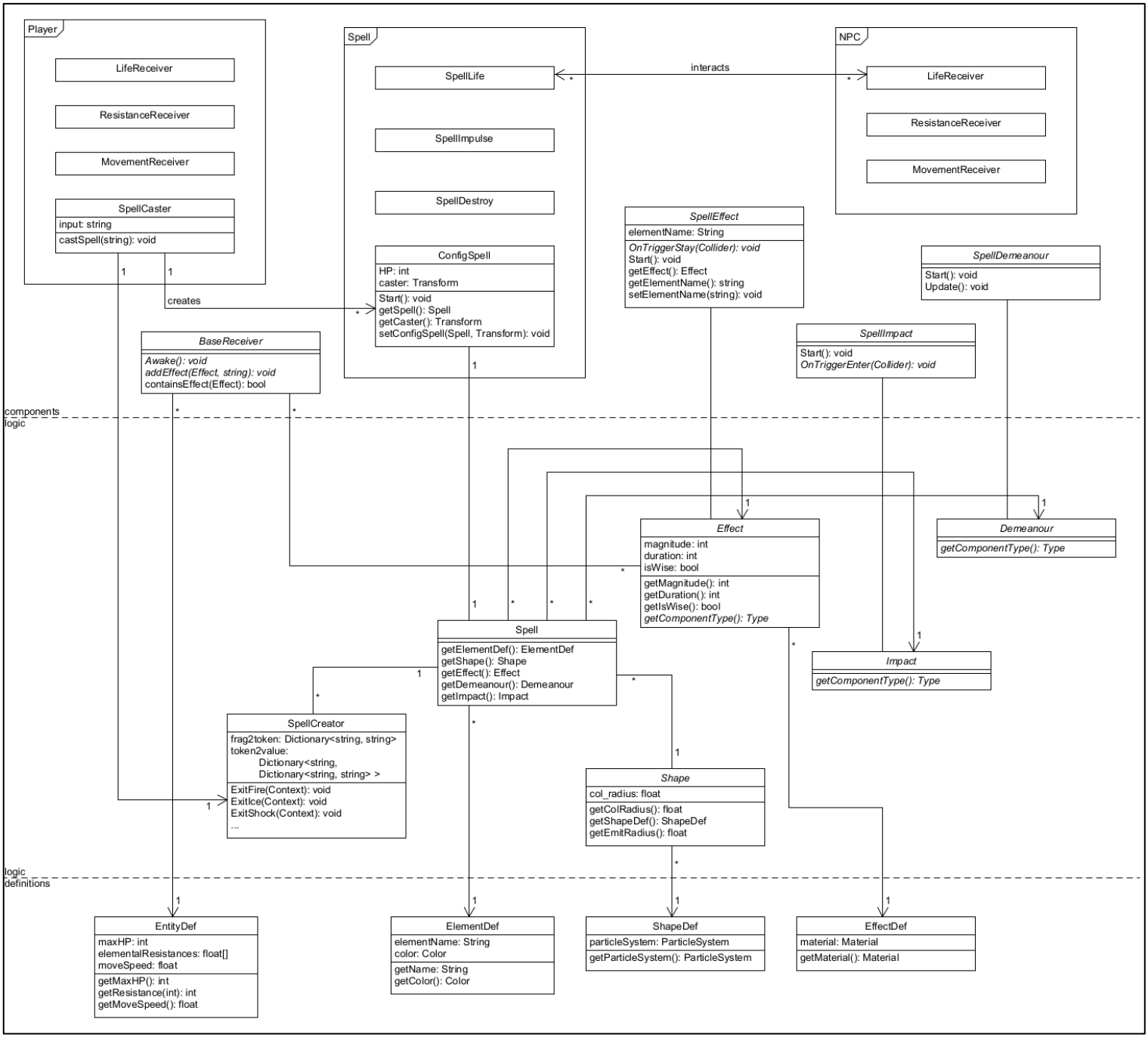


Figura 33: Diagrama de clases simplificado del sistema implementado. Fuente: Elaboración propia

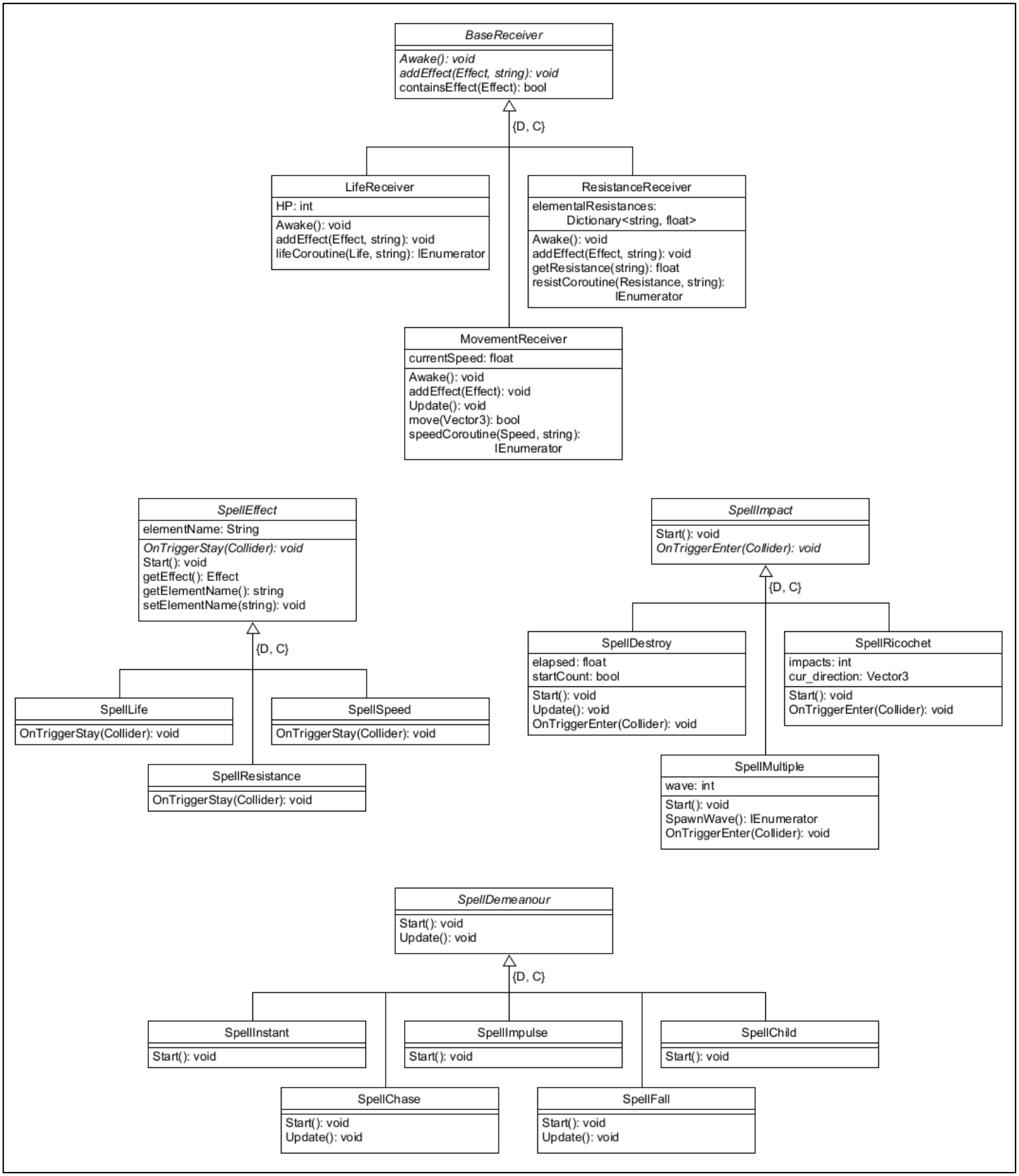


Figura 34: Modelo de jerarquías de componentes del sistema producido. Fuente: Elaboración propia

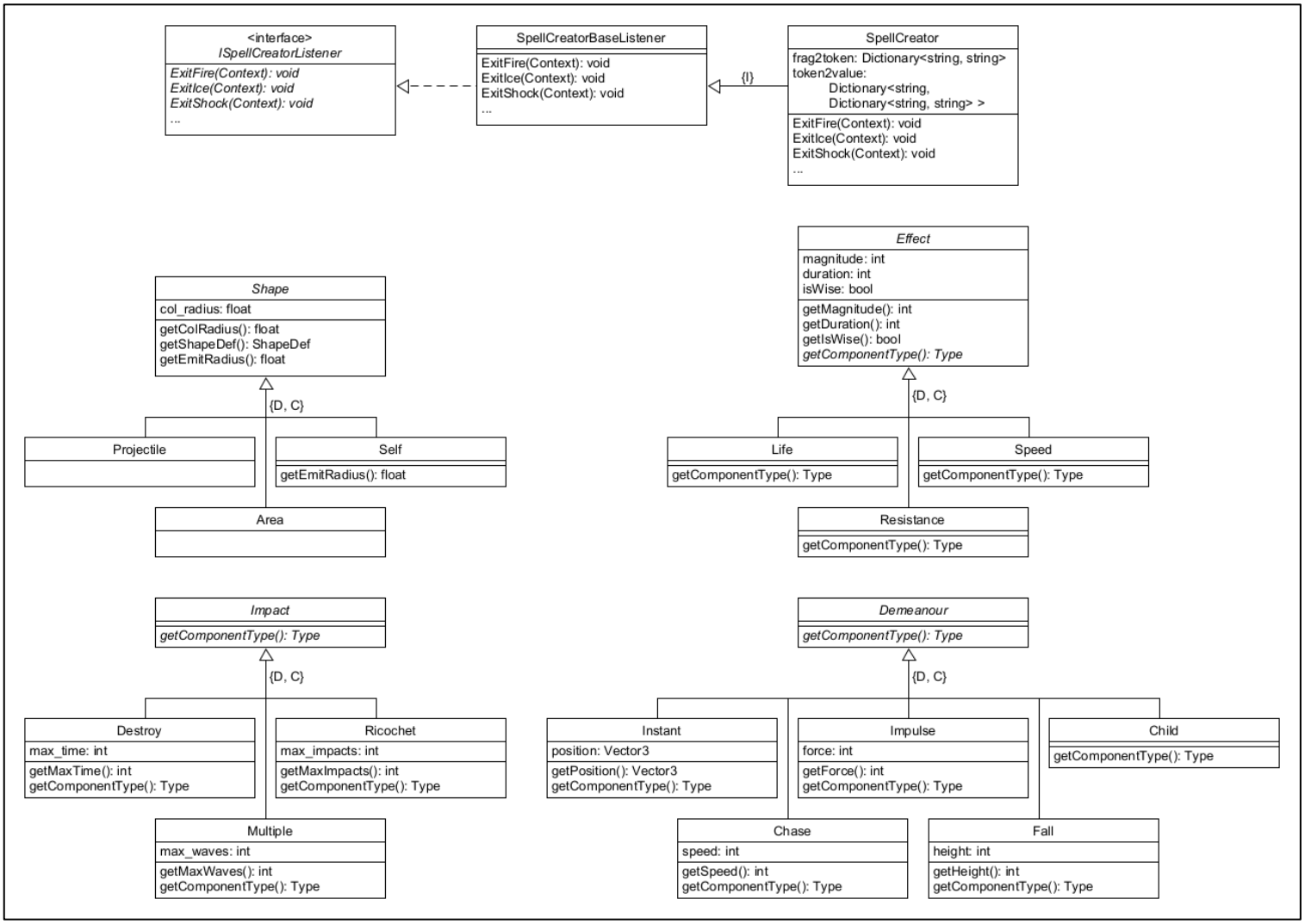


Figura 35: Modelo de jerarquías de lógica del sistema producido. Fuente: Elaboración propia

Restricciones textuales del sistema

- Toda instancia de *SpellEffect* debe de asociarse con el *Effect* que le corresponda. Por ejemplo, un *SpellLife* se relaciona con un efecto *Life*.
- Cualquier objeto *SpellImpact* se vincula con el *Impact* de su misma categoría.
- Las clases de *SpellDemeanour* tienen que ligarse a su *Demeanour* adecuado.
- Toda instancia de *BaseReceiver* responde a los *Effect* convenientes.
- El *Spell* de *SpellCreator* debe de coincidir con el de *SpellConfig*.

8.11. DIBUJADO DE RUNAS

Con la idea en mente de reducir la interfaz a su estado mínimo y ofrecer una UI minimalista y fácil de utilizar, el desarrollador pensó alternativas al clásico esquema de control basado en botones. Este punto es importante si se quiere abarcar un público objetivo más grande, puesto que un adulto tiene manos más grandes que un adolescente y no se adapta de la misma manera.

Es en este punto donde exponentes del estado del arte como Arx Fatalis o Eragon entran en consideración. Un sistema de dibujo de símbolos es ideal para el FPP implementado, puesto que el hardware ya ofrece los periféricos necesarios para introducir esta nueva mecánica. Como extra, también da una sensación aumentada de inmersión al jugador al tener que realizar una acción algo menos banal que pulsar un botón.

Pintar un símbolo en Unity es una funcionalidad bastante sencilla de programar (Figura 36). No obstante, el FPP necesita conocer qué runa ha dibujado el jugador para poder procesar la orden de generar un hechizo. Concebir un sistema de reconocimiento de símbolos es una tarea mucho más complicada, pero por suerte ya existen numerosos algoritmos eficientes que cumplen con esta función.

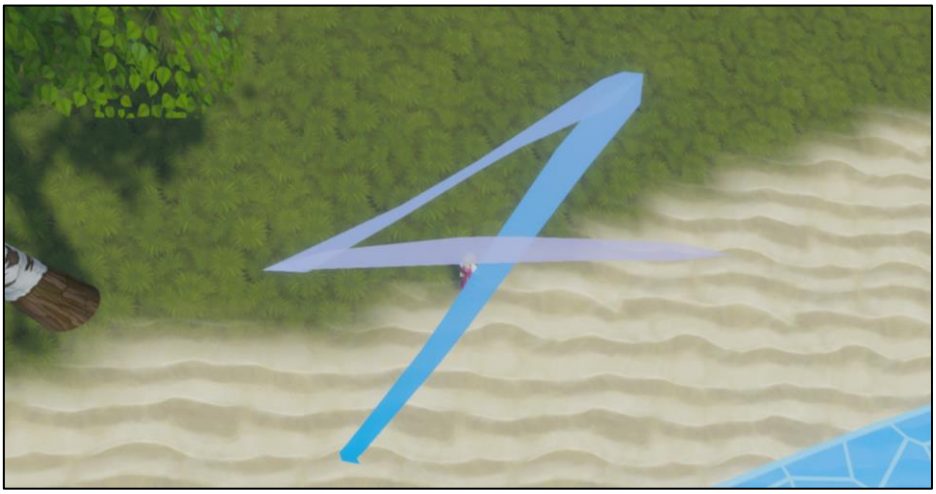


Figura 36: Dibujado de runas en el FPP. Fuente: Elaboración propia

Después de una pequeña investigación por parte del equipo, este encontró un conjunto de programas que satisfacen los requisitos: la *$-family*. Creada por Jacob O. Wobbrock de la University of Washington en colaboración con otros individuos [50], este grupo ofrece variaciones diversas de un algoritmo. Este se ha utilizado en proyectos con mercados totalmente distintos como videojuegos, smartwatches o incluso drones. Una vez examinado, se pueden destacar las tres distinciones siguientes:

- **$1:** como primera iteración publicada en el año 2007, esta distinción sólo permite reconocer gestos de un único trazado. Es la más sencilla de las tres, aunque no goza de las mejoras de rendimiento presentes en las otras variantes.
- **$P:** a diferencia de su antecesor ($N), este algoritmo es capaz de identificar símbolos con múltiples trazados. Es algo más eficiente puesto que representa los dibujos como una nube desordenada de puntos, hecho que reduce el número de datos a leer y escribir.
- **$Q:** basado en $P, esta iteración permite obtener rendimientos con hasta un *speedup* de 142 veces superior. Es ideal para dispositivos con prestaciones limitadas.

Los autores de la *$-family* ofrecen el código fuente en C# y JavaScript. No obstante, otros desarrolladores han realizado implementaciones propias en otros lenguajes y entornos. De todas ellas, destaca la desarrollada por "Da Viking Code" [51], una adaptación del algoritmo $P para el motor de Unity. Esto es ideal, puesto que permite a cualquier programador acomodar la versión de forma rápida a su proyecto.

Todas las clases importadas de este *asset* compone un *framework* que pertenece a la capa de Lógica, ya que ninguna hereda de *MonoBehaviour*. Existe un componente *PlayerInput* personalizado que recoge los puntos del trazado dibujado por el jugador que envía al nuevo singleton *RuneRecognizer*. Como su nombre ya indica, este trata de identificar el gesto entre todos los almacenados en varios archivos de extensión XML. Devolverá aquel que más se le parezca y se utilizará para crear el hechizo correspondiente en el *SpellCreator*.

8.12. ILUMINACIÓN

La elección del alumbrado del FPP es muy importante puesto que influencia en su estética final y, en especial, su rendimiento. Hay que recordar que este se ejecutará en dispositivos móviles, con una duración de la batería y potencia gráfica limitadas.

La escena se encuentra iluminada por una luz direccional de color blanco puro (255, 255, 255). Esta es estática, es decir, nunca cambiará su posición o rotación. Aunque a priori esta decisión puede parecer controvertida, esta característica es muy útil para ahorrar energía en el móvil, ya que da la posibilidad de crear sombras mixtas. Este modo de renderizado permite a Unity calcular umbras y penumbras de dos modos distintos.

- ***Realtime:*** como ya indica el propio nombre, este método calcula en tiempo real las sombras que producen los distintos elementos en la escena. Este es el tipo que se ha asignado a todos los elementos dinámicos en movimiento como el jugador o los NPCs de la escena. También hay configurados otros objetos como los hechizos para no tenerse en cuenta en el proceso de iluminación.
- ***Baked:*** muy utilizado en juegos antiguos y en dispositivos móviles, este procedimiento precalcula las sombras en el procesador y guarda su resultado en una textura conocida como *lightmap*. El trabajo se realiza en el ordenador donde se encuentra el editor de Unity y es una tarea que lleva algo de tiempo (en el caso particular del FPP, de dos a cinco minutos). No obstante, permite acelerar su ejecución en la plataforma objetivo final. El modo *baked* se utiliza en objetos estáticos que siempre emitirán la misma umbra.

El tamaño del *lightmap* está limitado en ambos casos a una textura cuadrada de tamaño 512x512 píxeles, para ahorrar espacio en memoria. El resultado final (Figura 37) es algo borroso si se mira de cerca, cosa que no ocurrirá puesto que el juego cuenta con una cámara situada encima del personaje principal (de estilo *top-down*).

No obstante, uno puede notar que la imagen es algo apagada. Para un videojuego de fantasía como el que se quiere implementar, es necesario avivar un poco la imagen. Esto se consigue mediante filtros de postprocesado. Unity ya dispone de algunos efectos configurables, entre los cuales se pueden destacar los siguientes.

- ***Bloom:*** afecta a los objetos emisores que hace destacar sus bordes dando la sensación de una luz muy brillante. Es perfecto para hacer resaltar los hechizos del jugador.
- ***Tone Mapping:*** es el proceso encargado de realizar la conversión de una imagen codificada en 16-bits (HDR o *High Dynamic Range*) a 8-bits (LDR) compatible con la mayoría de paneles.
- ***Color Adjustments:*** permite modificar propiedades básicas de una imagen, como el contraste, la saturación o la exposición a la luz.
- ***FXAA:*** algoritmo de *anti-aliasing* que elimina los posibles bordes de sierra que produzcan los objetos en la imagen final al renderizar en una resolución baja.

En la Figura 37 se pueden ver también los efectos que produce el postprocesado en la misma imagen.

Figura 37: Resultado de imagen final. Fuente: Elaboración propia

8.13. MOVIMIENTO Y ANIMACIONES

Como requisito del proyecto, es necesario programar un sistema que permita al jugador desplazarse por el escenario. Debido a la naturaleza humanoide del protagonista, también hay que producir un grafo de estados que reproduzca la animación que corresponda por cada una de sus acciones.

En cuanto al movimiento se refiere, de forma inicial se planteó realizar los movimientos en línea recta. El usuario pulsaba la posición final a la cual quería llegar y el sistema calculaba el vector de dirección correspondiente. Aunque de implementación simple, esta solución servía para la gran mayoría de casos. No obstante, era incapaz de reconocer obstáculos como paredes y NPCs. El usuario se veía entonces obligado a calcular su propia trayectoria de forma constante.

Para evitar este problema, se decidió utilizar un sistema algo más práctico. En la versión actual del TFG, se utilizan los compontes de *Navigation* ofrecidos por el propio Unity. Gracias a ellos, uno puede definir cualquier entidad no estática como un agente. Estos pueden desplazarse a través de una malla en el terreno precalculada que define sus zonas accesibles. La red es capaz de calcular el camino más rápido hasta una posición, teniendo en cuenta todos los obstáculos.

El proceso que inicia el movimiento se encuentra en la clase *MovementReceiver*, disponible en todos los personajes del FPP que contengan un *NavigationAgent*. Gracias al diseño basado en componentes, el proyecto puede asignar la misma función a todas las entidades, reaprovechando así el código ya realizado.

Por otro lado, es necesario incluir una serie de animaciones para dar una sensación de realismo al personaje. Realizar animaciones bien hechas es un trabajo costoso y requiere una virtud que no todo el mundo dispone, por desgracia. Sin embargo, por suerte existen numerosos archivos en la red con permisos abiertos que cualquiera puede utilizar. Un ejemplo de ello es Mixamo [52], web del equipo de Adobe que contiene *motions* de gran calidad.

Sin embargo, hacer que un personaje baile (por ejemplo) no es tan sencillo como bajar un fichero e insertarlo directamente en el modelo tridimensional. Toda animación basa sus movimientos en un esqueleto, formado por un conjunto de nodos y huesos a los cuales se asocian los vértices del objeto. Si este mueve su brazo derecho, todos los triángulos asociados a él recibirán las mismas transformaciones geométricas (translaciones, rotaciones y escalados).

Mixamo ofrece un método automático que permite, en pocos minutos, generar el *rig* de cualquier personaje humanoide. Este es además compatible con todas las animaciones disponibles en la web. No obstante, el esqueleto no es perfecto y requiere pequeños ajustes. Como se puede ver en la Figura 38, Blender admite realizar estas modificaciones y exportarlo en un nuevo modelo 3D en formato FBX estándar en la industria.

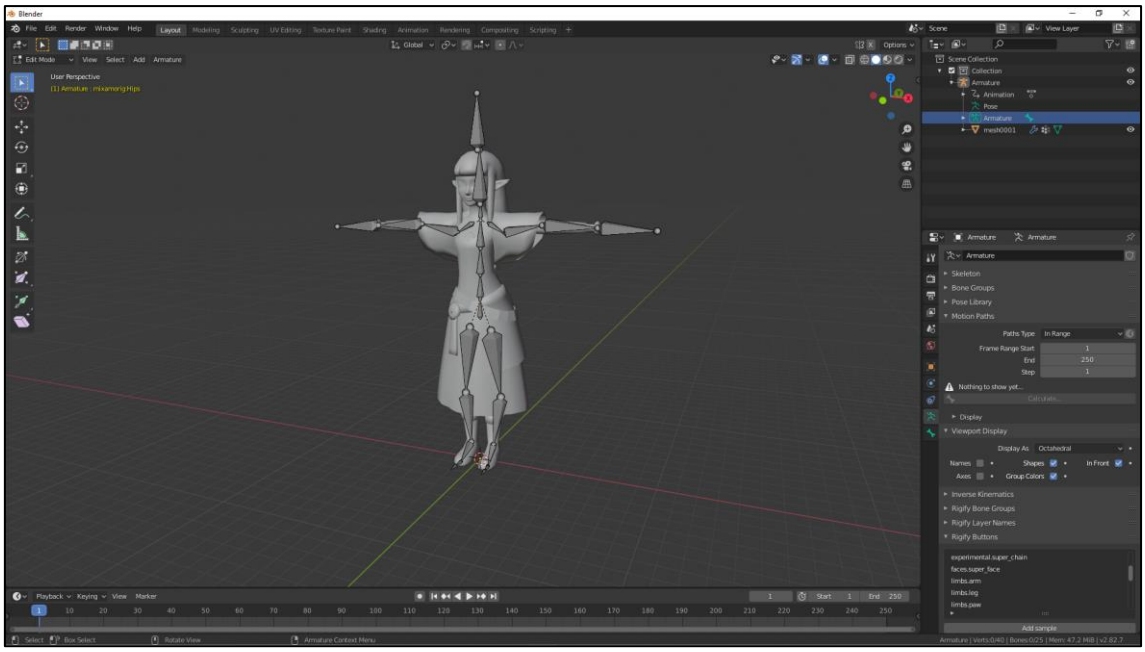


Figura 38: Modelo 3D del protagonista y su esqueleto cargado en Blender. Fuente: Elaboración propia

Una vez producido, el desarrollador ya puede proceder a realizar el grafo de animaciones. Es similar al esquema de una máquina de estados, donde cada uno de ellos representa un *motion* ejecutado en bucle. Si se produce un evento controlado por el sistema, este cambiará de animación y esperará a que ocurra otro acontecimiento de nuevo. Cambiar una animación a otra puede ser algo brusco, pero puede minimizarse mediante el uso de métodos de interpolación.

En el proyecto, todos los sucesos del personaje principal se controlan mediante el componente *HumanoidVisual*, sincronizado a su vez con todos los *Receivers* que dispone. Implementar una única clase de animación para cualquier entidad es difícil por el gran rango de animaciones y esqueletos que pueden llegar a contener. Es por ello que se han creado tantas como *rigs* que existan en el FPP.

8.14. SONIDO

Implementar un sistema de sonidos y de música (aunque sea básico) en el videojuego es fundamental para dar al usuario la sensación de acción y de ambiente. De manera similar al cine, el audio es capaz de transmitir impresiones e incluso sentimientos más poderosos que la propia imagen.

Para que un objeto de la escena pueda generar ruido, el desarrollador debe añadirle el componente *AudioSource* provisto por Unity. Dentro del mismo se pueden modificar varias opciones como el sonido a reproducir, su volumen o la capacidad de repetirse una vez agotado el tiempo de la pista de audio. Todos los parámetros configurables pueden verse a continuación en la Figura 39.

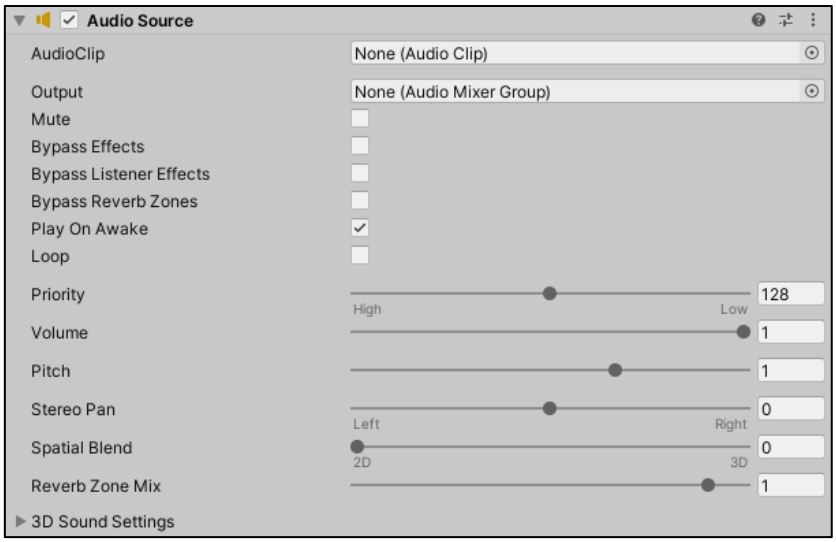


Figura 39: Componente AudioSource por defecto. Fuente: Elaboración propia

Entre todos ellos, es importante destacar el *Output* del componente. Por defecto, el sonido se transmite de forma directa al receptor del jugador que lo escuchará. No obstante, es posible tratarlo antes dentro de un *AudioMixer*. En él, el desarrollador puede añadir ciertos efectos sonoros como un filtro pasa bajos, eco o de distorsión.

Todas las fuentes asociadas a un mismo *AudioMixer* recibirán los mismos modificadores. Esta característica se aprovecha en el menú principal, lugar donde el jugador puede ajustar el volumen por separado de los distintos grupos de audio (General, Música y Efectos).

Aunque hay sonidos que se reproducen de forma continua, existen otros (pasos, creación e impacto de hechizos) que deben de ejecutarse en momentos específicos. Es por ello que ha sido necesario incluir un componente personalizado que se encargue de manejar todos estos eventos y active su *AudioSource* en el momento oportuno.

La clase implementa varios métodos públicos que cualquier otro objeto asociado a una entidad es capaz de utilizar. El archivo de audio puede estar asociado al componente de forma previa o bien está referenciado en un *ScriptableObject* de un hechizo.

8.15. INTELIGENCIA ARTIFICIAL

Para dar al proyecto una sensación de interactividad mínima con el entorno, es necesario que las distintas entidades que lo habiten se comporten en función de lo que haya a su alrededor como haría un animal o un ser humano. El jugador ya controla al protagonista, por lo que sólo haría falta desarrollar una IA para cada tipo de NPC.

Existen numerosas técnicas que permiten simular comportamientos muy avanzados, como redes neuronales o máquinas de estado finitas (FSM) basadas en el algoritmo MCST (Monte Carlo Search Tree). Generar una IA compleja no es el objetivo principal del TFG, pero sí es importante añadir un grado de interactividad mínimo.

Para ello, se ha implementado una máquina de Moore para cada entidad con una estructura similar a las que pueden encontrarse en videojuegos de la década de los 90. Su esquema puede verse a continuación en la Figura 40.

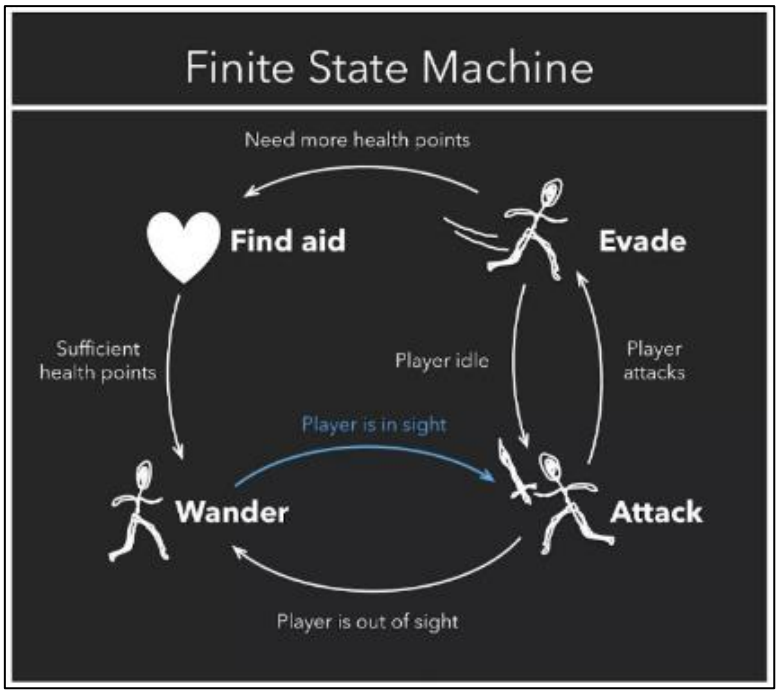


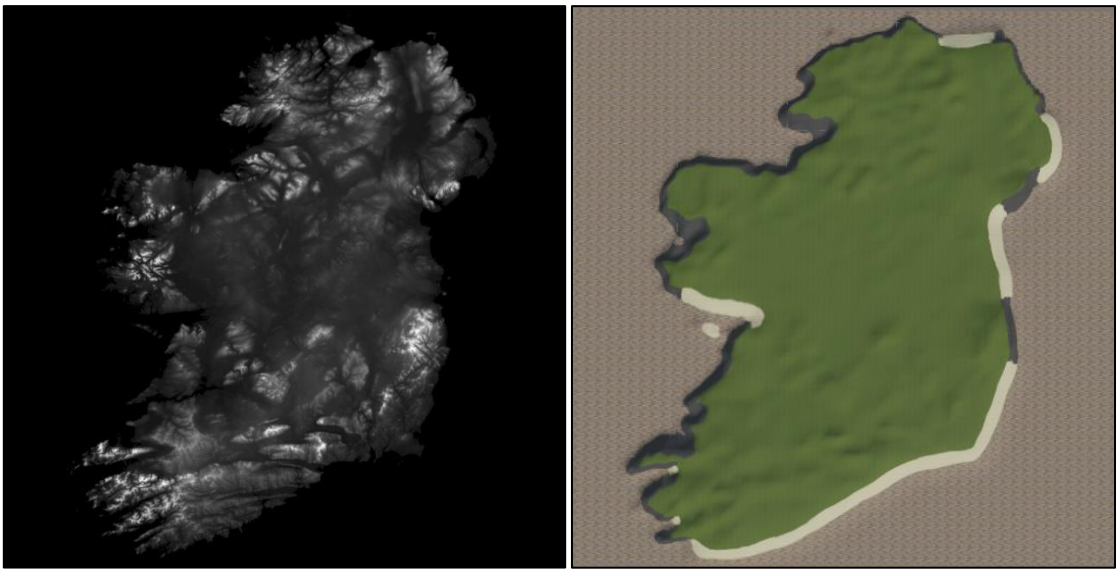
Figura 40: Ejemplo de máquina de estados finita. Fuente: [53]

8.16. ESCENARIO Y MODELADO

Para acabar con la implementación, en este apartado se comentará el diseño de la escena principal, sus optimizaciones y la justificación de la elección de los modelos tridimensionales utilizados. El objetivo final es presentar al jugador un producto coherente donde ningún elemento destaque por estar fuera de lugar.

Al crear el mapa, el desarrollador se ha basado en la idea de una isla desierta poblada por distintas criaturas. De esta manera, se puede limitar el área accesible por el jugador de forma sencilla. También es fácil incluir variedad en los elementos geográficos del terreno como colinas, playas u oleaje, entre otros.

Para ello, Unity permite crear orografías gracias a su objeto especial *Terrain*. Esto se realiza mediante el uso de distintos pinceles que elevan, suavizan y pintan su superficie (plana de forma original). Además, ofrece otras opciones configurables como su tamaño, la distancia máxima de render de sus árboles o la capacidad de importar y exportar el mapa en formato de *heightmap*, apreciable en la Figuras 41 y 42. Como se puede observar, se ha tomado como referencia una textura de la geografía de la isla de Irlanda. El mapa ha sido reescalado y alisado para que el jugador pueda acceder a todas las localizaciones en poco tiempo.



Figuras 41 y 42: Uso del heightmap de la isla de Irlanda en el terreno del FPP. Fuente: [54]

Como medida de optimización principal, se ha aprovechado la perspectiva top-down del videojuego para permitir no renderizar el terreno no visible por el usuario, conocida en Unity como *Occlusion Culling*. Gracias a ella, no ha sido necesario implementar otras técnicas más costosas (en tiempo y recursos) como LODs. Una demostración puede observarse en la Figura 43 y 44, que muestra la pequeña área del mapa que está dibujando.

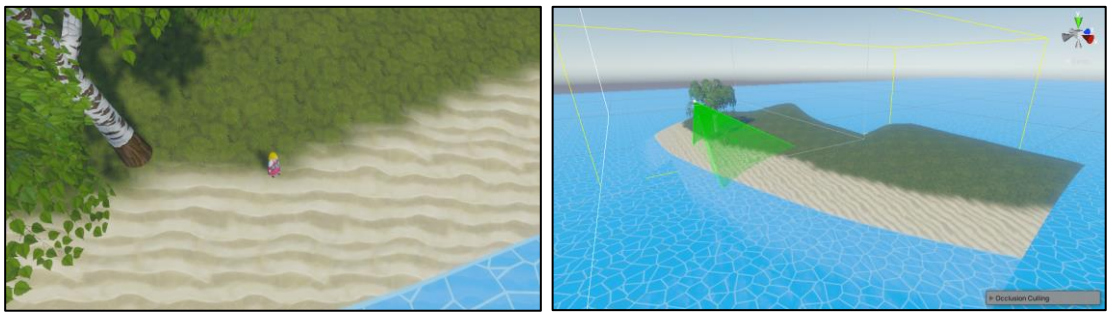


Figura 43 y 44: Efecto del occlusion culling en la escena. Fuente: Elaboración propia

Los más observadores también habrán notado la inclusión del océano que hay debajo del terreno. Aparte de limitar la zona explorable de la escena, en el ámbito visual el agua trata de imitar la estética animada del resto de texturas. Para lograrlo, el plano que lo conforma repite múltiples veces una misma textura basada en la función de ruido Voronoi. Su color, transparencia y amplitud de las olas son algunos de los parámetros del *shader* propio utilizado, producido gracias a la extensión de Unity Shader Graph. Gracias a él ha sido posible programarlo sin saber ningún lenguaje de programación de gráficos como GLSL o HLSL utilizados en OpenGL y DirectX respectivamente.

En cuanto a los modelos se refiere, uno debe tener en cuenta la plataforma objetivo del proyecto. Si se tiene en cuenta que se quiere realizar la demostración en dispositivos Android, el equipo ha tratado de evitar modelos y texturas muy pesados. Al realizar la selección, se establecieron los límites presentes en la Tabla 1 en función del *asset* importado.

Entidad	Número máximo de polígonos	Resolución máxima de textura
Protagonista	5000	256
NPC	2500	256
Árbol	1500	512
Terreno	-	1024
Partícula	-	128

Tabla 1: Detalle máximo permitido de los modelos y texturas de los distintos ítems de la escena. Fuente: Elaboración propia.

Si el candidato cumple con los límites descritos y coincide con el estilo caricaturesco del FPP, entonces será válido para ser utilizado en el proyecto.

CAPÍTULO 9: PLANIFICACIÓN

9.1. DESCRIPCIÓN DE LAS TAREAS

En el siguiente apartado se enumeran las diversas tareas que componen el proyecto junto a una pequeña descripción de su contenido y una aproximación del tiempo que han requerido. Para facilitar su lectura, los diferentes cometidos han sido agrupados en dos categorías en función de su área de trabajo: gestión/documentación del proyecto y desarrollo del mismo. Concretamente, la segunda categoría está dividida en varios subapartados que corresponden a las diversas iteraciones de trabajo.

En cuanto a la gestión temporal del proyecto, este se inicia el 15 de febrero de 2021 y finaliza con la entrega de su memoria el 21 de junio del mismo año (aproximadamente cuatro meses de desarrollo ó 135 días). Se ha estimado una dedicación de 4-5 horas diarias al TFG, aunque esta cantidad ha variado en función de la disponibilidad del desarrollador (exámenes y entregas de otras asignaturas).

Gestión del Proyecto

- **GDP1 - Definición del alcance y contexto (20h):** se redacta un documento que trate de introducir, justificar y acotar la idea del proyecto. También se hace especial hincapié en la metodología del proyecto. Debería finalizarse el 2 de marzo y no tiene dependencias.
- **GDP2 - Planificación temporal (8h):** se crea un segundo documento que especifica las tareas necesarias para la realización del TFG y su situación temporal. Debe estar lista antes del 8 de marzo. Dependencias: GDP 1.
- **GDP3 - Gestión económica y sostenibilidad (16h):** se constituye un tercer informe que estime e identifique los costes económicos del proyecto y presente un informe sobre el impacto en la sostenibilidad en la dimensión ambiental, social y económica. Este cometido debe de estar finalizado antes del 15 de marzo. Dependencias: GDP 2.
- **GDP4 - Documento final de GEP (8h):** por último, en esta tarea se ha creado un cuarto documento a partir de los tres de tareas anteriores. Además, se han incorporado las mejoras convenientes que haya indicado el tutor de GEP. Debe de estar acabada antes del 22 de marzo. Dependencias: GDP 3.

- **GDP5 - Redacción de la memoria (80h):** preparación de la entrega final. Esta tarea se ha realizado de forma continua y debe estar acabada el 21 de junio. Dependencias: GDP 4.
- **GDP 6 - Preparación de la presentación (12h):** elaboración de la defensa del proyecto en el tribunal. La fecha límite de esta fase es el 28 de junio. Dependencias: GDP 5.

Inception

- **I1 - GDD (16h):** este cometido consiste en la creación del documento de diseño del videojuego (en inglés *Game Design Document*) que especifica el *gameplay* y el concepto del FPP. La última versión del GDD se encuentra en el [Apéndice] y debía de finalizar antes del 28 de febrero. No tiene dependencias.
- **I2 - Preparación del entorno de desarrollo (8h):** instalación y puesta en marcha de las aplicaciones y herramientas que se utilizarán a lo largo del proyecto (véase Recursos). No tiene dependencias.
- **I3 - Especificación de la gramática (12h):** realización de un esquema UML que especifique los componentes a implementar de un hechizo y su interacción entre ellos. Dependencias: DI1.
- **I4 - Preparación de un proyecto de prueba (8h):** para poder testear las tareas que se realicen a continuación, será necesario crear una serie de funcionalidades básicas (escenario, enemigo, jugador, cámara y sistema de control básicos). Dependencias: DI2.

***Sprint* 1**

- **CH1 - Elementos (8h):** definición e implementación de los seis elementos naturales primarios y secundarios que tendrá disponible el jugador. Dependencias: DI3, DI4.
- **CH2 - Efectos (12h)**: producción de los efectos (es decir, de las consecuencias que tendrán los hechizos sobre el objetivo) y asignación a los diversos elementos. Dependencias: DCH1.
- **CH3 - Formas (12h):** creación de los contenedores que podrán tener los diversos elementos. Dependencias: DCH1.
- **CH4 - Comportamientos (16h):** implementación de la manera de actuar de los conjuros y asignar uno por defecto a cada forma. Dependencias: DCH3.

- **CH5 - Generación de un conjunto de hechizos predefinido (12h):** tal como una bola de fuego o una ventisca para asegurar el potencial de ocio de la idea del proyecto. Dependencias: DCH4.

***Sprint* 2**

- **CB1 - Sistema de vida (4h):** creación del componente de vida que incorporará el jugador y los enemigos. Si la vida del jugador llega a cero, la partida finalizará. Por contrapartida, si la vida del enemigo es cero, este morirá. Dependencias: DCH5.
- **CB2 - Diseño del primer escenario (16h):** es decir, el lugar donde sucederá la acción principal del videojuego. Estará inspirado en una isla. Dependencias: DI4.
- **CB3 - Enemigos (16h):** desarrollo de varios paquetes de IA básicos que den vida a los enemigos del juego. Dependencias: DCB1.
- **CB4 - Interfaz de usuario (8h):** diseño de la distribución y aspecto de los botones y elementos que aparecerán en pantalla (vida, atacar, opciones, etc.). No tiene dependencias.

***Sprint* 3**

- **G1 - Sistema de Input (12h):** creación del sistema de entrada que ofrezca al jugador todos los elementos disponibles para crear una frase (o hechizo), mediante su arrastre por la pantalla. Dependencias: DCB4.
- **G2 - Introducción a ANTLR (20h):** debido a la inexperiencia del desarrollador, se ha asignado una tarea con una sola finalidad: familiarizarse con la herramienta. No tiene dependencias.
- **G3 - Creación de la gramática (16h):** en un archivo con extensión .g4, que sea capaz de leer e interpretar la entrada del usuario. Dependencias: DG2.
- **G4 - Creación de hechizos (20h):** integración del plugin de ANTLR en el proyecto de Unity, de forma que los hechizos generados tengan asignados sus propiedades (elementos, efectos y forma) y su comportamiento. Dependencias: DG1, DG3.

***Sprint* 4**

- **P1 - Representación de la forma (16h):** por cada una de las formas disponibles, definir las propiedades (tamaño y velocidad de partículas, imágenes, etc.) que tendrá el sistema de partículas. Dependencias: DG4.
- **P2 - Representación del elemento (8h):** implementación de los colores específicos de cada elemento. Dependencias: DP1.
- **P3 - Integración del sistema de partículas en *Scriptable Objects* (20h):** a partir de los hechizos creados en la iteración anterior, ser capaz de añadir este sistema de partículas dinámico a los conjuros personalizados y que se muestren correctamente. Dependencias: DP2.

***Sprint* 5**

- **PD1 - Sonidos (12h):** agregación de efectos sonoros a las varias acciones presentes en el juego. Dependencias: DCB3, DP3.
- **PD2 - Iluminación (4h):** para dar un poder embellecedor al juego. No tiene dependencias.
- **PD3 - Menú principal (12h):** deberá de tener un buen aspecto para atraer al jugador. Como se especifica en el GDD, se representará como una escena más (sin enemigos). No tiene dependencias.
- **PD4 - Sistema de misiones (24h):** a partir de un set básico, añadir NPCs a la escena del menú que sean capaces de generar misiones. Dependencias: DD3.
- **PD5 - Sistema de guardado (8h):** que sea capaz de guardar la configuración y los hechizos del usuario. Dependencias: DD1, DD4.

Despliegue

- **D1 - Testing (12h):** garantizar que el FPP funcione correctamente en diversos dispositivos móviles. Dependencias: DPD5.
- **D2 - Corrección de bugs (24h):** corrección de los errores que surjan durante la realización de la tarea anterior. Dependencias: DD1.

9.2. RECURSOS MATERIALES

Hardware

Las fases de desarrollo se han realizado en un ordenador de altas prestaciones personalizado de las siguientes características:

- **CPU:** Ryzen 7 3700X (8 núcleos, 16 hilos) a 3.6 GHz.
- **Placa Base:** MSI B450 GAMING PRO CARBON AC.
- **RAM:** 16 GB a 3200 MHz CL16.
- **GPU:** GTX 1660 Super de 6 GB GDDR6.
- **SO:** Windows 10 Pro.

Para realizar el despliegue del FPP, se ha contado con un teléfono de gama media Android (en concreto un Pocophone F1 de Xiaomi) con las siguientes características:

- **CPU:** Qualcomm Snapdragon 845 (4x Kryo 385 2.8 GHz + 4x Kryo 385 1.8 GHz).
- **GPU:** Qualcomm Adreno 630 a 710 MHz.
- **RAM:** 6 GB LPDDR4X.
- **Pantalla:** LCD IPS FHD+ (1080x2246 píxeles) (416 PPIs).
- **Capacidad de la Batería:** 4000 mAh.

Software

Un proyecto de estas características solicita el uso de múltiples programas y utilidades. En producción, se ha utilizado utilizará Unity y Visual Studio Code (VSC). Los diagramas UML se han producido mediante la herramienta UMLet, un editor de esquemas de código abierto gratuito. El seguimiento del proyecto y la comunicación escrita desarrollador/director se realiza mediante Meet y Hangouts respectivamente. Por último, la documentación se ha creado con el editor de textos Microsoft Word y se ha compartido con el resto del equipo mediante Google Drive.

Todas las licencias de uso requeridas y su coste de estos programas se encuentran especificados en el próximo capítulo relevante al presupuesto económico del proyecto.

9.3. ESTIMACIONES Y GANTT

Resumen de tareas

ID	Tarea	Horas	Dependencias	Recursos
Gestión y documentación del proyecto				
GDP1	Definición del alcance y contexto	20	-	PC, GDrive
GDP2	Planificación temporal	8	GDP1	PC, GDrive
GDP3	Gestión económica y sostenibilidad	16	GDP2	PC, GDrive
GDP4	Documento final de GEP	8	GDP3	PC, GDrive
GDP5	Redacción de la memoria	80	GDP4	PC, GDrive
GDP6	Preparación de la presentación	12	GDP5	PC, GDrive
Inception				
I1	GDD	16	-	PC, GDrive
I2	Entorno de desarrollo	8	-	PC
I3	Especificación de la gramática	12	I1	PC, UMLet
I4	Creación del proyecto en Unity	8	I2	PC, Unity
***Sprint* 1: Componentes de un Hechizo**				
CH1	Elementos	8	I3, I4	PC, Unity, VSC
CH2	Efectos	12	CH1	PC, Unity, VSC
CH3	Formas	12	CH2	PC, Unity, VSC
CH4	Comportamientos	16	CH3	PC, Unity, VSC
CH5	Generación de un conjunto de hechizos predefinido	12	CH4	PC, Unity, VSC
***Sprint* 2: Componentes Básicos del Juego**				
CB1	Sistema de vida	4	CH5	PC, Unity, VSC
CB2	Diseño del escenario principal	16	I4	PC, Unity, VSC
CB3	Enemigos	16	CB1	PC, Unity, VSC
CB4	Interfaz de usuario	8	-	PC, Unity, VSC
***Sprint* 3: Gramática**				
G1	Sistema de Input	12	CB4	PC, Unity, VSC
G2	Introducción a ANTLR	20	-	PC, VSC
G3	Creación de la gramática	16	G2	PC, VSC
G4	Creación de hechizos	20	G1, G3	PC, Unity, VSC
***Sprint* 4: Partículas**				
P1	Representación de la forma	16	G4	PC, Unity, VSC
P2	Representación del elemento	8	P1	PC, Unity, VSC
P3	Integración del sistema de partículas en *ScriptableObjects*	20	P2	PC, Unity, VSC

***Sprint* 5: Pequeños Detalles**				
PD1	Sonidos	12	CB3, P3	PC, Unity, VSC
PD2	Iluminación	4	-	PC, Unity, VSC
PD3	Menú principal	12	-	PC, Unity, VSC
PD4	Animaciones	16	-	PC, Unity, VSC
PD5	Reconocimiento de runas	16	G4	PC, Unity, VSC
Despliegue				
D1	Testing	12	PD5	PC, Unity, A
D2	Correción de bugs	24	PD5	PC, Unity, A
TOTAL (horas)		500		

Tabla 2: Tabla resumen de tareas. Fuente: Elaboración propia

Diagrama de Gantt

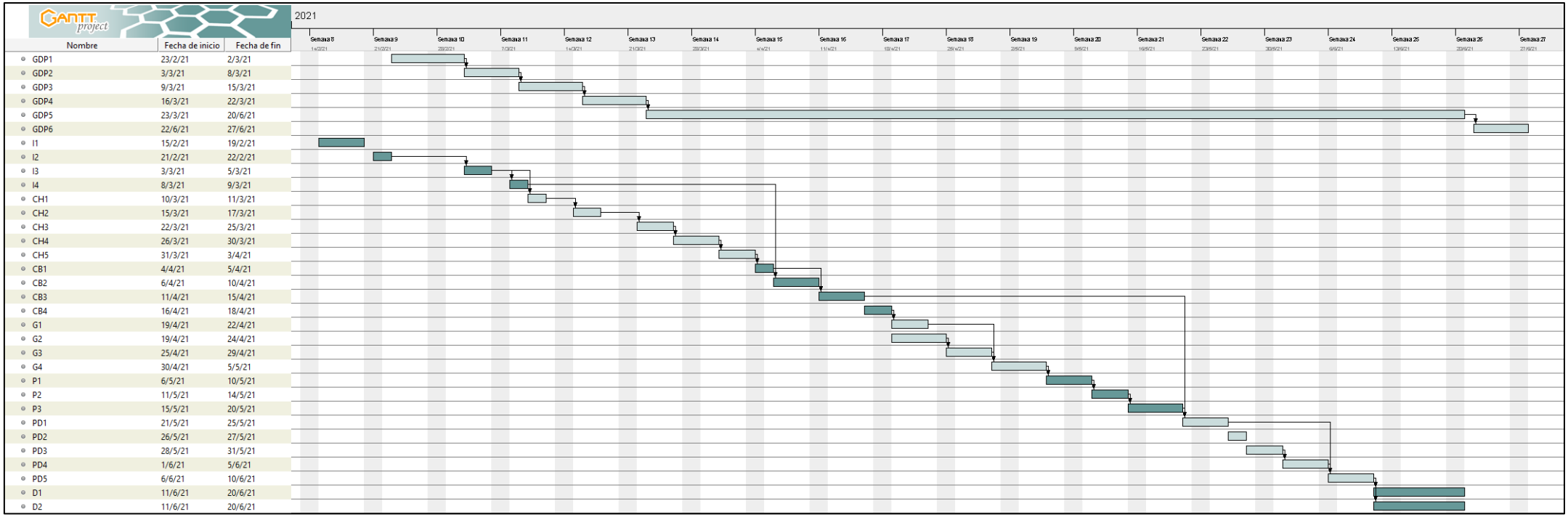


Figura 45: Diagrama de Gantt. Fuente: Elaboración propia

9.4. DESVIACIONES SOBRE EL PLAN

En las cuatro primeras etapas de desarrollo la planificación se ha seguido al pie de la letra y no se han acumulado tareas. No obstante, se ha incurrido en un pequeño retraso en el Sprint 3 debido a la asignación de trabajos y exámenes al desarrollador. Esto no ha supuesto ningún problema puesto que, a partir de ese momento, el autor ha podido invertir horas extras para recuperar el retraso. Esto ha sido posible porque el desarrollador ha estado única y exclusivamente dedicado a la finalización del TFG.

Por otro lado, es destacable mencionar que se ha cambiado la fecha de finalización del proyecto, pasando del 27 de mayo al más acorde 21 de junio. Esto se debe a una confusión por parte del autor durante la realización del informe de GEP.

En el resto de tareas, el trabajo se ha realizado acorde al plan impuesto en marzo de este mismo año.

CAPÍTULO 10: PRESUPUESTO

10.1. ESTIMACIÓN DE LOS COSTES

Personal

Tras planificar las tareas del proyecto (véase Descripción de Tareas), esta sección analizará y expondrá el coste económico que conllevan estas actividades. Para ello, se han creado cuatro roles encargados de llevar a cabo de forma individual (o en colaboración con otros roles) tareas distintas. Cada rol tiene un coste diferente, ya que el grado de responsabilidad o la complejidad de las tareas ejecutadas no es la misma. Ordenados de mayor a menor grado de compromiso, estos roles son los siguientes.

- **Jefe de proyecto:** es el principal responsable del proyecto, puesto que es el encargado de establecer su dirección y asegurar su correcto desarrollo.
- **Arquitecto de software:** de vital importancia, el arquitecto es el responsable de diseñar la arquitectura del proyecto, es decir, la especificación de los requisitos y de decisiones de alto nivel (plataformas, herramientas, clases y servicios).
- **Programador junior:** encargado de implementar los métodos definidos por el arquitecto para hacer realidad la visión del producto del jefe de proyecto.
- **Tester:** por último, este se responsabiliza de verificar la calidad del código implementado y asegurar su correcto funcionamiento.

La Tabla 3 contiene el sueldo promedio de los distintos roles en las diferentes empresas de nuestro país. El coste por hora se ha obtenido dividiendo el salario bruto anual (primera columna) por el número de horas trabajadas. Este último se calcula multiplicando el número de días laborales (365 días - 30 días de vacaciones - 52*2 días de fin de semana) por el número de horas al día laborales (se ha asumido jornada completa, es decir, 8 horas). La Seguridad Social, por su parte, supone un recargo del 30% sobre el salario original.

Rol	Sueldo (€/año)	Sueldo (€/h)	Sueldo + SS (€/h)
Jefe de proyecto	47630	27,06	35,18
Arquitecto de software	46914	26,66	34,65
Programador junior	19285	10,96	14,24
Tester	25201	14,32	18,61

Tabla 3: Sueldo y coste de los distintos roles del equipo de desarrollo. Fuente: Elaboración propia a partir de datos de [55].

En este caso particular, el rol de jefe de proyecto lo llevarán a cabo el autor y el director del mismo y, en menor medida, el ponente y el tutor de GEP. Por otro lado, los roles de Arquitecto de Software, Programador y Tester los asume de nuevo el autor del trabajo.

Con estos datos, se puede calcular el coste que requerirá cada tarea.

ID	Horas					Coste (€)
	JP	AS	PJ	T	Total	
Gestión y Documentación del Proyecto					**144**	**5065,92**
GDP1	20	0	0	0	20	703,60
GDP2	8	0	0	0	8	281,44
GDP3	16	0	0	0	16	562,88
GDP4	8	0	0	0	8	281,44
GDP5	80	0	0	0	80	2814,40
GDP6	12	0	0	0	12	422,16
Inception					**44**	**1341,96**
I1	16	0	0	0	16	562,88
I2	4	0	4	0	8	197,68
I3	2	10	0	0	12	416,86
I4	2	0	4	2	8	164,54
Sprint 1: Componentes de un hechizo					**60**	**1072,54**
CH1	2	0	6	0	8	155,80
CH2	2	0	10	0	12	212,76
CH3	2	0	10	0	12	212,76
CH4	2	0	14	0	16	269,72
CH5	2	0	8	2	12	221,50
Sprint 2: Componentes Básicos del Juego					**44**	**790,62**
CB1	1	0	2	1	4	82,27
CB2	2	0	14	0	16	269,72
CB3	2	0	12	2	16	278,46
CB4	2	0	5	1	8	160,17
Sprint 3: Gramática					**68**	**1495,39**
G1	1	0	10	1	12	196,19
G2	2	4	14	0	20	408,32
G3	2	8	6	0	16	433,00
G4	2	6	10	2	20	457,88
Sprint 4: Partículas					**44**	**811,56**
P1	4	0	12	0	16	311,60
P2	2	0	6	0	8	155,80
P3	2	0	14	4	20	344,16

Sprint 5: Pequeños Detalles					60	1060,19
PD1	2	0	8	2	12	221,50
PD2	1	0	3	0	4	77,90
PD3	1	0	10	1	12	196,19
PD4	1	0	12	3	16	261,89
PD5	1	2	10	3	16	302,71
Despliegue					**36**	**657,58**
D1	2	0	0	10	12	256,46
D2	2	0	18	4	24	401,12
Total	**210**	**30**	**222**	**38**	**500**	**12295,76**

Tabla 4: Coste personal para cada tarea. Fuente: Elaboración propia.

Costes Generales

En esta sección se incluyen los costes que no se consideran como costes de producción, pero son necesarios para el desarrollo correcto del proyecto. En este caso, se puede distinguir entre costes materiales (hardware y software) y costes indirectos, aunque la amortización se calcula mediante la misma fórmula para ambos casos.

*Amortización (€) = (coste (€) * duración del proyecto (h)) / (vida útil (años) * días laborables (días/año) * horas diarias)*

Con una duración del proyecto de 500 horas, 220 días laborables al año y una jornada de trabajo completa (8 horas), los costes materiales del proyecto son los que aparecen en la Tabla 4. Este coste hay que multiplicarlo por cuatro, puesto que se necesita un ordenador para cada empleado. La mayoría de programas de software que se han utilizado tienen licencia libre, pero existen ciertas excepciones que también aparecen mostradas en la Tabla 5.

Hardware			
Recurso	**Coste (€)**	**Vida útil (años)**	**Amortización (€)**
Ordenador personalizado	1190	6	56,34
Software			
Unity	1656	1	470,45
Windows 10	259	6	12,26
Total			**539,06**

Tabla 5: Amortización del hardware y software del proyecto. Fuente: Elaboración propia a partir de datos de [56] y [57]

Por último, se ha decidido alquilar una sala u oficina de trabajo para facilitar las cuentas y dotar al equipo de trabajo un lugar donde poder trabajar y reunirse cómodamente (Tabla 6).

Oficina			
Recurso	**Precio (€/mes)**	**Meses**	**Total (€)**
Coworking Space	300	4	1200

Tabla 6: Costes indirectos del área de trabajo. Fuente: Elaboración propia a partir de datos de [58]

Contingencia

Para todos aquellos costes y gastos no anticipados se ha añadido una partida de contingencia capaz de cubrirlos. En esta clase de proyectos, como mínimo siempre se asigna un 10% extra respecto al coste total. En concreto, este trabajo ha designado una contingencia del 15%. El impacto de esta partida puede verse reflejado en la Tabla 7.

Coste Personal (€)	Costes Generales (€)	Contingencia (%)	Contingencia (€)
12295,76	3356,25	15	2347,80

Tabla 7: Costes de contingencia sobre el proyecto. Fuente: Elaboración propia

Imprevistos

Anteriormente (véase Riesgos y Obstáculos) se especifican todos los riesgos detectados, así como las medidas y el curso de acción a tomar. Estas alternativas pueden conllevar un incremento de horas trabajadas en nuestro personal, que tendrán que ser remuneradas de forma conveniente. Todo esto se encuentra recogido en la Tabla 8 que se encuentra a continuación.

Riesgo	Probabilidad (%)	Coste estimado (€)	Coste (€)
Período de tiempo limitado	50	281,44	140,72
Inexperiencia en las tecnologías utilizadas	50	408,32	204,16
Compatibilidad entre plataformas	75	657,58	493,19
Total			**838,07**

Tabla 8: Control de imprevistos del proyecto. Fuente: Elaboración propia

Costes Totales

Por último, en la Tabla 9 se encuentra un resumen de la suma de todos los costes del proyecto (costes personales, generales, contingencia e imprevistos). Si se suman todos los conceptos mencionados, se puede apreciar que el trabajo cuesta aproximadamente algo menos de 20000€.

Concepto	Coste (€)
Costes Personales	12295,76
Costes Generales	3356,25
Contingencia	2347,80
Imprevistos	838,07
Total	**18837,88**

Tabla 9: Coste total del proyecto. Fuente: Elaboración propia

10.2. CONTROL DE GESTIÓN

Para garantizar que el proyecto sigue con la planificación previa, este apartado propondrá el uso de ciertos medidores e indicadores. Para ello, es necesario recoger información extra (horas reales dedicadas a cada tarea) y actuar en consecuencia si se produce alguna desviación fuera de la planificación.

Personal y Material

Para cada rol e ítem implicados en el proyecto, la desviación económica es:

$$Coste\ real = horas\_reales * coste\_rol$$

$$Desviación\ en\ coste\ por\ tarifa\ (o\ en\ precio) = (coste\_estimado - coste\_real) * horas\_reales$$

$$Desviación\ en\ eficiencia\ (o\ en\ consumo) = (horas\_estimadas - horas\_reales) * coste\_real$$

$$Desviación\ por\ rol/ítem\ total = Desviación\_precio + Desviación\_consumo$$

Total

$$Desviación\ total\ en\ coste\ por\ tarifa = \sum Desviación\_coste\_tarifa_r$$

$$Desviación\ total\ en\ eficiencia = \sum Desviación\_eficiencia_r$$

$$Desviación\ total = Desviación\_total\_coste\_tarifa + Desviación\_total\_eficiencia$$

donde los totales son el sumatorio de la desviación del conjunto de roles e ítems que han participado en el proyecto (jefe de proyecto, programador, ordenadores, etc.).

CAPÍTULO 11: LEYES Y REGULACIONES

Debido que la gramática es de elaboración propia y el FPP no recoge datos personales de ningún tipo, el proyecto nunca entra en conflicto con ninguna ley de protección de datos. No obstante, el trabajo sí que está sujeto a un conjunto de regulaciones y licencias impuestos por los creadores de las herramientas y recursos utilizados:

- En primer lugar, Unity establece en su licencia Personal que cualquier desarrollador puede monetizar su creación siempre y cuando la recaudación total de sus videojuegos no supere los 100.000 dólares anuales [59]. En caso de querer superar el límite, la empresa creadora deberá comprar una versión superior como Unity Plus (con un límite de 200.000 dólares) o Unity Pro/Enterprise (ilimitados).

  A diferencia de otras compañías (como Epic Games con su Unreal Engine), Unity no cobra ningún tipo de regalía por monetizar el proyecto. Además, el contenido creado con Unity Personal pertenece únicamente a la entidad desarrolladora.
- Por otro lado, por falta de tiempo el FPP utiliza modelos y texturas de otros creadores, por lo cual estos están sujetos a las condiciones de los autores de dichos recursos. La mayoría tienen el permiso para utilizarse en proyectos monetizables, pero otros están limitados para uso personal y/o académico. Por lo tanto, si se quiere generar algún tipo de ingreso con el videojuego, el autor deberá eliminar/sustituir aquellos recursos que no cumplan la normativa o pedir permiso de uso a la compañía propietaria.

CAPÍTULO 12: INFORME DE SOSTENIBILIDAD

En todo proyecto software es importante analizar las diferentes dimensiones de la sostenibilidad para conocer el impacto que un producto puede producir en la sociedad. La responsabilidad de toda consecuencia ocasionada en el entorno recae directamente en el equipo desarrollador del producto.

12.1. DIMENSIÓN AMBIENTAL

Aunque el consumo eléctrico de un dispositivo móvil es relativamente bajo, el impacto ambiental de una aplicación debe de medirse a una escala mayor. Uno debe de tener en cuenta la enorme cantidad de teléfonos y tabletas que la gente dispone tanto en sus casas como en sus trabajos. Si se suma el gasto energético de todos estos artilugios, se obtiene una cantidad que no se puede ignorar.

Es cierto que el consumo energético del producto ya se ha tenido en cuenta en los requisitos no funcionales del proyecto para optimizar el uso de la batería. No obstante, cualquier reducción energética, por muy mínima que sea, siempre es bienvenida a la hora de minimizar el impacto de la aplicación en el medio ambiente.

Por otro lado, el proyecto no requiere de ningún servidor externo para funcionar, factor importante que también ayuda a disminuir la huella ecológica.

12.2. DIMENSIÓN SOCIAL

Gracias a este trabajo, multitud de personas tienen otro medio más (junto al cine y a la lectura) con el cual poder distraerse y escapar, aunque sea por un momento, de los problemas del mundo real. El proyecto está abierto a un público muy amplio, por lo que se ha tenido especial cuidado a la hora de incluir elementos que de forma potencial puedan ofender a determinados colectivos culturales y/o religiosos.

12.3. DIMENSIÓN ECONÓMICA

Mediante el uso del metamodelo y el patrón arquitectónico ECS, el TFG ha hecho un mejor uso los fondos asignados gracias a la reutilización del código. Todo aquello que se ha ahorrado en las tareas se ha podido invertir en mejoras de materiales esenciales. El objetivo final de todo ello es asegurar unas condiciones dignas y justas para todos los empleados involucrados en el trabajo.

CAPÍTULO 13: CONCLUSIÓN

Por último, ya en el apartado final del proyecto, ha llegado la hora de realizar una autocrítica en retrospectiva y evaluar paso por paso los resultados finales del mismo. El camino recorrido ha sido extenso y, por ello, se ha dividido este capítulo en varios subapartados.

13.1. CUMPLIMIENTO DE LOS OBJETIVOS

En primer lugar, el objetivo principal del proyecto se ha cumplido con creces. La gramática propuesta de forma inicial en el GDD ha sido ampliada en muchos aspectos. Se han implementado una multitud de componentes planteados y existe la propuesta de varios comportamientos para versiones posteriores del proyecto. Además, se ha logrado diseñar un sistema modular (metamodelo) que admite la inclusión fácil de nuevos efectos. Esta gramática tiene también la capacidad de adaptarse al nivel de habilidad del usuario, requisito fundamental para atraer al público que se especificó en la fase inicial.

En lo que al prototipo se refiere, sus principales metas también se han logrado con éxito. El FPP dispone de un escenario principal donde los potenciales clientes pueden probar el potencial de este nuevo modelo. Sus personajes están animados en su totalidad y contiene toda clase de subsistemas básicos que cualquiera esperaría tener (sonido, sombras, movimiento, entre otros).

Por desgracia, debido a la cantidad de tiempo limitada, no ha sido posible implementar ninguno de los aspectos opcionales propuestos en el Capítulo 3. Más sobre este tema se comentará en el apartado 13.3.

13.2. DESEMPEÑO DE COMPETENCIAS TÉCNICAS

En esta sección se mostrarán el planteamiento y el trabajo de las distintas competencias técnicas asociadas al proyecto. Estas fueron planteadas de forma conjunta entre el autor y el director del TFG antes de su matriculación.

CES 1.1: Desarrollar, mantener y evaluar sistemas y servicios software complejos y críticos

Diseñar un metamodelo requiere un nivel de abstracción superior al de la creación de un modelo estándar tradicional. Ser capaz de integrarlo en otro sistema software como puede ser un videojuego aumenta todavía más su complejidad. El desarrollador ha tenido que plantear de forma meticulosa la arquitectura del FPP, proceso que conllevó varias semanas de trabajo y de *feedback* por parte del director.

El resultado final, sin embargo, cumple con todos los requisitos planteados en el inicio y demuestra que los videojuegos son otra área más de aplicación apta de los metamodelos.

CES 1.2: Dar solución a problemas de integración en función de las estrategias, de los estándares y de las tecnologías disponibles

Aunque el FPP ha sido desarrollado en el motor de Unity, ha sido necesario integrar el código generado por ANTLR dentro del mismo. Este problema se ha resuelto mediante la inclusión de la capa de Lógica en la arquitectura, puesto que esta no depende de ningún programa. Todo el proceso es transparente al usuario, que no nota ningún tipo de retraso a la hora de crear hechizos.

CES 1.3: Identificar, evaluar y gestionar los riesgos potenciales asociados a la construcción de software que se puedan presentar

En la fase inicial del proyecto se planteó un apartado de riesgos implícito a su construcción. La planificación se realizó con estos peligros en mente, habiéndose añadido partidas de contingencia para asegurar el cumplimiento de los objetivos aún en el peor de los casos. Aunque no todas las medidas hayan sido necesarias, es una práctica útil a tener en cuenta para cualquier trabajo profesional.

CES 1.7: Controlar la calidad y diseñar pruebas en la producción de software

Aunque no se ha diseñado ningún tipo de test unitario durante el desarrollo del TFG, se ha incluido y realizado una fase final de despliegue en la planificación. Su finalidad es garantizar el correcto funcionamiento del sistema en dispositivos Android mediante la monitorización de los recursos utilizados y la resolución de bugs.

CES 1.8: Desarrollar, mantener y evaluar sistemas de control y de tiempo real

Todo videojuego que existe en el mercado es un sistema ejecutado en tiempo real que tiene que seguir funcionando de forma independiente a la entrada del jugador. Para ello, el desarrollador debe de entender la su estructura de funcionamiento (*Game Loop*) y garantizar su correcto rendimiento para poder dar una imagen final fluida y estable.

CES 2.1: Definir y gestionar los requisitos de un sistema software

La estructura final del TFG y su implementación está condicionada por el análisis de requisitos realizado en sus primeras fases de especificación. En ese momento fue cuando se identificaron y se determinaron sus criterios de aceptación que establecen el cumplimiento de los mismos. Este proceso es fundamental para fijar unos mínimos de calidad y para acabar de acotar el alcance de cualquier proyecto.

CES 3.1: Desarrollar servicios y aplicaciones multimedia

Los videojuegos están considerados como sistemas multimedia, puesto que presentan información utilizando múltiples canales de comunicación (texto, audio y vídeo). El FPP implementado es, entonces, una aplicación multimedia y ha requerido un estudio intenso de la presentación de todos estos datos para ofrecer la mejor experiencia de usuario posible (UX).

13.3. PLANES FUTUROS

Gracias a la popularización del fenómeno de Internet, en la actualidad es posible dar de forma fácil soporte a los distintos productos informáticos mediante actualizaciones.

Aunque la versión final del proyecto cumple con todos los objetivos propuestos en el mes de marzo, este es sin lugar a dudas mejorable en varios aspectos.

- En primer lugar, el producto puede mejorarse con la inclusión de las funcionalidades extras definidas por los objetivos opcionales del Capítulo 3. Todas ellas permitirían explotar aún más el metamodelo planteado, puesto que ofrecerían una multitud de situaciones nuevas y distintas a las ya existentes en las que utilizar los hechizos. Quizás, incluso, abriría las puertas a nuevas combinaciones de efectos que no tendrían sentido sin estas condiciones.

- El apartado visual también podría beneficiarse con la inclusión de sistemas de partículas mejor elaborados por parte de especialistas. Aunque el método actual de tratamiento de efectos funciona y transmite todo el *feedback* necesario, hay que admitir que después de un rato es algo monótono.
- Por último, la gramática de hechizos se vería muy beneficiada si estuviera implementada en un videojuego real (en vez de un prototipo). En ese entorno, el sistema podría incorporarse junto a otros elementos (historia, rompecabezas o sigilo) dando lugar a experiencias jugables únicas. De esta forma, el producto lograría diferenciarse del resto y atraería a un público mayor.

13.4. CONCLUSIÓN FINAL

Como último apartado, después de un largo recorrido de continua dedicación, hablaré con un tono más personal sobre cómo me ha afectado la realización de este proyecto.

A nivel de conocimientos, este TFG me ha ayudado a consolidar todas las ideas adquiridas durante los cuatro años de carrera. Este acontecimiento es la recompensa a todos estos años de intensa dedicación. Además, me ha permitido explorar áreas de la Informática (como los intérpretes) que eran todavía desconocidas para mí.

En cuanto a la forma de trabajar, gracias a él he aprendido a escuchar los distintos puntos de vista de los varios miembros implicados en el proyecto y de otros amigos y familiares cercanos. Gracias a ellos y a mis propios criterios, me he visto capacitado tomar la iniciativa y realizar todas las decisiones importantes que han marcado el rumbo del mismo. Por otro lado, la proactividad e interés a la hora de trabajar ha sido fundamental para empujar el TFG hacia adelante.

Por último, me gustaría destacar el hecho que la realización del trabajo me ha permitido saborear la experiencia de estar involucrado en un proyecto que podría ser real. Con él he aprendido el impacto que una aplicación puede inducir en la sociedad. Y es que una simple idea puede modificar en un giro de 180 grados el rumbo de todo el mundo.

CAPÍTULO 14: REFERENCIAS

[1] Pedro Herrero. La industria del videojuego facturó en 2020, en todo el mundo, más que el cine y los deportes juntos en EEUU. [Consulta: 21 de abril de 2021]. AS – Meristation. URL: https://as.com/meristation/2020/12/26/noticias/1608992024_963325.html

[2] Omri Wallach. 50 Years of Gaming History, by Revenue Stream (1970-2020). [Consulta: 28 de abril de 2021]. Visual Capitalist. URL: https://www.visualcapitalist.com/50-years-gaming-history-revenue-stream/

[3] Samit Sarkar. Diablo 3 lifetime sales top 30 million units. [Consulta: 27 de febrero de 2021]. Polygon. URL: https://www.polygon.com/2015/8/4/9097497/diablo-3-sales-30-million-units

[4] Chris Suellentrop. 'Skyrim' Creator on Why We'll Have to Wait for Another 'Elder Scrolls'. [Consulta: 27 de febrero de 2021]. Rolling Stone. URL: https://www.rollingstone.com/culture/culture-features/skyrim-creator-on-why-well-have-to-wait-for-another-elder-scrolls-128377/

[5] Eddie Makuch. Witcher 3's Sales By System--How The Game Has Sold On PC, PS4, Xbox One, And Switch. [Consulta: 27 de febrero de 2021]. GameSpot. URL: https://www.gamespot.com/articles/witcher-3s-sales-by-system-how-the-game-has-sold-o/1100-6475832/

[6] Arx Fatalis. [Consulta: 1 de marzo de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/Arx_Fatalis

[7] Arx Fatalis for PC Reviews. [Consulta: 24 de mayo de 2021]. Metacritic. URL: https://www.metacritic.com/game/pc/arx-fatalis

[8] Arx Fatalis. [Consulta: 24 de mayo de 2021]. Meristation. URL: https://as.com/meristation/juegos/arx_fatalis/

[9] Ray Hardgrit. Arx Fatalis – PC. [Consulta: 1 de marzo de 2021]. Superadventures in Gaming. URL: https://superadventuresingaming.blogspot.com/2014/02/arx-fatalis-pc.html

[10] The Elder Scrolls III: Morrowind. [Consulta: 1 de marzo de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/The_Elder_Scrolls_III:_Morrowind

[11] The Elder Scrolls IV: Oblivion. [Consulta: 1 de marzo de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/The_Elder_Scrolls_IV:_Oblivion

[12] The Elder Scrolls III: Morrowind for PC Reviews. [Consulta: 24 de mayo de 2021]. Metacritic. URL: https://www.metacritic.com/game/pc/the-elder-scrolls-iii-morrowind

[13] The Elder Scrolls IV: Oblivion for PC Reviews. [Consulta: 24 de mayo de 2021]. Metacritic. URL: https://www.metacritic.com/game/pc/the-elder-scrolls-iv-oblivion

[14] Lost Magic. [Consulta: 2 de marzo de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/LostMagic

[15] Lost Magic for DS Reviews. [Consulta: 24 de mayo de 2021]. Metacritic. URL: https://www.metacritic.com/game/ds/lost-magic

[16] Imágenes de Lost Magic. [Consulta: 1 de mayo de 2021]. 3DJuegos. URL: https://www.3djuegos.com/juegos/imagenes/1105/0/lostmagic/#-img-81466-1105-0-0

[17] Eragon (video game). [Consulta: 30 de abril de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/Eragon_(video_game)

[18] Eragon for DS Reviews. [Consulta: 25 de mayo de 2021]. Metacritic. URL: https://www.metacritic.com/game/ds/eragon

[19] Arend Hart. Eragon DS Review. [Consulta: 30 de abril de 2021]. Game Chronicles. URL: http://www.gamechronicles.com/reviews/nds/eragon/eragon.htm

[20] Two Worlds II. [Consulta: 2 de marzo de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/Two_Worlds_II

[21] Two Worlds II for PC Reviews. [Consulta: 24 de mayo de 2021]. URL: https://www.metacritic.com/game/pc/two-worlds-ii

[22] Outward. [Consulta: 24 de mayo de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/Outward

[23] Outward for PC Reviews. [Consulta: 24 de mayo de 2021]. Metacritic. URL: https://www.metacritic.com/game/pc/outward

[24] Rune Magic. [Consulta: 24 de mayo de 2021]. Outward Wiki. URL: https://outward.fandom.com/wiki/Rune_Magic

[25] Valheim. [Consulta: 24 de mayo de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/Valheim

[26] Valheim en Steam. [Consulta: 24 de mayo de 2021]. Steam. URL: https://store.steampowered.com/app/892970/Valheim/

[27] Explicación de la trama y la historia de Valheim. [Consulta: 24 de mayo de 2021]. La Neta Neta. URL: https://lanetaneta.com/explicacion-de-la-trama-y-la-historia-de-valheim-screen-rant/

[28] Miquel Rodríguez. Scrum: El pasado y el futuro. [Consulta: 18 de junio de 2021]. Netmind. URL: https://netmind.net/es/scrum-el-pasado-y-el-futuro/

[29] Bob Hartman. New to agile? INVEST in good user stories. [Consulta: 26 de mayo de 2021]. Agile For All. URL: https://agileforall.com/new-to-agile-invest-in-good-user-stories/

[30] James Robertson & Suzanne Robertson. Volere Requirements Specification Template. [Consulta: 26 de mayo de 2021]. University of Illinois Chicago. URL: https://www.cs.uic.edu/~i440/VolereMaterials/templateArchive16/c%20Volere%20template16.pdf

[31] Gramático, gramática | Definición | Diccionario de la lengua española | RAE - ASALE. [Consulta: 28 de febrero de 2021]. Real Academia Española. URL: https://dle.rae.es/gram%C3%A1tico

[32] Sintaxis | Definición | Diccionario de la lengua española | RAE - ASALE. [Consulta: 28 de febrero de 2021]. Real Academia Española. URL: https://dle.rae.es/sintaxis

[33] Sintagma | Definición | Diccionario de la lengua española | RAE - ASALE. [Consulta: 28 de febrero de 2021]. Real Academia Española. URL: https://dle.rae.es/sintagma

[34] Interpreter. [Consulta: 02 de marzo 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/Interpreter_(computing)

[35] Enrique José García Cota. Guía práctica de ANTLR 2.7.2. [Consulta: 28 de febrero 2021]. Departamento de Lenguajes y Sistemas Informáticos – Universidad de Sevilla. URL: http://www.lsi.us.es/~troyano/documentos/guia.pdf

[36] What is a Parser? [Consulta: 28 de febrero 2021]. Techopedia. URL: https://www.techopedia.com/definition/3854/parser

[37] Abstract Syntax Tree. [Consulta: 28 de febrero 2021]. Leanovate. URL: https://leanovate.github.io/bedcon/talk/abstract_syntax_tree.html

[38] Order of execution for event functions. [Consulta: 17 de junio de 2021]. Unity Documentation. URL: https://docs.unity3d.com/2020.2/Documentation/Manual/ExecutionOrder.html

[39] Metamodelling. [Consulta: 3 de junio de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/Metamodeling

[40] Ejemplos de análisis de oraciones simples III. [Consulta: 3 de junio de 2021]. Aulafácil. URL: https://www.aulafacil.com/cursos/lenguaje-secundaria-eso/analisis-sintactico/ejemplos-de-analisis-de-oraciones-simples-iii-l27676

[41] Unity User Manual: Using Components. [Consulta: 4 de junio de 2021]. Unity. URL: https://docs.unity3d.com/2020.2/Documentation/Manual/UsingComponents.html

[42] Entity Component System. [Consulta: 5 de junio de 2021]. Wikipedia. URL: https://en.wikipedia.org/wiki/Entity_component_system

[43] HackersUPC, Manuel Rello Saltor. El ingeniero de componentes, una introducción a la arquitectura ECS. [Consulta: 5 de junio de 2021]. Youtube. URL: https://www.youtube.com/watch?v=PJ-hGg5eXbM

[44] Terence Parr. About The ANTLR Parser Generator. [Consulta: 1 de marzo de 2021]. ANTLR. URL: https://www.antlr.org/about.html

[45] Mike Lischke. ANTLR4 grammar syntax support. [Consulta: 5 de junio de 2021]. Visual Studio Marketplace. URL: https://marketplace.visualstudio.com/items?itemName=mike-lischke.vscode-antlr4

[46] Unity Technologies. Debugger for Unity. [Consulta: 5 de junio de 2021]. Visual Studio Marketplace. URL: https://marketplace.visualstudio.com/items?itemName=Unity.unity-debug

[47] Terence Parr. (2013). The Definitive ANTLR 4 Reference. The Pragmatic Programmers. ISBN: 9781934356999

[48] Saumitra Srivastav. Antlr4 - Visitor vs Listener Pattern. [Consulta: 8 de junio de 2021]. Saumitra's Blog. URL: https://saumitra.me/blog/antlr4-visitor-vs-listener-pattern/

[49] Typicons. Heart Icon. [Consulta: 20 de junio de 2021]. Iconscout. URL: https://iconscout.com/icon/heart-1157

[50] Jacob O. Wobbrock et al. Impact of $-family. [Consulta: 12 de junio de 2021]. University of Washington. URL: https://depts.washington.edu/acelab/proj/dollar/impact.html

[51] Da Viking Code. PDollar Point-Cloud Gesture Recognizer. [Consulta: 12 de junio de 2021]. Unity Asset Store. URL: https://assetstore.unity.com/packages/tools/input-management/pdollar-point-cloud-gesture-recognizer-21660

[52] Mixamo. Adobe. URL: https://www.mixamo.com/#/

[53] Laura E. Shummon Maass. Artificial Intelligence in Video Games. [Consulta: 14 de junio de 2021]. Towards Data Science. URL: https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22

[54] Explain like I'm five: Terrain and .Raw Heightmaps. [Consulta: 13 de junio de 2021]. Unity Forum. URL: https://forum.unity.com/threads/explain-like-im-five-terrain-and-raw-heightmaps.428341/

[55] Glassdoor. [Consulta: 13 de marzo de 2021]. URL: https://www.glassdoor.es/index.htm

[56] Elige el plan adecuado para ti. [Consulta: 15 de marzo de 2021]. Unity Store. URL: https://store.unity.com/es/compare-plans?currency=EUR

[57] Windows 10. [Consulta: 15 de marzo de 2021]. Microsoft Store España. URL: https://www.microsoft.com/es-es/store/b/windows

[58] CoworkingSpain. [Consulta: 14 de marzo de 2021]. URL: https://coworkingspain.es/espacios/coworking/barcelona/utopicus-placa-catalunya

[59] Unity Personal. [Consulta: 28 de mayo de 2021]. Unity. URL: https://store.unity.com/es/products/unity-personal

CAPÍTULO 15: APÉNDICE

15.1. GDD

Resumen

En la actualidad, fundamentalmente hay dos tipos de sistemas de magia presentes en los videojuegos: un primero basado en hechizos ya predefinidos (que suele ser el más común, presente en sagas como Final Fantasy o World of Warcraft) y un segundo basado en conjuros configurables con opciones preestablecidas (véase la serie Elder Scrolls). Estos sistemas han funcionado muy bien a lo largo de estos años, pero su fórmula empieza a presentar cierto desgaste y nadie ha dedicado ningún esfuerzo en avanzar por esta dirección.

Por este motivo el proyecto quiere presentar una solución más innovadora. Para aumentar la variedad disponible al jugador, el TFG propone bajar un nivel y dar al usuario los componentes, que llamaremos sintagmas, necesarios para crear un hechizo. Este nombre no se ha escogido a la ligera, ya que para que el conjuro tenga éxito y funcione, el jugador tendrá que combinar sintagmas de manera que el resultado final tenga sentido. Dicho de otra forma, el proyecto se propone crear un lenguaje similar a cualquier otro, donde los hechizos hagan el papel de oración y, como tal, están formados por diversos sintagmas regidos por una gramática que les aportará sentido y coherencia.

Programar esta gramática en un videojuego implica también replantearse otros aspectos, como por ejemplo los apartados de visualización y de audio. Estos se tendrán que adaptar de forma que el sistema sea capaz de asignar los efectos y sonidos adecuados a cada conjuro para que el jugador reciba el *feedback* correcto.

Plataforma

Entre todas las plataformas disponibles en la actualidad, el proyecto se desarrollará fundamentalmente para dispositivos móviles debido a los siguientes motivos:

- Mayor flexibilidad a la hora de realizar la demostración en el día de la presentación. La idea principal es que cada persona perteneciente al jurado o al público pueda descargarse la demo y probarla en su propio teléfono.

- Capacidad de atraer a un público objetivo más amplio gracias a un esquema de control más simplificado respecto a otras plataformas como PC.
- En concreto, el videojuego será producido para Android, motivado principalmente por la falta de dispositivos Apple sobre los que poder testear durante las diversas fases de desarrollo.

Gameplay

Historia y Objetivos

Aunque el videojuego no va a tener una historia como tal, es necesario añadir algún tipo de trasfondo para dar una razón de ser al jugador. Por otro lado, también es conveniente diseñar un conjunto de objetivos y/o misiones que mantengan al usuario distraído y que favorezcan su progresión.

La acción tendrá lugar en nuestra aldea natal y sus alrededores. Nuestro personaje es el hechicero de la misma y su objetivo será encargarse de mantenerlo a salvo de los monstruos que lo acechan. Además, como todo buen sabio, el protagonista deberá solucionar los diversos problemas de los pueblerinos para asegurar el correcto funcionamiento de la aldea, como por ejemplo curar enfermedades u obtener metales alquímicos.

Este trasfondo determinará los tipos de misiones que recibirá el jugador, que por el momento serán tres:

- **Eliminación o de caza:** el jugador deberá matar una cantidad determinada de enemigos para completar con éxito la misión.
- **Recolección:** como protagonista deberemos buscar X número de ciertos objetos para superar la *quest*.
- **Fabricación:** para favorecer la experimentación del jugador con el sistema de magia, este deberá de formular un hechizo que cumpla con los requisitos de la misión para poder finalizarla.

A cambio de completar estos objetivos, los aldeanos proveerán al jugador con poder mágico, que será necesario para posteriormente poder crear nuevos conjuros en el altar de encantamiento.

Gramática de hechizos

Siendo el elemento principal y diferenciador del proyecto, la gramática de conjuros será el conjunto de normas y de leyes que el jugador debe cumplir con el objetivo de crear hechizos únicos y totalmente personalizables. En la Figura I podemos observar un esquema preliminar de esta gramática.

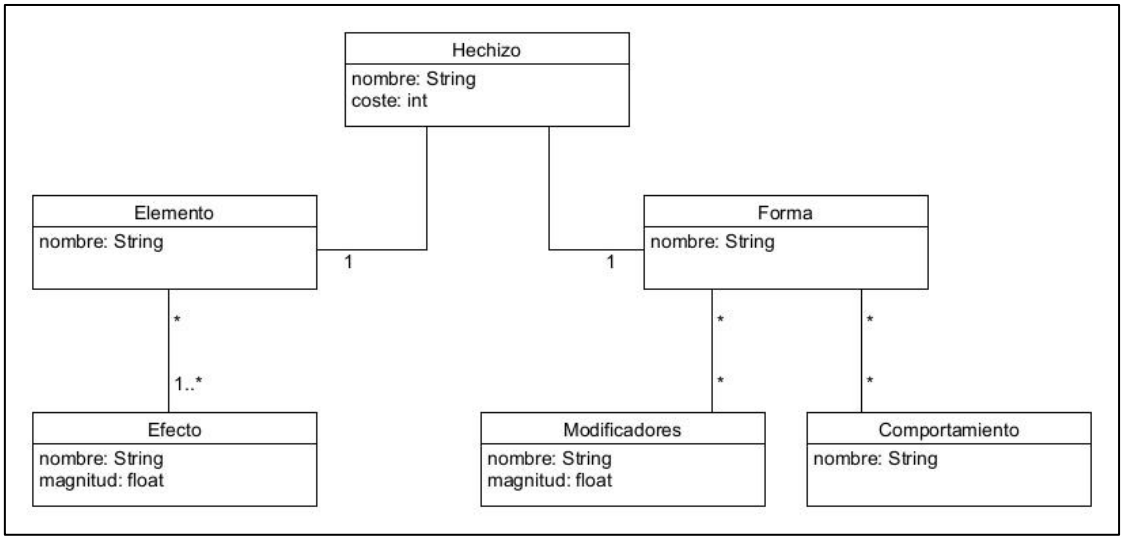


Figura I: Esquema inicial UML de los componentes de un hechizo. Fuente: Elaboración propia

A continuación, comentaré en qué consiste cada concepto. Para hacer la explicación más fácil, vamos a suponer que queremos crear la típica bola de fuego que inflige X puntos de daño a un enemigo:

- **Hechizo:** es, en definitiva, el producto final que obtenemos de la combinación de sintagmas (en nuestro ejemplo sería la propia bola de fuego). Se puede utilizar a cambio de un coste en cualquier situación, ya sea para realizar daño o para recuperarse de un largo combate.
- **Elemento:** conforma de qué está hecho el conjuro que vamos a crear. El elemento que conforma nuestro hechizo es el fuego. Cada enemigo tendrá ciertas resistencias y debilidades elementales, por lo que el jugador deberá explotarlas si quiere derrotar a sus adversarios con la máxima eficacia posible.
    - **Efecto:** es la consecuencia que ocurre en la entidad donde impacta el conjuro. En el ejemplo, nuestra bola de fuego contendrá sólo un efecto (daño) pero en otro conjuro, podremos añadir múltiples efectos. No obstante, para favorecer la variedad de hechizos, los efectos están limitados a ciertos elementos.

- **Forma:** constituye el molde que daremos a nuestro elemento. En nuestro caso, la forma que tiene el hechizo es de proyectil (bola).
    - **Modificador:** define las características físicas de la forma del conjuro. Considerando el caso de proyectil, podríamos añadir modificadores como la velocidad o el tamaño.
    - **Comportamiento:** por último, este apartado definirá la actuación de nuestro hechizo. Nuestra bola de fuego actualmente no tiene ningún comportamiento, pero podríamos hacer, por ejemplo, que persiguiera al enemigo que tengamos más cercano.

***Loop* principal**

Habiendo visto los apartados previos, podemos empezar a hablar del bucle que va a regir el *gameplay* principal del videojuego. En él podríamos englobar los diversos ítems del diagrama (Figura II) en tres fases:

- **Fase inicial:** en esta etapa el jugador busca una misión entre los diversos NPCs de la aldea y/o el tablón de anuncios. Una vez que ha encontrado la *quest* que más le atrae, el protagonista la acepta y procede (o no) a prepararse antes de partir, ya sea informándose de los enemigos a los cuales se va a enfrentar o creando nuevos hechizos que le serán de utilidad.
- **Fase de desarrollo:** el jugador explora el mundo para buscar a su objetivo (enemigos o ingredientes) y puede entrar en combate. Destruir enemigos recompensará al jugador con poder mágico, aunque perder el combate comportará una penalización sobre el jugador.
- **Fase final:** una vez se ha cumplido el objetivo de la misión, el usuario procederá (si así lo prefiere) a entregar la *quest* a la misma entidad que se la proporcionó al principio. Aquí obtendrá su recompensa y el jugador podrá decidir si quiere aceptar una nueva misión.

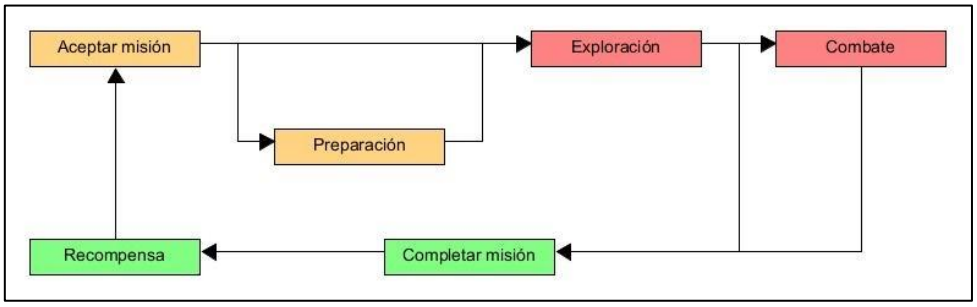


Figura II: Diagrama del loop principal. Fuente: Elaboración propia

Casting de hechizos

Para evitar llenar la pantalla de demasiados botones que ocupen un espacio considerable y aprovechar las ventajas que nos puede aportar un panel táctil, el videojuego implementará un método de entrada donde el jugador dibuja sobre la pantalla el glifo del hechizo que corresponda. Este símbolo será asignado por el jugador en el altar a la hora de crear el conjuro.

Este sistema, aunque poco conocido, ya se ha implementado en algunos títulos como por ejemplo Arx Fatalis, lanzado en PC en noviembre de 2002 (Figura III). Por otro lado, la mayoría de juegos de la consola Wii original utilizan el detector de movimientos del mando para realizar acciones, como Okami, lanzado en abril de 2006, que lo utiliza para efectuar movimientos diversos (destruir paredes, acuchillar enemigos, etc.).

Figura III: Sistema de casting de Arx Fatalis. Fuente: [9]

Multijugador

Para añadir una cierta gracia al proyecto, el videojuego contará con un sistema multijugador cooperativo de hasta un máximo de cuatro personas. Al no disponer de presupuesto, se basará en un "listen server", es decir, uno de los jugadores se encargará de alojar la partida mientras que el resto actuará como cliente.

En cuanto a las dinámicas de juego, el cliente se limitará a seguir los pasos del host. A efectos prácticos, esto se traduce en que el jefe aceptará la misión que más le convenga y los seguidores se limitarán a ayudarle con la tarea. Una vez completada, todos los jugadores recibirán la misma recompensa que será transferida a su partida en el móvil. El premio que suelten los enemigos también se repetirá para cada usuario de la partida.

Por último, cabe mencionar que el fuego amigo estará activado, por lo que es posible que los jugadores puedan hacerse daño entre sí. De esta manera se obligará a que mantengan una buena comunicación entre los componentes del grupo. Como efecto colateral, será posible que los usuarios puedan realizar *PvP* entre ellos si así lo desean.

Jefes

Con la incorporación de un sistema de jefes se espera aumentar el grado de dificultad en ciertas peleas. Como de costumbre, el enemigo tendrá una vida y daño (y su debida gran recompensa) bastante superiores al resto de enemigos, aunque seguirá un patrón a la hora de actuar que el jugador deberá aprender para poder explotarlo. Por otro lado, el jefe tendrá unas fortalezas y debilidades determinadas de forma que algunos hechizos serán más eficaces que otros.

Respecto al multijugador, el sistema de jefes recompensará a aquellos que colaboren entre sí, ya sea mediante la adopción de roles o la combinación de ataques de diversos jugadores. Será en estas batallas donde se explotará al máximo el sistema de magia.

Sistema de progresión

De implementación sencilla, el sistema de progresión consistirá de cinco categorías: Principiante, Aprendiz, Brujo, Sabio y Gran Maestre. Obviamente, el jugador pertenece inicialmente en la primera categoría y deberá ir subiendo de estatus si quiere desbloquear nuevos poderes y efectos que podrán materializarse en nuevos hechizos.

El objetivo final del sistema de progresión será guiar al jugador a través de la nueva gramática de conjuros y no introducirlo de golpe en un sistema complejo que no conoce inicialmente. Además, dará sentido a las misiones que acepte ya que notará como su personaje va evolucionando a medida que avanza en el juego.

UI y Cámara

La interfaz de usuario será sencilla con el objetivo de proporcionar al usuario el control más cómodo posible dentro de un teléfono Android. Para ello, seguiremos una serie de puntos clave vitales para conseguirlo:

- Un buen sistema de partículas será necesario para explotar al máximo el sistema de magia.

- El juego deberá tener una cámara perspectiva en tercera persona. Los videojuegos primera persona en Android tienden a tener un esquema de controles más complejo y puede resultar ortopédico para muchas personas.
- Evitar a toda costa esquemas de control como el *thumbstick*. Personalmente opino que hay alternativas mucho más cómodas y sencillas de utilizar.
- Implementar la menor cantidad de botones posible. Estos deberán de tener un tamaño considerable para que sean fáciles de pulsar.

El resultado de esta rúbrica será similar al de la Figura IV. Tendremos una cámara *top-down* centrada en el protagonista (cubo azul). Para movernos, el jugador pulsará el punto de la escena al cual se quiera mover y el juego automáticamente calculará la ruta más corta para llegar. Por otro lado, el apuntado tendrá un funcionamiento similar: el usuario simplemente seleccionará el objetivo tocándolo en la pantalla. Finalmente, tendremos dos botones que nos permitirán conjurar hechizos y abrir el menú de pausa (botones azul y negro respectivamente).

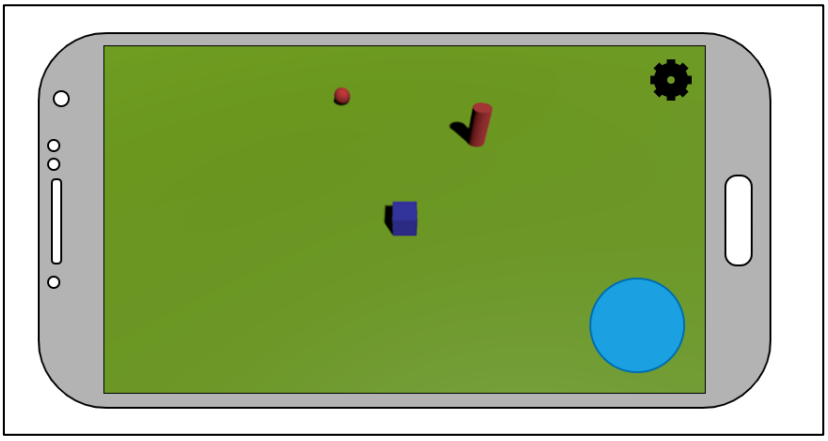


Figura IV: Mockup de la ventana principal. Fuente: Elaboración propia.

Niveles

Aunque no sea el objetivo principal del proyecto, un buen diseño de niveles es primordial si queremos mantener el interés de los jugadores. Este influye en factores muy diversos como por ejemplo la curva de dificultad o la variedad visual de los niveles. Teniendo esto en consideración, el videojuego consistirá en tres niveles:

- Un primer nivel que actuará como *lobby* o menú principal (la aldea). Dentro de esta escena el jugador podrá crear y probar los hechizos que haya creado manipulando la gramática a su antojo. También podrá recoger misiones menores para darle objetivos que cumplir.

- El segundo será el nivel principal, que actuará como escena principal que el usuario podrá explorar y donde aprenderá a combatir con los enemigos de la región mediante el uso de sus conjuros.
- La última escena consistirá en un nivel cerrado basado en el diseño de una mazmorra. La dificultad de esta instancia será más elevada respecto al segundo nivel, aunque ofrecerá mejores recompensas.