Desarrollo de una Herramienta de Autoría en Unity para la Creación de Juegos de Rol con Combate Basado en Turnos

Marcelino Pérez Durán y Juan José Prieto Escolar

FACULTAD DE INFORMÁTICA
DEPARTAMENTO DE INGENIERÍA DEL SOFTWARE
E INTELIGENCIA ARTIFICIAL
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Grado en Ingeniería del Software

Madrid, 8 de junio de 2018

Director: Prof. Dr. Federico Peinado Gil
Codirector: Víctor Manuel Pérez Colado

Autorización de difusión y utilización

Los alumnos Marcelino Pérez Durán y Juan José Prieto Escolar, junto al director de este Trabajo de Fin de Grado (TFG), el Prof. Dr. Federico Peinado Gil, y al codirector Victor Manuel Pérez Colado autorizamos a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores tanto la propia memoria de este trabajo, como el código, los contenidos audiovisuales (incluyendo imágenes de los autores), la documentación y el prototipo desarrollado.

Así mismo autorizamos a la UCM a que este trabajo sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

Fdo. Marcelino Pérez Durán

Fdo. Juan José Pietro Escolar

Fdo. Prof. Dr. Federico Peinado Gil

Fdo. Victor Manuel Pérez Colado

Agradecimientos

A nuestra familia, amigos y a toda la gente que nos ha apoyado durante todo el año. También agradecer a nuestro director del proyecto Federico Peinado Gil y codirector Victor Manuel Pérez Colado, por el apoyo y dedicación que nos han dado.

Índice general

Índice de figuras

Índice de cuadros

Resumen

En los últimos años no ha dejado de aumentar la producción de videojuegos desarrollados tanto de grandes productoras como de estudios independientes para todo tipo de plataformas Debido al gran volumen de lanzamientos, para ser competitivo y sobrevivir en este mercado es fundamental disponer de herramientas que permitan crear videojuegos multiplataforma de gran calidad y sin asumir unos costes elevados, ni en tiempo, ni en esfuerzo de desarrollo. Es cierto que existen entornos de desarrollo y herramientas muy consolidadas para crear títulos de los géneros más trabajados, como los juegos de plataformas o de disparos en primera persona, pero no ocurre lo mismo en otros géneros más complejos de abordar, como los juegos de rol, donde las herramientas escasean o son más limitadas.

Los juegos de rol se dividen a su vez en diversos subgéneros, siendo el juego de rol táctico o TRPG (del inglés *Tactical Role-Playing Game*) uno de los más afectados en cuanto a carencia de herramientas de autoría modernas orientadas a la producción multiplataforma. Este subgénero se caracteriza por estructurar sus movimientos y acciones tácticas por turnos, y también por estructurar el escenario mediante casillas, a menudo ofreciendo una perspectiva isométrica del mismo.

El objetivo de este proyecto es crear una herramienta de autoría para juegos de género TRPG sobre Unity, dado que es el entorno de desarrollo de videojuegos más popular en la actualidad. El proyecto toma como comienzo el prototipo de una herramienta anterior, que será sometida a revisión y análisis, atendiendo a los principios de la Ingeniería del Software para dar lugar a una primera versión oficial llamada TRPG Maker. Mientras que existe la necesidad del desarrollo por completo de un motor RPG táctico que maneje los diferentes aspectos de un juego RPG, el apartado gráfico se sustentará en la herramienta IsoUnity, que se conectará con la herramienta TRPG y proporcionará la gestión de escenarios, personajes y animaciones.

Las pruebas y análisis del software existente concluyeron en la necesidad de una reimplementación casi en su totalidad del código. Con esta primera versión oficial se ha pasado de una prueba de concepto a una herramienta más estable y cercana a su explotación en proyectos reales de producción de videojuegos. No sólo se ha replanteado el diseño, reconstruido la arquitectura software, y reimplementado todo el código fuente desde cero para que el proyecto sea más mantenible y ampliable, sino que además el proyecto cuenta con una demostración de las funcionalidades fundamentales de la gestión de la base de datos y el combate por turnos. Actualmente la herramienta de autoría TRPG Maker se encuentra disponible de forma íntegra, libre y gratuita en la plataforma GitHub.

Palabras clave

Desarrollo de Videojuegos, Informática del Entretenimiento, Juegos de Rol Táctico, Entornos y Herramientas de Desarrollo, Ingeniería del Software

Abstract

In the last few years the videogame production has not stopped increasing developed both large producers and independent studies for all types of platforms. Due to the large volume of launches, to be competitive and survive in this market, it is essential to have tools that allow creating high-quality multiplatform videogames without taking on high costs, neither in time nor in development effort. It is true that there are development environments and well-established tools to create titles of the most worked genres, such as the games of platforms or firing in the first person, but it is not the same in other genres more complex to address, like role-playing games, where tools are scarce or more limited.

Role-playing games are divided into different sub-genres, being the tactical role-playing game or TRPG (from English textit Tactical Role-Playing Game) one of the most affected in terms of lack of modern authoring tools oriented towards multiplatform production. This subgenre is characterized by structuring its movements and tactical actions by turns, and also by structuring the scenario using boxes, often offering an isometric perspective of the same.

The goal of this project is to create an authoring tool for TRPG genre games on Unity, becouse that it is the most popular videogame development environment today. The project takes as I start the prototype of a previous tool, which will be subjected to review and analysis, according to the principles of Software Engineering to create a first official version called TRPG Maker. While there is a need for the full development of a tactical RPG engine that handles the different aspects of an RPG game, the graphic section will be supported by the IsoUnity tool, which will connect with the TRPG tool and provide the management of scenarios, characters and animations.

The testing and analysis of the existing software resulted in the need for almost complete reimplementation of the code. With this first official version it has gone from a proof of concept to a more stable tool that is close to its exploitation in real videogame production projects.

Not only the design has been redesigned, the software architecture has been reconstructed, and all the source code has been reimplemented from scratch in order to make the project more maintainable and expandable, but also the project has a demonstration of the fundamental functionalities of the management of the base of data and turn-based combat. Currently the TRPG Maker authoring tool is available in full, free and free on the GitHub platform.

Keywords

Video Game Development, Entertainment Computing, Tactical Role-Playing Games, Development Environments and Tools, Software Engineering

Capítulo 1

Introducción

Si algo caracteriza a la Industria del Videojuego en esta última década es su incesante producción de títulos[1], ya no desarrollados únicamente por grandes productoras (las conocidas como AAA) sino también por infinidad de estudios independientes. Hoy día los videojuegos se publican simultáneamente en todo tipo de plataformas (PC, consolas, teléfonos inteligentes...) y en su mayoría cumplen con unos requisitos de calidad software, audiovisual y de diseño muy exigentes.

Los desarrolladores de videojuegos, si quieren tener éxito en este complejo sector, deben ser competitivos y para ello es necesario que consigan abaratar los costes de producción, tanto en tiempo como en esfuerzo y complejidad, de los proyectos de desarrollo software. Para esto, afortunadamente, existen herramientas que permiten crear videojuegos multiplataforma de calidad, con relativo poco esfuerzo y con relativa celeridad.

Es cierto que hoy día existen entornos de desarrollo integrados, como Unity, y herramientas muy consolidadas para crear videojuegos que pertenezcan a los géneros más populares y fácilmente atrayente para el público, como los de plataformas en 2D o los de disparo en primera persona (conocidos como *shooters*) en primera persona en 3D. Sin embargo no hay tantas herramientas ni tan potentes en otros géneros que habitualmente no eran abordados por los estudios de desarrollo de recursos económicos más humildes. Este es el caso de los juegos de rol táctico.

[1]Distribución de juegos publicados en Steam entre 2004 y 2016: `https://www.statista.com/statistics/750099/steam-games-release-annual-distribution/`

1.1. Juegos de rol táctico

El uso de la tecnología para el ocio lleva con nosotros desde hace más de 50 años. Desde que el físico William Higinbotham en la Brookhaven National Laboratory, Nueva York, inventara un juego similar al ping-pong sobre un osciloscopio [1], hasta que Nolan Bushnell, 24 de mayo de 1972, el fundador de la compañía Atari Inc, publicara "Pong", el juego que pondría en marcha la industria del videojuego, pasaron más de 10 años.

Comenzó entonces el desarrollo de plataformas domésticas para poder jugar a videojuegos, las primeras videoconsolas como la Atari 2600; con juegos tales como Lunar Lander, Space Invaders, Pac-Man y Dragonstomper (primer videojuego de rol en consola, de 1982).

El primer videojuego de rol de ordenador puede considerarse "Dungeons and Dragons", en 1980, una adaptación del famoso juego de mesa.

Los juegos de rol se dividen a su vez en diversos subgéneros, siendo el juego de rol táctico o TRPG (del inglés *Tactical Role-Playing Game*) uno de los más afectados en cuando a carencia de herramientas de autoría modernas orientadas a la producción multiplataforma. Este subgénero se caracteriza por estructurar sus movimientos y acciones, como las de combate, por turnos y por dividir el escenario en casillas, a menudo utilizando una perspectiva isométrica o similar.

El *turno* en este tipo de videojuegos es clave, ya que el modo en que se realizan las acciones hace que este tipo de videojuegos se diferencien de otros. Cada turno permite al jugador pensar como mover a su equipo por las casillas del tablero de juego. Suele ser pausado, ya que este tipo de juegos no acostumbra a penalizar el tiempo destinado a ejecutar una acción, y es posible estar el tiempo deseado meditando la jugada que se quiera realizar.

Sagas de videojuegos importantes como Final Fantasy Tactics o Fire Emblem han quedado en la memoria de los aficionados e incluso siguen publicando nuevas entregas, y juegos nuevos como Banner Saga, sin intentar cambiar las mecánicas ya definidas por los clásicos, han triunfado, dejando claro que existe un mercado para este género.

1.2. Entornos de desarrollo y herramientas de autoría

A medida que las tecnologías han ido mejorando, el sector del videojuego ha ido evolucionando con un incremento de funcionalidad, gráficos y en definitiva, ofreciendo juegos de mucha mayor complejidad. Esta complejidad se ve reflejada en mucho más trabajo para elaborar contenido y a su vez, mucho mayor coste de desarrollo.

Spacewar!, considerado por muchos el primer juego para ordenador [2], fue programado en código ensamblador [3], mientras que algunas de las videoconsolas de Atari usaban el lenguaje BASIC para programar sus juegos. Lenguajes que para realizar un videojuego es duro y costoso en tiempo y resultados.

Con estos primeros videojuegos sobre la mesa, los creadores comenzaron a desarrollar nuevas formas de juego. La popular Nintendo [4] redefinió los estándares de los videojuegos de plataformas con Super Mario Bros. [5], Sierra Entertainment [6], por ejemplo, hizo algo parecido en su época con las bases de las aventuras gráficas con King's Quest. Las empresas del sector creaban y crean sus productos, en su gran mayoría, con motores gráficos y herramientas propias, a las que el público general y los desarrolladores independientes no tienen acceso.

Desarrollar un videojuego en la actualidad para comerciar en el mercado conlleva mucho riesgo y un gran capital inicial, y sólo las empresas con mayor capacidad financiera pueden producir proyectos grandes, similar a lo que ocurre en la industria del cine. Hoy en día, estamos experimentando un irrupción de desarrolladores independientes, que crean videojuegos sin las pretensiones económicas de las grandes producciones, pero que arriesgan en lo jugable, incluso en lo narrativo o buscan nuevos horizontes en el videojuego como medio. Estas empresas pueden existir gracias a, en parte, herramientas como Unity [7], un entorno de desarrollo que ofrece una cantidad enorme de herramientas al desarrollador independiente, que tiene una versión gratuita y que facilita en gran medida todo el proceso de desarrollo de un videojuego.

Hoy día disponemos de herramientas que facilitan la creación de videojuegos según el tipo de juego que queramos realizar, aunque encontramos géneros donde hay más ayudas para el desarrollo y otros donde hay menos. En el género rol existe RPG Maker [8], una herramienta para el desarrollo de este tipo de videojuegos, aunque con resultados de una calidad gráfica muy por debajo de herramientas como Unity o Unreal Engine [9].

El desarrollo, publicación y distribución física un videojuego es un trabajo más costoso de lo que parece, pero el acceso a los videojuegos desde el punto de vista del cliente no podría ser más sencillo gracias a las plataformas de descarga en ordenador como Steam [10], GOG [11] y similares, o las tiendas digitales de las videoconsolas y de los teléfonos inteligentes, que permiten comprar y descargar todos los títulos que se desee en apenas unos pocos pasos. Esta accesibilidad ha propiciado el crecimiento de estudios independientes, haciendo que la distribución de videojuegos de determinado nicho sea algo más sencillo para pequeños estudios.

Por un lado las herramientas de autoría que facilitan la producción de videojuegos y por otro las plataformas que posibilitan la distribución de los mismos, ha ocasionado este auge por el llamado "mundo indie", del desarrollo independiente de videojuegos.

1.3. Propósito del trabajo

Este trabajo pretende contribuir a que existan herramientas funcionales útiles y completas para poder desarrollar videojuegos de rol táctico. Se busca que desarrolladores sin mucha experiencia ni amplios conocimientos en programación, puedan crear este tipo de juegos de forma sencilla e intuitiva. Para ello se propone crear una herramienta, con una importante componente visual y sin apenas escribir código, mediante la cual el creador pueda definir diversos parámetros (atributos, habilidades, objetos, armas, y similares) de una forma consistente y accesible.

La herramienta propuesta se fundamentará en el que es probablemente el entorno de desarrollo de videojuegos más conocido hoy día y que permite construir juegos para gran cantidad de plataformas, Unity, siguiendo una prueba de concepto que ya anticiparon dos alumnos de la Facultad, Javier Druet y Luis Alfonso González [12], e integrando nuestro trabajo con otra herramienta que ha sido creada aquí, en la Facultad de Informática de la Universidad Complutense de Madrid, y que cada vez cobra más relevancia: IsoUnity [13].

Una vez realizada la herramienta, consideramos importante crear una demostración, una prueba de un juego realizado íntegramente con nuestra herramienta para demostrar las capacidades de la misma.

El tipo de funcionalidades que resultan de interés para este trabajo son el manejo de personajes con distintas clases, propiedades y atributos, el uso de objetos y armas que modifiquen las propiedades de estos personajes, la gestión del inventario de dichos objetos y armas, y por último un sistema de combate por turnos que tenga una configuración por defecto, aunque luego cada desarrollador pueda modificar lo que desee.

Además, el proyecto se plantea con la intención de facilitar y permitir ampliaciones sucesivas de futuros alumnos o de cualquier otro desarrollador amante del software libre que desee contribuir.

1.4. Estructura del trabajo

Tras este capítulo introductorio, en el Capítulo 2 se expone una revisión del estado de la cuestión en el campo de los juegos de rol basados en turnos y sus herramientas de autoría. En el Capítulo 3 se detallan los objetivos y la especificación de la herramienta a desarrollar, y en el 4 se agrupa todo lo relativo al análisis, diseño e implementación del software. En el Capítulo 5 exponemos las pruebas realizadas sobre un juego de ejemplo, con los resultados obtenidos y discutimos los aspectos positivos y negativos del proyecto, concluimos en el 6 recapitulando todo lo que permite hacer actualmente la herramienta y cuales son los planes para mejorarla en el futuro.

Capítulo 2

Estado de la cuestión

Nuestro proyecto se basa en la continuidad del desarrollo del primer prototipo de TRPG Maker[1], una herramienta que hace posible crear videojuegos de rol tácticos con combate basado en turnos. Por tanto, y tomando como referencia algunos puntos marcados ya en el prototipo, nuestra revisión del estado del arte tiene cuatro pilares fundamentales:

1. Investigación a fondo de los videojuegos de referencia
2. Análisis de distintas herramientas para el desarrollo de este tipo de juegos
3. Estudio en profundidad del prototipo de TRPG Maker
4. Estudio de la herramienta IsoUnity [14], para conectar de manera desacoplada TRPG Maker con este editor de niveles

El orden de esta lista viene dado por los pasos que vamos a seguir para realizar nuestro trabajo, por lo que en este capítulo vamos a revisar algunos de los videojuegos que hemos tomado como referencia, así como herramientas y entornos de desarrollo para encontrar requisitos para nuestro proyecto. Por otra parte vamos a realizar el estudio de la herramienta original de la que trata este proyecto, TRPGMaker, para conocer el estado de la herramienta y las mejoras necesarias tras la búsqueda de requisitos de los puntos anteriores. También se estudiará y probará la herramienta IsoUnity, que ha sido utilizada por la primera versión de TRPGMaker para la creación de escenarios y gestión de eventos, y así poder conocer su potencial y el modo en el que ha sido utilizado anteriormente.

[1]Prototipo TRPG Maker - `https://github.com/Narratech/TRPGMaker/tree/old-version`

2.1. Videojuegos de referencia

En esta sección nos centramos en los distintos videojuegos que tomaremos como referencia para la realización de este proyecto, seleccionando aquellos videojuegos más importantes dentro del género, así como títulos actuales que han triunfado en la industria.

2.1.1. Final Fantasy Tactics

La saga Final Fantasy Tactics ha sido un referente importante para este trabajo. Se compone por títulos tales como *Final Fantasy Tactics* [15], *Final Fantasy Tactics Advance* [16], *The War of The Lions* [17] y *Advance 2: Grimoire of the Rift* [18]. Podemos decir que fue la saga tomada como referencia principal por los autores del prototipo de TRPG Maker, muchas de las funcionalidades de este videojuego quedaron sin implementar, por lo que son el objetivo inicial de nuestro proyecto.

SquareSoft es la empresa que creo esta franquicia, en 1997, siendo estos videojuegos desarrollados por el equipo que surgió de una fusión de las empresas Square y Enix[19]. Esta saga de videojuegos se define por pertenecer al género RPG táctico, con una ambientación de fantasía. Dispone de una serie de características que nos resultan básicas para este tipo de software.

Figura 2.1: Opciones de batalla en Final Fantasy Tactics

De las funcionalidades de Final Fantasy Tactics cabe destacar:

1. **Turnos**. En este videojuego y siendo una característica básica en todo juego de rol táctico, el tiempo se gestiona mediante turnos.

   Un turno es el orden en que van sucediéndose o alternándose el momento de actuación de cada personaje para realizar determinadas acciones.

   En los videojuegos de rol clásicos como Final Fantasy VII, desarrollado y publicado por Square originalmente para la plataforma PlayStation, el turno existe para permitir a los personajes realizar ataques, habilidades, usar objetos o defender en un determinado tiempo.

   En Final Fantasy Tactics, al igual que en la mayoría de los videojuegos de rol tácticos, el turno permite a cada personaje atacar, defender, usar habilidades, usar objetos y, como diferencia de otros géneros de rol, también moverse por el escenario. El turno acaba cuando el jugador así lo decide, seleccionando *esperar*. En este momento se permite al jugador seleccionar la dirección en la que el personaje va a mirar hasta su próximo turno, haciendo que los ataques enemigos sobre este personaje sean más o menos poderosos en función de la trayectoria y la dirección.

2. **Combates**. El sistema de combate se basa en la posición, el turno (anteriormente descrito), habilidades y atributos. Los combates se disputan en un terreno de juego acotado, con diferentes alturas, y dividido en cuadriculas donde los personajes están situados. Cada personaje en el terreno de juego dispone de un turno para realizar las siguientes acciones[2]:

   a) Mover. Al seleccionar esta opción aparecen dibujados con un color azulado en el terreno de juego las posiciones a las que el personaje puede moverse (véase Figura 2.2), seleccionando una de estas posiciones, el personaje va en esa dirección. Al realizar el movimiento, aparecerán de nuevo las acciones que puede realizar, excepto la de movimiento.

   b) Actuar: Al seleccionar esta opción aparecen diversas posibles acciones:

[2]Manual Final Fantasy Tactics - `http://www.gamesdatabase.org/Media/SYSTEM/Sony_Playstation/Manual/formated/Final_Fantasy_Tactics_-_1998_-_Sony_Computer_Entertainment.pdf`

1) Atacar. Al seleccionarlo, aparecerán dibujado en el terreno de juego con un color rojizo el área en el que el personaje puede atacar, al seleccionar un enemigo que se encuentre dentro del área el personaje atacará al seleccionado con su ataque básico, éste perderá vida si el ataque es exitoso.
2) Habilidades. Las habilidades son ataques especiales que puede realizar un personaje. Dependiendo de los trabajos[3], los objetos equipados y los poderes mágicos del personaje, aparecerán diferentes habilidades para utilizar.

c) Esperar. Al seleccionar esperar, el personaje pasará el turno al siguiente.

d) Estatus. Al seleccionarlo, aparecerá en pantalla información del personaje, sus atributos e información sobre el mismo.

e) Auto atacar. Permite realizar de forma automática el turno, dejando una serie de prioridades.

Figura 2.2: Casillas de movimiento en Final Fantasy Tactics

[3]Característica de Final Fantasy Tactics que otorga unas habilidades comunes y unas específicas.

Al finalizar el turno de combate de cada personaje, este gana una serie de puntos de experiencia y puntos de trabajo si ha realizado una acción exitosa. Estos puntos pueden utilizarse posteriormente para mejorar sus habilidades o cambiar de oficio y aprender habilidades especiales exclusivas del mismo.

2.1.2. Fire Emblem

La franquicia *Fire Emblem* es otro importante referente en el género RPG táctico. La serie consta de 15 juegos, hasta el momento, y 3 futuros anunciados. La saga ha visto la luz en la mayoría de consolas de Nintendo (NES, SNES, Game Boy Advance, GameCube, Wii, Nintendo DS y Nintendo 3DS) además de dispositivos móviles con *Fire Emblem Heroes* [20]. *Fire Emblem: Rekka no Ken* [21], el séptimo juego de la franquicia, se convirtió en el primer juego de la saga en lanzarse internacionalmente, presentado fuera de Japón como *Fire Emblem*.

De las funcionalidades de *Fire Emblem* cabe destacar:

1. **Modo de juego**. Consiste en un mapa en el que, por turnos, el jugador maneja sus unidades/personajes con el objetivo de derrotar a los enemigos para cumplir con la misión del capítulo.

2. **Misiones**. Las misiones consisten en batallas tácticas en la que deberemos tomar un castillo, sobrevivir un cierto número de turnos el ataque enemigo, acabar con todos los enemigos del mapa o derrotar a un jefe.

3. **Sistema de combate**. Las armas tienen una ventaja frente a algunas y desventaja frente a otras, de modo que, por ejemplo, la lanza derrota a la espada, la espada derrota al hacha, y el hacha tiene ventaja sobre la lanza. En Fire Emblem también hay ataques mágicos, y con ello un triángulo de ventajas y desventajas de magia similar al de las armas.

4. **Muerte**. Los personajes de Fire Emblem no son inmortales, si una unidad muere en combate, no podrá volverse a usar en combate y desaparecerá del mapa, pero la partida podrá continuarse. Este modo de muerte funciona con toda unidad, excepto con los personajes principales, que si mueren, la partida habrá acabado, teniendo que volverse a jugar el capítulo.

5. **Capacidad de movimiento y alcance** Una unidad tiene un máximo de espacios que se puede desplazar por turno. La distancia de movimiento se muestra en azul. Los bosques y las montañas reducen la capacidad de desplazamiento. En algunos mapas puede haber lluvia o nieve durante algunos turnos, ante estos climas la capacidad de movimiento también se ve limitada.

6. **Dinero**. El jugador consigue dinero a lo largo de la historia derrotando enemigos o vendiendo equipaciones que no necesite, este dinero se puede usar para comprar armas y objetos especiales de las tiendas.

7. **Información de ataque**. El jugador puede visualizar una estimación de cuantos puntos de vida sufrirá y cuantos ocasionará a su rival si le ataca en combate, así como la probabilidad de acierto de ataque (Figura 2.3).

Figura 2.3: Pantalla con información antes de confirmar un ataque.

8. **Interacción social**. El jugador podrá visitar pueblos, tener conversaciones con otros personajes, intercambiar objetos entre personajes y adquirir puntos de experiencia en cada lucha para aumentar el nivel y las características de cada unidad.

2.1.3. Otros referentes

Los títulos anteriormente analizados fueron los más importantes y de los que más requisitos para la herramienta se han sacado, pero existen varios videojuegos que también fueron probados y analizados para conocer las necesidades de desarrollo para crear videojuegos TRPG, en esta subsección se mencionan y explican estos videojuegos.

The Banner Saga[22] es un videojuego táctico de ambientación vikinga. El juego cuenta con una historia que se desarrolla dando lugar a batallas por turnos en el que el jugador puede formar y luchar con un equipo de combate con diversas habilidades. Fue lanzado en 2014, siendo un título que comenzaba una nueva saga, por un equipo de desarrollo muy pequeño. Este título mantiene las mecánicas clásicas, pero dotando al juego de una gran dificultad, por lo que es importante utilizar el turno de los personajes de forma correcta.

Figura 2.4: Una conversación en *Shadowrun Returns*

Shadowrun Returns[23] combina los combates de rol táctico basado en turnos con la exploración de escenarios y la posibilidad de interactuar con otros personajes como en juegos RPG (Figura 2.4), abriendo el camino a un modo de juego mucho más narrativo, una característica en la que queremos trabajar para poder usar la funcionalidad de combate por turnos de la herramienta TRPG Maker con los entornos de exploración y RPG clásicos que permite crear IsoUnity, permitiéndonos tener un buen ejemplo de como realizar las intersecciones entre modo táctico/combate y modo exploración.

Advance Wars [24] es un videojuego para la consola de Nintendo Gameboy Advance. Se trata de un videojuego bélico en el que el jugador controla un ejercito, que se enfrenta a otro controlado por el enemigo rival[4]. En cada turno el jugador puede mover sus tropas por el mapa, limitando a un movimiento por unidad (también puede atacar si entra en el rango de ataque una vez movido). El jugador tiene información sobre sus tropas y las tropas rivales si se posiciona con el cursor encima de la tropa (Figura 2.5), tal como vida restante, rango de ataque, rango de movimiento y terreno en el que se encuentra. El ataque es más o menos efectivo dependiendo del terreno en el que se encuentre la unidad que ataca y la unidad a atacar.

Figura 2.5: Información previa al ataque en *Advance Wars*

Una vez acabado el turno pasa a jugar el siguiente jugador, y cuando ambos jugadores han terminado, empieza un nuevo día. Este contador de días es utilizado para lograr determinados objetivos dentro del combate.

Cada escuadrón en el terreno de juego tiene una utilidad: las unidades de infantería pueden capturar edificios, las de transporte pueden llevar otras unidades, y generalmente suministrar combustible y munición [5]

El jugador gana la partida cuando acaba con todas las tropas enemigas o conquista el cuartel general del enemigo.

[4]Game Design Review: Advance Wars - Dual Strike `http://www.lostgarden.com/2005/09/game-design-review-advance-wars-dual.html`

[5]Artículo de Advance Wars en Wikipedia: `https://es.wikipedia.org/wiki/Advance_Wars`

La saga *Diablo*[25] no pertenece al género TRPG, pero si cuenta con muchas características importantes en el género rol. En Diablo, un personaje esta formado de varios huecos, como puede verse en la Figura 2.6, donde cada hueco es una parte del cuerpo del personaje, y puede añadirse armas y objetos a estos huecos para aumentar los *atributos* del personaje. En este juego, se debe elegir una clase, esta clase hará que el personaje tenga unas u otras habilidades. La clase también influye en los atributos, ya que unos tendrán más valor que otros en función de la clase escogida.

Figura 2.6: Inventario del jugador en *Diablo III*

Las habilidades son muy variadas unas de otras, existiendo habilidades en área, a objetivo o pasivas (habilidades que mejoran características del propio personaje). Las habilidades en Diablo suelen ser ofensivas, a excepción de las pasivas.

2.2. Herramientas para el desarrollo de videojuegos

En esta sección se revisan una serie de utilidades pasa el desarrollo de videojuegos, algunos entornos de desarrollo genéricos como Unity y otras herramientas más particulares para el género del rol, como RPG Maker[8]. El propósito de este análisis es conocer las características comunes de necesaria inclusión para este tipo de herramientas.

2.2.1. Unreal Engine

Unreal Engine [9] es un entorno de desarrollo para videojuegos creado por Epic Games. Desde el año 2015 cambió su modelo de negocio, pasando a ser de acceso gratuito y cobrando exclusivamente un porcentaje de regalías[6], haciéndolo aun más accesible para pequeños estudios y cualquier persona que quiera iniciarse en los videojuegos.

Las características más importantes de Unreal son las siguientes[7]:

- **Potencia**. La potencia y calidad gráfica que ofrece su motor es de las más impresionantes del mercado. Cuenta con un gestor de partículas y de iluminación que con poco esfuerzo permiten obtener una gran calidad en el apartado visual de un videojuego.
- **Comunidad**. Unreal Engine cuenta con muchos seguidores y es de las herramientas más longevas de la industria, por lo que existe una gran comunidad. En los últimos años ha mejorado mucho su curva de aprendizaje y hoy día existe una buena cantidad de tutoriales y foros dedicados a este entorno de desarrollo.
- **Multiplataforma**. Permite exportar el proyecto a las principales plataformas sin apenas tener que realizar modificaciones.

[6]Unreal Engine 4 pasa a ser gratuito: `https://www.unrealengine.com/en-US/blog/ue4-is-free`

[7]Análisis de Unreal Engine, por Carlos Coronado - `http://www.zehngames.com/developers/unreal-engine-4-herramienta-versatil-de-desarrollo/`

- **Extensiones**[8]. Unreal Engine permite la creación de extensiones para su herramienta, dando la posibilidad de agregar funciones nuevas y modificar la funcionalidad incorporada, ya sea modificando o sin modificar el código abierto del motor.

En definitiva, Unreal Engine es una herramienta muy potente, y razonablemente usable y extensible. Un punto negativo para este entorno de desarrollo es que requiere un equipo muy potente para poder trabajar con él y está pensado para usar un flujo de trabajo muy profesional, orientado a equipos de tamaño medio o grande. Podría ser la base para la herramienta TRPG Maker, pero el desconocimiento de las APIs internas de Unreal Engine y el tiempo que se debería dedicar para desarrollarla como extensión de este entorno ha generado que se opte por otra herramienta más accesible.

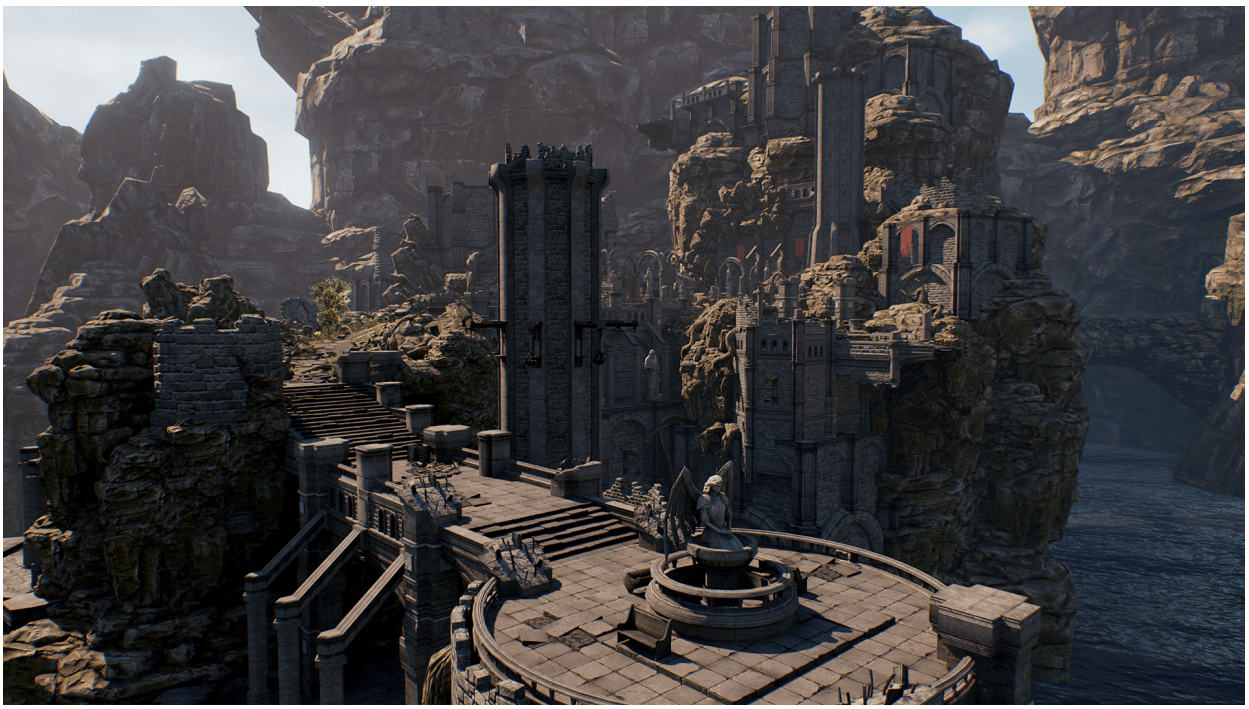


Figura 2.7: Escenario mostrando las posibilidades gráficas de *Unreal Engine 4.9*.[9]

[8] Extensiones para Unreal Engine - `https://docs.unrealengine.com/en-us/Programming/Plugins`

[9] *Infinity Blade: Grass Lands* (Epic Games) - `https://www.unrealengine.com/marketplace/infinity-blade-plain-lands`

2.2.2. Unity

Unity [7] es un entorno de desarrollo de videojuego multiplataforma creado por Unity Technologies. Se encuentra disponible como plataforma de desarrollo para Microsoft Windows, OS X y Linux. Cuenta con una de las mayores comunidad de desarrolladores independientes del mundo[10] y mucha documentación sobre su uso y sobre su expansibilidad.

Las características más destacables de Unity son las siguientes:

- **Potencia**. Unity no cuenta con la misma potencia gráfica que Unreal Engine, pero aun así tiene una calidad muy buena. Cuenta con gestores de partículas y de iluminación potentes.
- **Comunidad**. Ya se ha mencionado anteriormente, pero la comunidad de Unity es muy grande, existiendo gran cantidad de material y tutoriales, así como foros de discusión y ayuda sobre la herramienta.
- **Multiplataforma**. Permite exportar el proyecto a una gran cantidad de plataformas sin apenas realizar modificaciones en el proyecto.
- **Extensiones**[11]. Unity proporciona clases para realizar extensiones de su editor, para agregar nuevas funcionalidades o modificar las funcionalidades existentes en la herramienta.

En conclusión, Unity cuenta con una gran comunidad y mucho material para conocer a fondo la herramienta, es gratuita hasta superar un umbral de beneficios y es extensible. Teniendo en cuenta que la primera versión de TRPG Maker está implementada íntegramente sobre Unity, el conocimiento y estudio previo de esta herramienta ha sido crucial para poder realizar el proyecto. Al haber sido el prototipo desarrollado previamente con Unity, ahora resulta más sencillo continuar con su utilización.

[10] Comunidad de unity - `https://unity3d.com/es/community`
[11] Extensiones para Unity - `https://docs.unity3d.com/Manual/Plugins.html`

Figura 2.8: Decorado de un escenario mostrando las posibilidades gráficas de *Unity 2018*.[12]

2.2.3. IsoUnity

IsoUnity [13] es una herramienta desarrollada por los hermanos Pérez-Colado, siendo una extensión de Unity que permite la creación de escenarios con perspectiva isométrica. Cuenta con un gestor de eventos, entidades y componentes.

IsoUnity, desarrollada en nuestra Facultad, permite la creación de pequeñas aventuras isométricas, en estilo retro, que transcurren en escenarios divididos en casillas, siendo características que han coexistido históricamente en el género siendo especialmente destacable por ello la saga Final Fantasy Tactics.

En IsoUnity cabe destacar los siguientes aspectos:

[12]Ejemplo de renderizado de Unity 2018: `https://unity3d.com/srp`

- **Creación de mapas**. Consiste en colocar casillas de manera ordenada en una rejilla en el eje horizontal. Estas celdas o casillas colocadas en el mapa son únicas, sin solapamientos entre ellas, permitiendo añadir información a cada celda de manera independiente. Todas las celdas mantienen el mismo ancho, pero puede variarse la altura.

- **Pintar las celdas del mapa**. La herramienta permite pintar las caras de las celdas creadas, permitiendo colocar diferentes texturas sobre las caras de las celdas, usando el ratón del ordenador a modo de pincel.

- **Gestión del juego**. La gestión del juego es gestionada por una clase llamada *Game*, que proporciona a los gestores existentes, actualizaciones y eventos en tiempo de ejecución. El gestor de mapas permite distribuir eventos y actualizaciones a entidades.

- **Eventos**. Cada entidad que se encuentra en el mapa permite el envío y recepción de eventos, con una comunicación en tiempo real, lo que posibilita la acción en el juego de forma constante.

- **Controlador del juego**. El controlador del juego permite a las entidades del mapa recibir información de controles físicos (y táctiles) y así poder realizar movimientos, gracias a su vez, al gestor de eventos, que recibirá el evento de *movimiento* o *acción* mediante el controlador del juego.

Existen otras herramientas similares a IsoUnity como *Isometric Builder*[13], que permiten realizar mapas isométricos dividiendo el escenario en cuadrículas, pero se mantendrá IsoUnity como herramienta para la creación de escenarios y gestión de eventos de la nueva versión de TRPG Maker debido a disponer como codirector de este trabajo a uno de los dos autores originales de IsoUnity.

[13]Página de compra de Isometric Builder en Unity Asset Store: `https://assetstore.unity.com/packages/tools/sprite-management/isometric-builder-98848`

Sin embargo se intentará no *depender* de IsoUnity para poder utilizar TRPG Maker, ya que al ser un proyecto universitario no garantiza futuras revisiones ni su continuidad, por lo que es necesario primero plantear la herramienta TRPG Maker como herramienta única, y contar con IsoUnity como un posible *add-on*, para poder generar una demostración de como integrarlo con otras herramientas.

Figura 2.9: Captura de *Cloister Secrets* desarrollado con *IsoUnity*[14]

[14]Videojuego experimental desarollado con IsoUnity - `http://narratech.com/en/cloister-secrets/`

2.2.4. RPG Maker

Esta herramienta ha sido la que más ha influido en el desarrollo del proyecto, imprescindible para conocer las necesidades de los usuarios y para orientar el diseño de nuestra herramienta. La base de datos, analizada en las siguientes lineas, es posiblemente una de nuestras principal referencia.

RPG Maker es una intuitiva herramienta que nació con la idea de que todo el que quisiera pudiera crear un videojuego de rol. No es necesario saber ningún tipo de lenguaje de programación. Para usar la herramienta simplemente tenemos que seguir instrucciones que aparecen en pantalla.

Han publicado diversas versiones de RPG Maker a lo largo de su historia. De entre todas las versiones existentes se ha estudiado la versión RPG Maker VX Ace [26]. La herramienta no está disponible de forma oficial en español, por lo que algunos de los términos usados estarán escritos en inglés.

La herramienta incluye un editor de mapas, eventos y base de datos que nos proporciona todos los recursos necesarios para crear el videojuego que se desee. Además, existen comunidades online que ofrecen ayuda a los usuarios de la herramienta en sus creaciones, comparten componentes gráficos, sonidos, fondos y archivos para crear videojuegos y una amplitud de tutoriales para conocer su funcionamiento.

Los mapas son una parte fundamental de la herramienta, es el lugar en el que se va a desarrollar la acción y nos vamos a mover. No hemos probado en profundidad está parte de la herramienta porque queda fuera del alcance del proyecto.

La base de datos de RPG Maker se divide en pestañas [27] [28], incluyendo:

1. **Actores**

   La pestaña de actores contiene los datos de los personajes que pueden convertirse en miembros del equipo en el juego.

Los actores, o personajes principales, son la parte esencial de un videojuego de rol, recayendo sobre ellos la historia y siendo el vínculo entre el jugador y el mundo del juego. Aquí se definen sus características, atributos, se le da un nombre y se define un primer equipo, o equipo básico que tendrá el personaje en un primer momento.

2. **Clases**

   La pestaña de clases le permite modificar lo que hace que cada clase sea diferente entre sí, incluyendo la curva de dificultad en que sube de nivel, qué atributos se mejoran, y qué habilidades se aprenden al alcanzar ciertos niveles.

3. **Habilidades**

   La pestaña de habilidades es donde se juntan todos los datos para hacer cualquier tipo de ataque mágico o especial que se pueda realizar. Las habilidades son las distintas destrezas, normales o especiales, que los personajes usarán para provocar algún tipo de efecto en enemigos, aliados o uno mismo. Las habilidades pueden ser un simple ataque físico, magia ofensiva, magia curativa o ataque especial. Se deben especificar una serie de propiedades:

   - Descripción
   - Tipo de Habilidad
   - Coste de MP (puntos mágicos, de sus siglas Magic Points)
   - A quien afecta la habilidad, sea a uno, a todos o a varios enemigos al azar, o a uno, a todos, o a los aliados que tengan vida 0, sea uno concreto, todos los que tengan esa vida o hasta el propio personaje que lanza la habilidad.

   Además se puede especificar el resultado de la habilidad al usarse con una serie de formulas y operaciones, como puede verse en la parte superior derecha de la Figura 2.10.

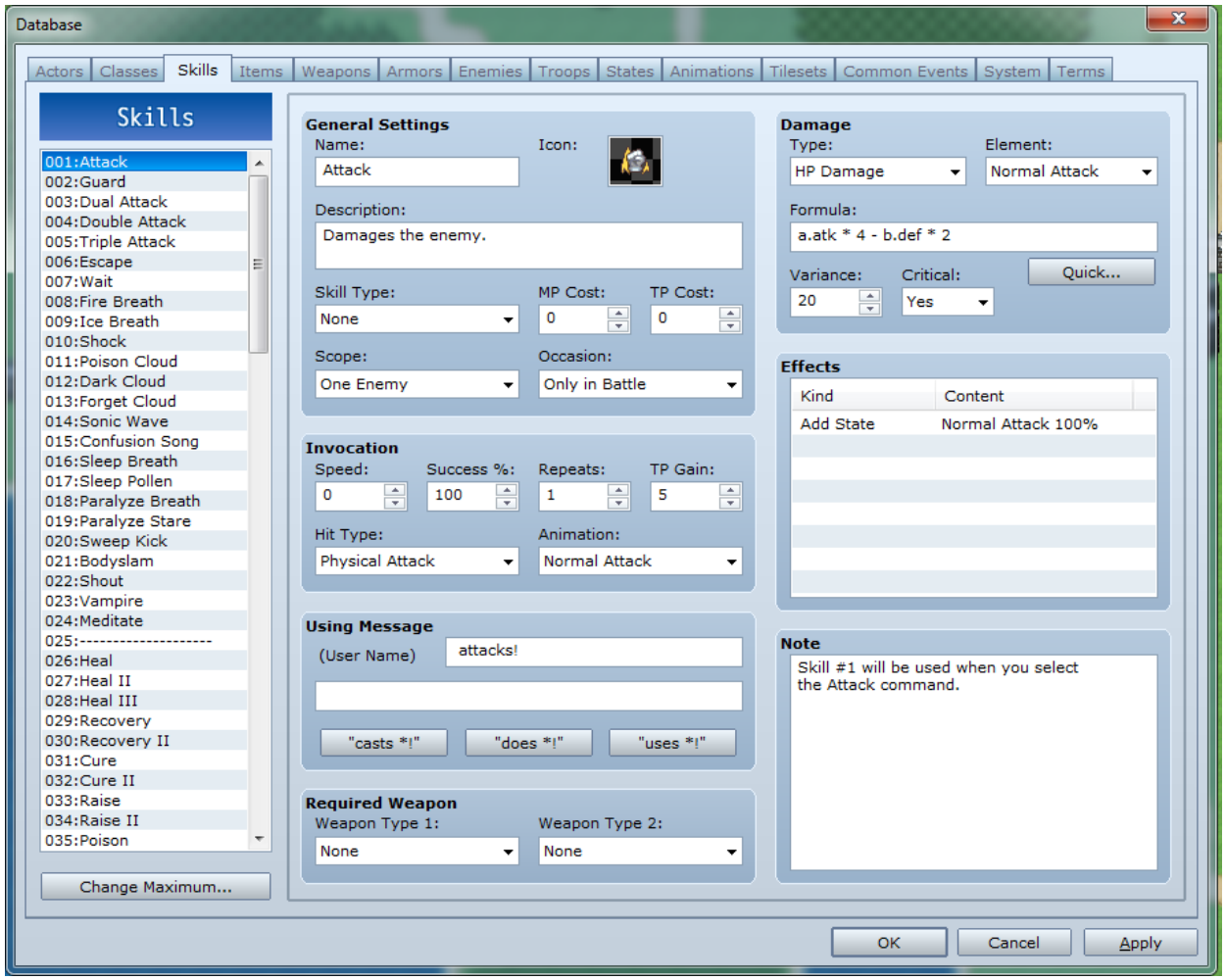


Figura 2.10: Pestaña de habilidades de la base de datos de *RPG Maker*

4. Objetos, Armas y Armaduras

La pestaña de objetos contiene los datos que determinan que sucederá al usar los objetos, los cambios que otorgan a cada atributo.

La pestaña de armas (véase Figura 2.11) contiene los datos de todas las armas del juego y es donde puedes agregar más.

La pestaña de armaduras es muy similar a la pestaña de armas, pero encontrándose aquí el total de las armaduras que existan en el juego.

La configuración general de objetos, armas y armaduras sigue el siguiente esquema:

- Nombre
- Descripción
- Tipo de arma/armadura/objeto

  El tipo de arma al que pertenece, para definir que tipo de personaje puede usarla.
- Cambios en los parámetros

  Aquí se definen que atributos o estadísticas cambian de valor al equiparse el objeto, arma o armadura un personaje

5. **Enemigos**

   La pestaña de enemigos es donde se define todas las estadísticas, recompensas y acciones de los enemigos. Los enemigos son todas las criaturas que existen con la intención de eliminar a los protagonistas/actores del juego, se engloban desde los monstruos más básicos hasta el jefe del final de juego.

6. **Tropas**

   La pestaña de tropas contiene los datos de todos los posibles diseños de grupos de enemigos para los combates.

7. **Estados**

   La pestaña de estados contiene una serie de estados que provocan cambios en los atributos de los personajes.

   Los estados son efectos especiales provocados por un objeto o magia, con el que un miembro del equipo, el equipo entero o un enemigo o grupo entero de enemigos ve alguno de sus atributos modificados momentáneamente.

   En esta pestaña se definen estos efectos, pudiendo definir que atributos cambian y durante cuanto tiempo.

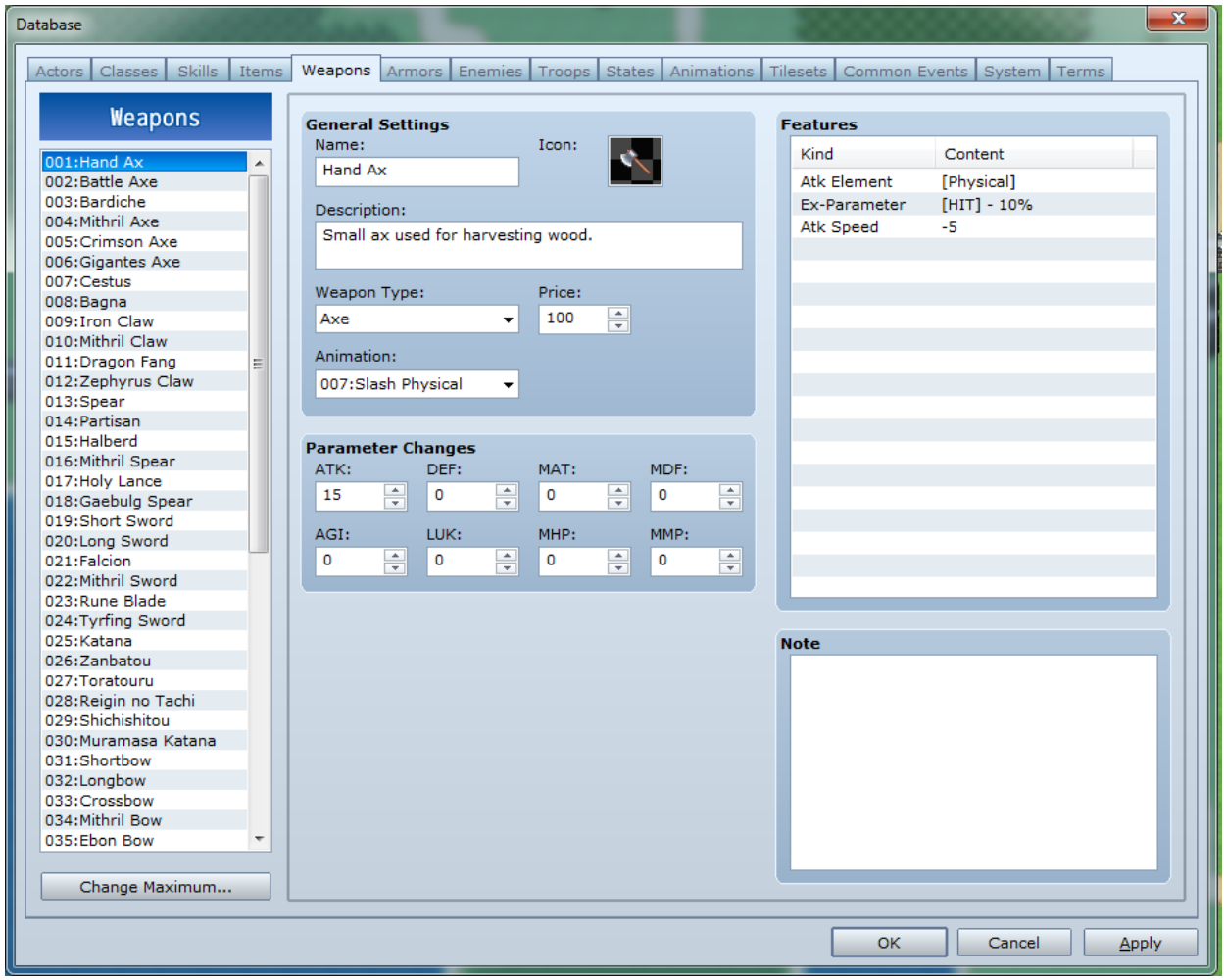


Figura 2.11: Pestaña de armas de la base de datos de *RPG Maker*

Cada pestaña anterior esta diseñada para que el usuario pueda añadir, eliminar y modificar cualquier dato de la base de datos, que originalmente aparecerá vacía excepto en alguna pestaña.

2.2.5. Prototipo TRPG Maker

El prototipo de TRPG Maker es el proyecto original sobre el que se basa este trabajo. Se creó como una extensión de Unity, muy ligada a la herramienta IsoUnity en realidad, con la que poder crear videojuegos de rol tácticos de una forma más sencilla que programándolo desde cero.

El prototipo creado por los alumnos Javier Druet y Luis Alfonso González [12] ha servido para tener una primera idea de la herramienta que se desea obtener en este proyecto.

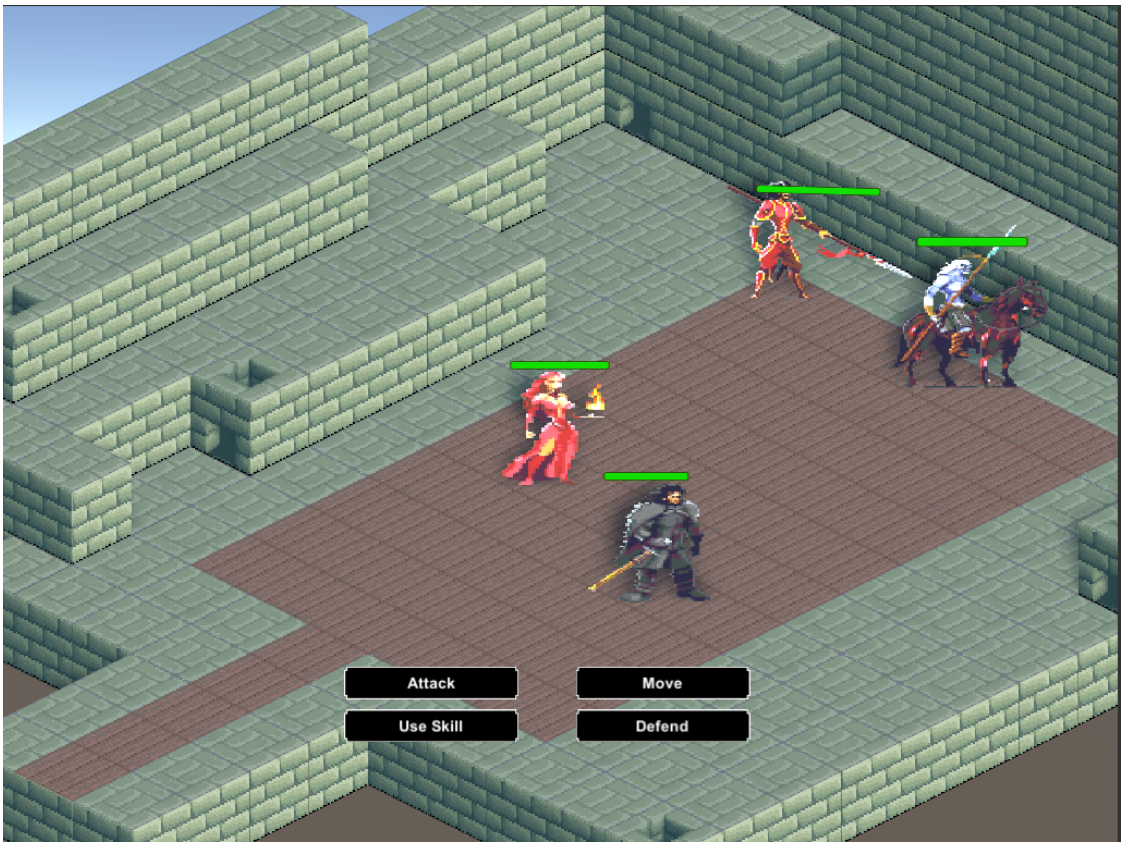


Figura 2.12: Ejemplo de combate en el prototipo de *TRPG Maker*

En este proyecto se definen y se realiza una primera implementación de los conceptos básicos necesarios para la herramienta, tales como: Base de datos, personajes, atributos, etiquetas, fórmulas, objetos, clases y habilidades.

La conexión con IsoUnity en este prototipo es muy intrusiva y está muy acoplada, usando una gran cantidad de métodos de IsoUnity en las propias clases de TRPG Maker.

El sistema de combate de este primer prototipo cuenta con el concepto de turno, estado definido en cada turno: mover, atacar, uso de habilidades y pasar turno. No existe la posibilidad de ser atacado por el enemigo.

El objetivo principal de nuestro proyecto era continuar y expandir la herramienta, pero debido a la escasa funcionalidad, la baja mantenibilidad, la nula documentación, y lo muy acoplado que se encontraba con la herramienta a IsoUnity, se decidió partir de cero, tomando como referencia la idea y conceptos iniciales, pero recreando la herramienta basándose en un diseño consistente y bien definido.

Capítulo 3

Objetivos y especificación

El objetivo principal de este proyecto es desarrollar una herramienta que permita realizar videojuegos de rol táctico por turnos, en Unity.

En los videojuegos de rol tácticos o TRPG, el jugador controla a un grupo de personajes que ocupan posiciones en un terreno dividido en cuadriculas, generalmente, y que se enfrenta a otro grupo de personajes en combate. La gestión del tiempo se realiza por turnos, en los que se puede realizar un número limitado de acciones, siendo habitual las acciones atacar, mover y el uso de habilidades. Los combates finalizan al vencer, perder, empatar o también si se produce alguna condición especial.

3.1. Objetivos

Los objetivos generales del proyecto a la hora de desarrollar la herramienta *TRPG Maker* son los siguientes:

- Aprender todo lo posible sobre los videojuegos de rol táctico y sobre las herramientas que facilitan su desarrollo
- Analizar las posibilidades y las características internas del prototipo TRPG Maker y la herramienta IsoUnity

- Desarrollar una primera versión oficial de la herramienta TRPG Maker, mejorando el diseño original, añadiendo un nuevo sistema de combate basado en turnos y probándolo con usuarios reales

- Crear una escena de ejemplo jugable que ilustre la utilidad y las capacidades de esta herramienta

En las próximas subsecciones plantearemos la metodología y plan de trabajo que se ha seguido, así como los requisitos necesarios para lograrlo.

3.2. Plan de trabajo

Para la realización del proyecto se ha necesitado adquirir conocimiento sobre el género del que trata el trabajo, los juegos TRPG. Ha sido necesario un primer contacto con las herramientas que iban a ser usadas en este proyecto, Unity e IsoUnity, con su respectivos periodos de aprendizaje.

En esta primera fase, solapada con el estudio y análisis del prototipo preexistente dio lugar a la creación de dos repositorios diferentes en GitHub donde se realizaron en uno, pruebas de IsoUnity[1] y en otro, pruebas del prototipo de TRPG Maker[2]

La organización del trabajo se acordó el día 26 de octubre de 2017, quedando fijado desde esa primera reunión las herramientas que se usarían en el proyecto.

1. Se usará la plataforma Overleaf[3] para escribir la memoria en el sistema de composición de textos LaTeX[4].

2. Para el mantenimiento del proyecto se usará un sistema de control de versiones basado en Git, en concreto la plataforma GitHub[5].

[1]IsoTest: `https-//github.com/WyrnCael/IsoTest`
[2]TRPGTest - `https://github.com/mperez01/TRPGTest`
[3]Overleaf - `https://www.overleaf.com/`
[4]LaTeX - `https://www.latex-project.org/`
[5]GitHub - `https://github.com/`

3. Se usará la herramienta Slack[6] para la comunicación entre los miembros del equipo, el director y co-director.

4. Se usará la herramienta administración de proyectos Trello[7] para precisar y conocer el estado de los objetivos.

5. Se usará el servicio Google Drive proporcionado por la Universidad Complutense de Madrid para la redacción de textos, prototipos y presentaciones.

Como recursos físicos para este proceso se necesitarán ordenadores con sistema operativo Windows, Mac o Linux que puedan ejecutar correctamente Unity. Se usará el entorno de desarrollo Visual Studio[8] para el desarrollo de los scripts necesarios para la herramienta.

Se estableció también reuniones cada 2 semanas con los miembros del equipo y el director del proyecto. En estas reuniones se establecerían objetivos que deberían ser mostrados en la siguiente reunión.

Se marcaron cuatro hitos principales en el calendario, en los que se debería exponer a todos los miembros del grupo de investigación Naratech[9] el estado del proyecto en una presentación de 10 minutos, así como fijar una serie de objetivos de cara al siguiente hito. Las fechas y objetivos de estos cuatro hitos se enumeran a continuación:

1. Reunión pre-Navidad (21/12/17)

   El objetivo para esta reunión será explicar el proyecto, así como *tantear* una primera versión ejecutable de la herramienta

2. Reunión pre-Semana Santa (22/03/18)

   El objetivo para esta reunión será poseer una versión avanzada del proyecto, logrando tener una versión ejecutable de la herramienta con funcionalidades maduras.

[6]Slack - `https://slack.com/`
[7]Trello - `https://trello.com/`
[8]Visual Studio - `https://www.visualstudio.com/es/`
[9]Narratech Laboratories - `http://narratech.com/es/`

3. Prueba de código (3/04/18)

   El objetivo en esta fecha será permitir el uso de la herramienta TRPG Maker de forma abierta, con intención de recibir opiniones sobre la misma. Por tanto, para esta fecha se deberá contar con una versión estable de la herramienta, que permita realizar todas o la mayoría de las funcionalidades finales para el proyecto.

4. Reunión pre-Verano (25/05/18)

   El objetivo para esta última reunión será poseer una versión final del proyecto, junto con una presentación lo más semejante a la que se usará en la presentación final.

3.3. Metodología y herramientas

En esta sección se tratará sobre la metodología utilizada en este proyecto, así como de las herramientas de las que se ha hecho uso a lo largo de todo el trabajo.

3.3.1. Metodología

La metodología utilizada para este proyecto esta basada en el *desarrollo ágil*, marcando pequeños objetivos en *sprints* de 2 semanas para poder llegar a las grandes metas de las cuatro fechas mencionadas anteriormente.

Al comienzo del proyecto, se trató de poner en práctica un modelo iterativo e incremental pero más tradicional [29], pero en el análisis de requisitos de la herramienta que se iba a reformar en este proyecto, se decidió reescribirla por completo debido a lo poco documentada y muy ligada a la herramienta IsoUnity que estaba, entre otros motivos. Por ello se decidió cambiar la metodología a una que permitiera minimizar los riesgos y estar en constante contacto con los clientes, en este caso, el director y el co-director del proyecto. Para ello se eligió como referencia la metodología ágil [30] *Scrum*. Esta metodología permite tener una gran capacidad de reacción para realizar cambios en los requisitos, que fueron constantes debido a la reestructuración profunda que sufrió la herramienta TRPG Maker.

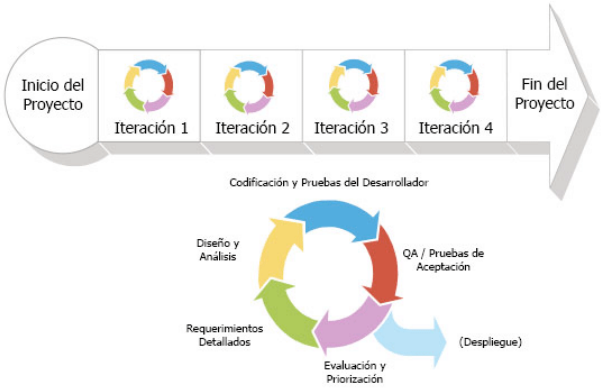


Figura 3.1: Marco de trabajo *Scrum* utilizado

El proceso de metodología ágil ***Scrum*** se siguió, con algunos diferencias respecto a su uso 'canónico', tomando como *sprints* las reuniones cada 2 semanas con el director y co-director del proyecto.

En las reuniones se definían las funcionalidades deseadas, se comprobaba el estado del proyecto y se iban añadiendo nuevas funcionalidades a medida que se realizaban reuniones. Las funcionalidades pedidas en cada *sprint* se trabajaban durante el 'sprint' y se mostraban en la siguiente reunión.

3.3.2. Herramientas utilizadas

Como se explicaba en la Sección 3.2 hemos utilizado las siguientes herramientas para organizar el trabajo:

LaTex y Overleaf

Overleaf es una herramienta de escritura y publicación en línea de LaTeX que permite el proceso de escritura, edición y publicación de documentos científicos. Se utilizará dicha herramienta online para la redacción de la memoria, permitiendo la edición colaborativa y en tiempo real del documento, así como generando un diseño profesional y elegante al texto.

GitHub

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Ha sido utilizada para albergar todo el código del proyecto, así como para mantener un control de versiones de la herramienta. El código del prototipo TRPG Maker se ha incluido en una *rama* aparte del proyecto en GitHub, para conservarlo intacto por razones históricas.

La herramienta que vamos a realizar se encuentra en su totalidad en la rama master del repositorio: `https://github.com/Narratech/TRPGMaker`.

Google Drive

Google Drive es el servicio de alojamiento de archivos de la empresa estadounidense *Google*. Este servicio permite el almacenamiento de multitud de documentos y archivos en la nube, así como la posibilidad de editarlos en tiempo real por varios usuarios al mismo tiempo. Se utilizó para albergar todos los documentos adicionales referentes al proyecto.

Google Forms

Google Forms es una aplicación que permite realizar formularios y encuestas dentro del paquete de aplicaciones de Google. Se ha utilizado para obtener información de la experiencia de uso de en la prueba con usuarios reales.

Trello

Trello es un software de administración de proyectos con interfaz web, cliente para iOS y Android para organizar proyectos. La herramienta permite crear *tableros* y dividirlos según se desee. Se utilizó este software para apuntar y clasificar objetivos según el estado de los mismos, poniendo cada uno de ellos en distintos lugares del *tablero* según su estado.

Slack

Slack es una herramienta de comunicación en equipo para organizar proyectos. Se usó como medio para contactar con el director y co-director del proyecto, así como otros compañeros del grupo de investigación.

Esta herramienta permite tener una gran organización dentro de un equipo, dividendose en canales. Este proyecto se incluyó en el equipo Narratech, creando un canal privado teniendo como participantes del canal al director, co-director y los integrantes de este proyecto. Además de este canal privado, se tuvo acceso a diferentes canales con diferentes temas de interés.

3.4. Especificación de requisitos software

En las próximas subsecciones se especifican los requisitos software necesarios para completar y extender en cierto modo el prototipo de la herramienta *TRPG Maker* que supone el punto de partida de este trabajo.

3.4.1. Elementos necesarios en un videojuego de rol táctico

Los videojuegos de rol tácticos son un subgénero de los videojuegos de rol (RPG), encontrándonos con elementos comunes que los definen.

Estos elementos son necesarios para poder crear un videojuego de rol, siendo requisito la existencia y posibilidad de definirlos en cualquier sistema que pretenda dar la posibilidad de desarrollar videojuegos de rol, sean tácticos o no. Los elementos ya existían en la primera versión del proyecto, y han sido tomados como referencia para esta nueva versión.

Por tanto, se definen los elementos básicos como:

Personaje

Un personaje se puede definir, como el conjunto de todas las secciones anteriormente mencionadas. Haciendo que cada personaje sea único en función de los atributos, las etiquetas, las fórmulas, los huecos, los objetos, las habilidades y la clases especializada que se le haya asignado.

Atributos

Un atributo es un dato numérico que describe una característica de un personaje. Siendo el conjunto de atributos, con diferentes valores, lo que define y diferencia a uno u otro personaje.

Los atributos principales son la vida, la fuerza o el daño, y la velocidad de ataque. Añadiéndose en los videojuegos de rol tácticos atributos básicos como el rango de ataque y el rango de movimiento. Estos atributos son el núcleo de un personaje y suelen encontrarse en todos los personajes de un mismo videojuego para mantener la coherencia en él.

De este modo, pueden definirse dos conjuntos de atributos, los básicos o primarios, que deben encontrarse en todo personaje de un mismo videojuego, y los atributos secundarios, que pueden existir o no en los personajes, siendo un ejemplo claro el atributo "Maná" que suele ser usado por personajes del tipo "Mago", y que no es necesario para personajes que no hagan uso de magia y por tanto, no usen dicho atributo.

Etiquetas

Las etiquetas o *tags* son utilizadas en un videojuego para clasificar elementos y personajes. De este modo, se puede vincular elementos con personajes, elementos con otros elementos y entre clases, en función de esa etiqueta.

Un ejemplo claro es añadir una etiqueta a un arma como "fuego" y añadir a su vez a un personaje esa etiqueta. De esta manera se puede permitir que este personaje porte objetos y armas de tipo fuego, y sabemos que objetos son de tipo fuego por la misma etiqueta.

Fórmulas y huecos

Un personaje en un videjuego de rol también cuenta con "huecos" que pueden contar con modificadores para sus atributos. Estos huecos definen la estructura en la que se divide un personaje, como por ejemplo cabeza, mano izquierda y mano derecha. Los modificadores de atributos son definidos por fórmulas matemáticas que modifican el valor de los atributos

Objetos

Los huecos anteriormente mencionados pueden ser partes del cuerpo, en los que un personaje puede equiparse objetos (ropas, armaduras y armas) que mediante unas expresiones matemáticas (fórmulas) pueden modificar los valores de los atributos del personaje.

Habilidades

Las habilidades son acciones que pueden realizar los personajes, pudiendo tratarse de ataques especiales, curación propia y de aliados. Estas habilidades pueden ser lanzadas a distancia o en rango sobre otro personaje o un grupo de personajes.

Clases especializadas

Las clases especializadas son utilizadas en la mayoría de juegos de rol para diferenciar las capacidades y habilidades de los personajes. Existen muchos tipos, como *bárbaros, magos, guerrero, paladín* y un largo etcétera. Dependiendo de la ambientación del juego estos tipos de clases varían.

En estas clases especializadas se deberán definir las características de un personaje, el tipo de habilidades que puede utilizar y los objetos y armas que puede portar. Una clase especializada puede estar presente en distintos personajes, lo que permite la existencia de varios personajes con la misma clase especializada.

Equipos

Un equipo consiste en un grupo o conjunto de personajes aliados que se enfrentan en batalla juntos contra enemigos.

3.4.2. Defición, relación y almacenamiento de estos elementos

Conocidos los elementos necesarios en un videojuego de rol táctico, existe la necesidad de tener herramientas que permitan definir, relacionar y almacenar de manera persistente estos elementos. Es por ello que existe la necesidad de contar con una base de datos para estos elementos.

Esta base de datos debe de contar con la suficiente flexibilidad para poder almacenar diferentes elementos, distinguiéndose unos de otros.

Por ello se plantea una base de datos estructurada, dividida para cada uno de los elementos definidos en la anterior Sección 3.4.1.

El planteamiento de esta base de datos se caracteriza por definir individualmente cada elemento y relacionarlos todos en personajes y clases especializadas, que será la forma de definirlos. En equipos, se almacenaran referencias a cada uno de los personajes que se quiera añadir a uno u otro equipo.

Base de datos y editor

La base de datos general contara con los elementos definidos en 3.4.1 de forma individual. Para poder añadir, eliminar y modificar esta base de datos será necesario un editor. El editor contará con las siguientes funcionalidades para cada elemento:

1. La tabla y editor de *atributos* será de la forma del cuadro 3.1. El campo *isCore* tiene la función de darle al atributo en cuestión el grado *núcleo*, que lo convierte en un atributo básico, y todo personaje tiene que tenerlo en sus atributos, lo que hace que automáticamente, tras seleccionar la opción, aparezca en todos los personajes.

Cuadro 3.1: Tabla conceptual de atributos

Atributos	
ID	Entero con el identificador del atributo
Nombre	Cadena de texto con el nombre del atributo
Descripción	Cadena de texto con la información del atributo
isCore	Bool para identificar si el atributo es core o no

2. La tabla y editor de *etiquetas* será de la forma del Cuadro 3.2.

Cuadro 3.2: Tabla conceptual de etiquetas

Tags	
Nombre	Cadena de texto con el nombre de la etiqueta

3. La tabla y editor de *tipos de huecos* será de la forma del Cuadro 3.3

Cuadro 3.3: Tabla conceptual de tipos de huecos

Slot Types	
Nombre	Cadena de texto con el nombre del tipo de hueco

4. La tabla y editor de *objetos* será de la forma del Cuadro 3.4. En esta tabla se encuentra una serie de relaciones con las tablas descritas anteriormente. Por un lado se halla el campo *Etiquetas*, que será un conjunto de etiquetas previamente definidos en la tabla etiquetas (Cuadro 3.2). Por otro lado se encuentra el campo *fórmulas*, que consiste en expresiones matemáticas que modifican el valor de un *atributo* que haya sido definido en la tabla atributos (Cuadro 3.1). Por último se puede observar el campo *Combinación de huecos*, que consiste en una serie conexiones entre los elementos definidos en la tabla tipos de huecos, donde se definen las posibles combinaciones de *huecos* necesarios para que un personaje pueda equipar un objeto.

Cuadro 3.4: Tabla conceptual de tipos de objetos

Objetos	
Nombre	Cadena de texto con el nombre del objeto
Descripcion	Cadena de texto con la descripción del objeto
Icono	Imagen del objeto
Etiquetas	Conjunto de cadenas de texto
Formulas	Conjunto de expresiones matemáticas
Combinación de huecos	Conjunto de combinaciones de tipos de hueco

5. La tabla y editor de habilidades será de la forma del Cuadro 3.5. Tipo de habilidad consiste en un enumerado con cuatro posibles elecciones: un objetivo, proyectil, área o área en objetivo. Este campo definirá el modo en el que la habilidad hará efecto al ser lanzada por un personaje. El campo *daño* consiste en un entero, positivo o negativo, con el efecto en la vida del personaje o personajes del que sea objetivo la habilidad. El motivo de positivo o negativo reside en la posibilidad de ser una habilidad de ataque o una habilidad de curación.

Cuadro 3.5: Tabla conceptual de habilidades

Habilidades	
Nombre	Cadena de texto con el nombre de la habilidad
Descripcion	Cadena de texto con la descripción de la habilidad
Tipo de habilidad	Enumerado con el tipo de habilidad
Daño	Entero con el daño que causa la habilidad

6. La tabla y editor de clases especializadas será de la forma del Cuadro 3.6. En esta tabla se reúnen varios de los elementos previamente definidos en las tablas anteriores. El campo *atributos* contendrá un conjunto de atributos definidos en la tabla atributos (Cuadro 3.1). El campo etiquetas contendrá un conjunto de etiquetas definidas en la tabla de etiquetas (Cuadro 3.2). El campo *huecos* consiste en asociar un *hueco* añadido en la tabla tipos de huecos (Cuadro 3.3) a un *objeto* definido en la tabla de objetos (Cuadro 3.4). El campo habilidades radica en añadir un conjunto de *habilidades* añadidas previamente en la tabla habilidades.

Cuadro 3.6: Tabla conceptual de clase especializada

Specialized Class	
Nombre	Cadena de texto con el nombre de la habilidad
Atributos	Conjunto de atributos
Tags	Conjunto de etiquetas
Huecos	Conjunto de huecos con objeto
Habilidades	Conjunto de habilidades

7. La tabla y editor de personajes será de la forma del Cuadro 3.7. Un personaje tiene en su interior atributos, huecos y clases especializadas.

   Los atributos dentro del personaje se dividen en dos grupos, por un lado contiene los atributos principales que todos los personajes deben llevar y que han sido definidos en el Cuadro 3.1) de los atributos, a los que se añaden los deseados por parte del desarrollador para ese personaje en cuestión y los heredados de las *clases especializadas* que se le hayan asignado en la tabla clase especializada (Cuadro 3.6). Por otro lado contiene esos mismos atributos pero con los valores calculados a través de las fórmulas que lleve el personaje asignadas, ya sea a través de los *objetos* que lleve equipados, pasivas o clases especializadas, estos atributos sirven para calcular las acciones dentro del juego.

   Los *huecos* también se ven divididos en dos grupos, los *huecos* añadidos para el personaje que se este tratando por parte del desarrollador y los que vienen heredados de las las *clases especializadas* asignadas y se han definido en el interior de las mismas como se menciona en el apartado anterior.

   Por ultimo el personaje puede contener *clases especializadas* definidas anteriormente, de este modo se pueden asignar tantas *clases especializadas* como el desarrollador vea necesario.

Cuadro 3.7: Tabla conceptual de personajes

Characters	
Nombre	Cadena de texto con el nombre del personaje
Atributos	Conjunto de atributos
Huecos	Conjunto de huecos
Clases especializadas	Conjunto de clases especializadas

3.4.3. Conexión con otras herramientas

Uno de los objetivos principales de este proyecto respecto a la versión anterior de la herramienta, es hacer de TRPG Maker una extensión de Unity independiente, permitiendo la conexión con cualquier herramienta que se desee. Para ello se ha de definir una interfaz que defina las implementaciones necesarias para poder utilizar TRPG Maker con cualquier herramienta. Para ello, también será necesario una serie de atributos previamente definidos en la base de datos que sean básicos para poder obtener los resultados deseados.

Atributos necesarios

Dado que se pretende implementar una interfaz que permita la conexión de la herramienta objeto de este proyecto con otras herramientas o sistemas para crear videojuegos, es necesario definir una serie de *atributos* básicos para el correcto funcionamiento. Estos atributos deberán ser definidos en la base de datos, concretamente en la tabla atributos (Cuadro 3.1). Siendo dichos atributos los siguientes:

1. **Vida**

   Un atributo de vida será necesario para conocer los puntos de vida que tiene cada uno de los personajes.

2. **Daño**

   Este atributo será necesario para calcular la cantidad de *vida* que un personaje puede restar a otro.

3. **Rango de movimiento**

   Este atributo será necesario para calcular el rango de movimiento de cada uno de los personajes.

4. **Rango de salto**

   Este atributo será necesario para calcular cuantas posiciones verticales puede ascender un personaje en su movimiento.

5. **Rango de ataque horizontal**

   Este atributo será necesario para calcular la distancia de ataque en horizontal de los personajes.

6. **Rango de ataque en altura**

   Este atributo será necesario para calcular la distancia de ataque en vertical de los personajes.

Cada uno de estos atributos deberá ser definido como principal, tal y como ha sido explicado en el punto anterior sobre atributos.

Interfaz para conectar a otras herramientas

Una vez conocidos los atributos necesarios, se nombran y describen a continuación las clases que deberán existir en la interfaz para conectar la herramienta TRPG Maker con un gestor de eventos:

1. **Calcular el área de ataque**

   En este método deberá calcularse, dado un personaje y una posición o casilla, el área de ataque de dicho personaje. Para ello es necesario definir que *atributo* es el responsable de definir la distancia de ataque

2. **Mostrar el área**

   Este método deberá mostrar por pantalla el área de ataque y de movimiento que tiene un personaje.

3. **Mover cámara hacia el personaje**

   Este método deberá implementar la transición de un personaje a otro

4. **Mover personaje**

   Este método implementará el movimiento de un personaje. De la posición o casilla inicial, a una posición o casilla destino.

5. **Establecer posición del personaje**

   Este método deberá implementar el cálculo de la posición de un personaje.

6. **Mostrar animación**

   Este método será usado para llamar a la animación del personaje

7. **Obtener personaje situado en una casilla**

   Este método deberá ser implementado para devolver al personaje que se encuentra en una posición dada.

8. **Obtener la casilla donde se sitúa un personaje**

   Este método deberá ser implementado para devolver la casilla o posición en la que se encuentre un personaje dado.

9. **Métodos para la inteligencia artificial**

   En este método o conjunto de métodos deberán ser implementados para otorgar a los personajes no jugables una inteligencia artificial.

3.4.4. Gestión del tiempo y combate

La gestión de tiempo utilizada en los videojuegos de rol tácticos es denominada turno, habiendo sido explicado la página 8 del Capítulo 2

El turno podrá ser dado a cada personaje en el campo de batalla de forma aleatoria o según un atributo. De ser según un atributo, este deberá ser definido previamente en la tabla de atributos (Cuadro 3.1) y marcado como principal, así se evaluará dicho atributo para cada personaje, para determinar el orden de turnos en el combate.

Por otro lado y muy ligado al turno se encuentra el combate, que será necesario implementar para poder realizar un videojuego mínimamente interesante con la herramienta. En el combate, cada personaje en el terreno de juego podrá realizar ciertas acciones en su *turno*, siendo dichas acciones las siguientes:

1. **Atacar**

   Al seleccionar esta acción se deberá *dibujar* sobre el terreno de juego las casillas en las que el personaje que tenga el *turno* puede realizar su ataque básico (daño que realiza un personaje dado por el atributo *daño*).

2. **Habilidades**

   Al seleccionar esta acción se deberá mostrar todas las *habilidades* que tiene disponibles para usar el personaje que tenga el *turno*. Al seleccionar una de las habilidades existentes se permitirán varias formas de utilizarla, permitiendo que se pueda seleccionar sólo a un objetivo, realizar una habilidad en área desde el personaje que se este manejando o realizarlo en área desde el objetivo que se pueda seleccionar.

3. **Objetos**

   Al seleccionar esta acción se deberá mostrar todos los *objetos* que dispone el *equipo* del personaje que disponga el *turno*. Al seleccionar alguno de estos objetos aparecerán distintas opciones en función del objeto

4. **Pasar turno**

   Al seleccionar esta acción el turno del personaje actual pasará al siguiente.

El combate es el resultado de una serie finita de turnos, que finaliza según el desarrollador defina, pudiendo y teniendo identificados los siguientes motivos de fin de combate:

- Cuando no quedan enemigos oponentes, y el resultado será victoria para el jugador.
- Cuando no quedan personajes jugables, y el resultado será derrota para el jugador.
- Cuando se definen ciertos criterios para acabar el combate, como un número determinado de turnos, o una cantidad de tiempo que debe trascurrir hasta acabar el combate.

3.4.5. Demostración de la herramienta

Una vez realizadas todos las especificaciones tratadas en este capitulo, se realizará una demostración de la herramienta creando un videojuego.

Este videojuego consistirá en un combate entre dos *equipos*, uno controlado por un jugador y otro controlado por la inteligencia artificial que se desarrollará en los métodos descritos en la subsección 3.4.3 de este capítulo.

Esta demostración permitirá al jugador realizar las acciones definidas en la subsección anterior. La demostración de la herramienta, al contener el groso de su funcionalidad en el editor, se ejecutará en el propio editor de Unity para poder realizar las modificaciones deseadas desde la base de datos, así como crear y eliminar todo elemento que se desee.

Para el uso correcto de la herramienta y su demostración, se facilitará un manual de uso en donde se explicará paso a paso como usar la herramienta.

Capítulo 4

Análisis, diseño e implementación

En este capítulo se abordará el análisis, el diseño y la implementación del sistema especificado en el capítulo anterior. Se establecerá las formas en las que el usuario debe interactuar con el sistema, así como los análisis y el diseño realizado para implementar las especificaciones.

Los apartados que se expondrán en este capítulo estarán divididos precisamente en una sección de análisis y diseño, y otra más orientada a la implementación. Dentro de esas secciones se cubrirán los siguientes apartados:

- Base de datos
- Editor de la base de datos
- Gestión del juego
- Conexión con otras herramientas

4.1. Análisis y diseño

En esta sección trataremos sobre el análisis y diseño de las especificaciones extraídas en el capitulo anterior. Se tratara de refinar los requisitos, modelar y dar forma al sistema.

4.1.1. Vista general del sistema

Según las especificaciones, el sistema puede diferenciarse en tres grandes núcleos.

La **base de datos**, encargada de almacenar toda la información necesaria sobre el videojuego que se desee crear, tal y como se precisa en la sección 3.4.2 del capitulo anterior. Esta base de datos necesita de un **editor** para poder añadir, eliminar y modificar la información, siendo su interfaz gráfica de usuario el elemento básico que servirá para dar control al autor del juego que se estará creando.

Hacer que TRPG Maker sea completamente independiente, pero permitiendo la conexión con otras herramientas da lugar a la necesidad de diseñar una colección de métodos abstractos y propiedades (interfaz) para tener todo lo necesario para conectar la herramienta bien definido.

Por último, el sistema gestor del juego será el encargado del turno y del sistema de combate, siendo requerido para tratar los eventos del juego, así como el intermediario entre la base de datos y los objetos[1] de Unity.

4.1.2. Base de datos

Tal y como se ha visto en el capitulo anterior, la base de datos deberá almacenar la información necesaria para crear un videojuego de rol táctico.

Las relaciones necesarias en la base de datos pueden verse en la Figura 4.1. La base de datos se caracterizará por tener los elementos simples: atributos, etiquetas, tipos de huecos y equipos; que son usados por los elementos complejos: personajes, clases especializadas y objetos.

Esta base de datos debe poder ser totalmente editable por el usuario de la herramienta, pudiendo poblar, modificar y eliminar la información de cada una de las entidades. Siendo el proceso de *editar* clave para el correcto funcionamiento. Se analizará en la próxima subsección el editor de la base de datos para su correcto diseño e implementación.

[1]Objetos de Unity - `https://docs.unity3d.com/es/current/Manual/GameObjects.html`

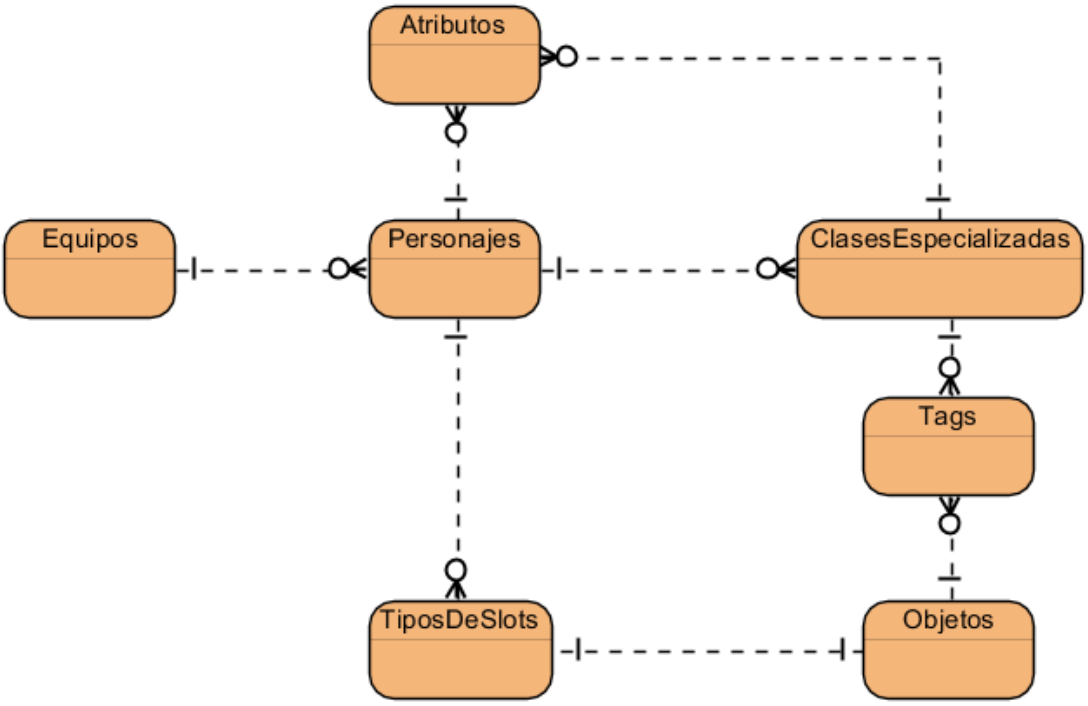


Figura 4.1: Diagrama Entidad-Relación de la base de datos

4.1.3. Editor de la base de datos

El editor de la base de datos es una de las partes más importantes de este proyecto, ya que es la forma con la que el usuario interactuará con el sistema.

Al tratarse de una base de datos con diferentes relaciones, es imprescindible que al modificar o eliminar algún componente que haya sido previamente relacionado, no corrompa la base de datos dejando información incoherente.

Como se mencionó con anterioridad, el editor de la base de datos es la puerta de entrada del usuario con la herramienta, debiendo tener un correcto funcionamiento, ser accesible, intuitiva, estar bien diseñada para ser comprensible para el usuario. En las siguientes subsecciones se definirá el analizará y se explicará el diseño del editor de la base de datos.

Diseño del editor de la Base de Datos

El editor de la Base de Datos deberá permitir dar acceso a los diferentes *elementos* que pueden definirse en la base de datos. Añadir, eliminar y modificarlos, para ello se precisa de una ventana, que contenga diferentes botones para poder acceder al *elemento* que se desee añadir, modificar o eliminar, siendo el resultado deseado similar a la maqueta[2] que puede verse en la Figura 4.2.

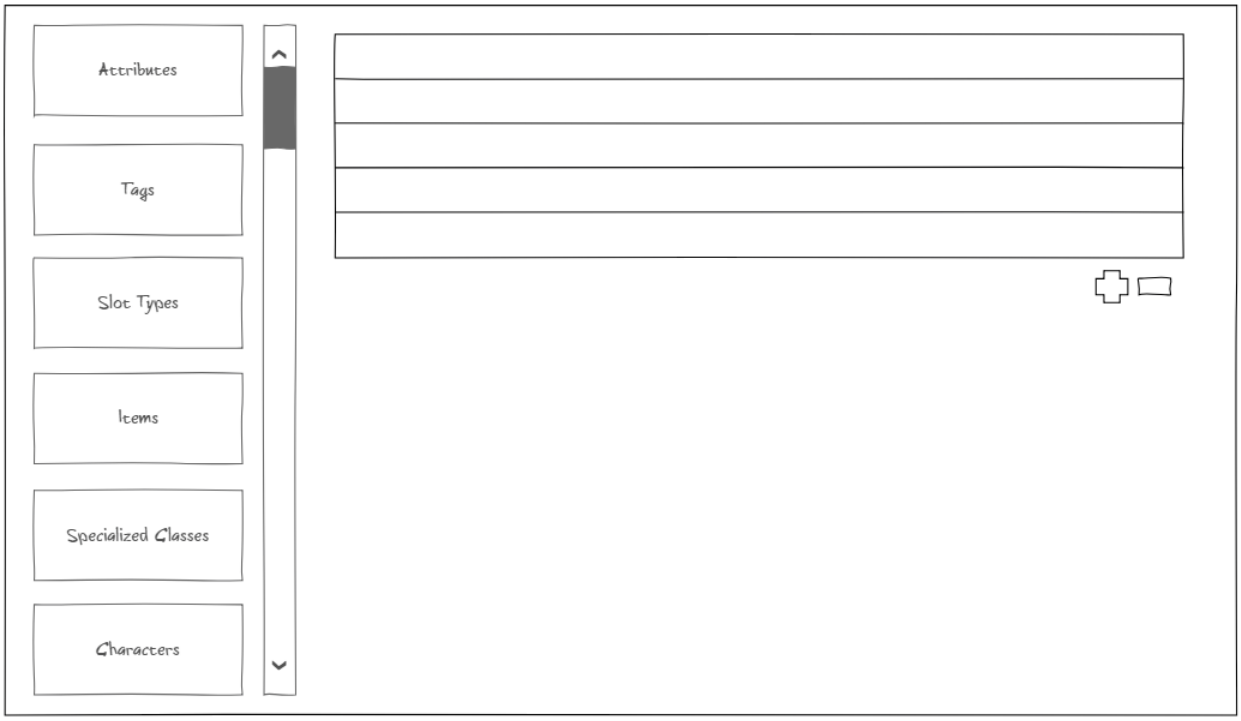


Figura 4.2: Maqueta de la ventana de atributos

Tal y como aparece en la Figura 4.2, se desea disponer de una sección lateral izquierda donde se divide todos los elementos que forman la base de datos, bien diferenciados.

Al pulsar sobre cada uno de los diferentes elementos del panel lateral izquierda de la ventana deberá mostrar en el resto de la ventana una lista con todos los datos que existan en esa elemento, siendo el ejemplo de la Figura 4.2 una lista con todos los *atributos* existentes en la Base de Datos.

[2]Una maqueta (mockup o mock-up) es un modelo simplificado de la interfaz de una aplicación, utilizada para para la demostración, evaluación del diseño, promoción, y otros fines

Para añadir y eliminar datos a la base de datos, en la parte derecha, donde se encuentra la lista con todos los datos del elemento, deberá existir un botón que permita eliminar y otro que permita crear, debiendo crear un nuevo dato en la base de datos al pulsa sobre *crear*. Teniendo seleccionado un dato existente y pulsando sobre el botón *eliminar*, el dato seleccionado deberá ser eliminado de la base de datos.

El diseño de la ventana de los elementos *atributos*, *etiquetas* y *tipos de huecos* seguirán un patrón similar, en forma de lista con el conjunto de los datos que existan. Los *atributos* como única salvedad respecto a *etiquetas* y *tipos de huecos*, será que los datos necesarios para definir un atributo serán mayores, y por tanto deberá contar con una forma para mostrar esta información, para ello, al pulsar sobre cada *atributo*, éste se desplegará mostrando toda su información, pudiendo modificarse desde esa misma ventana.

El diseño de la ventana del elemento *objeto* cambiará respecto al diseño de los elementos descritos anteriormente.

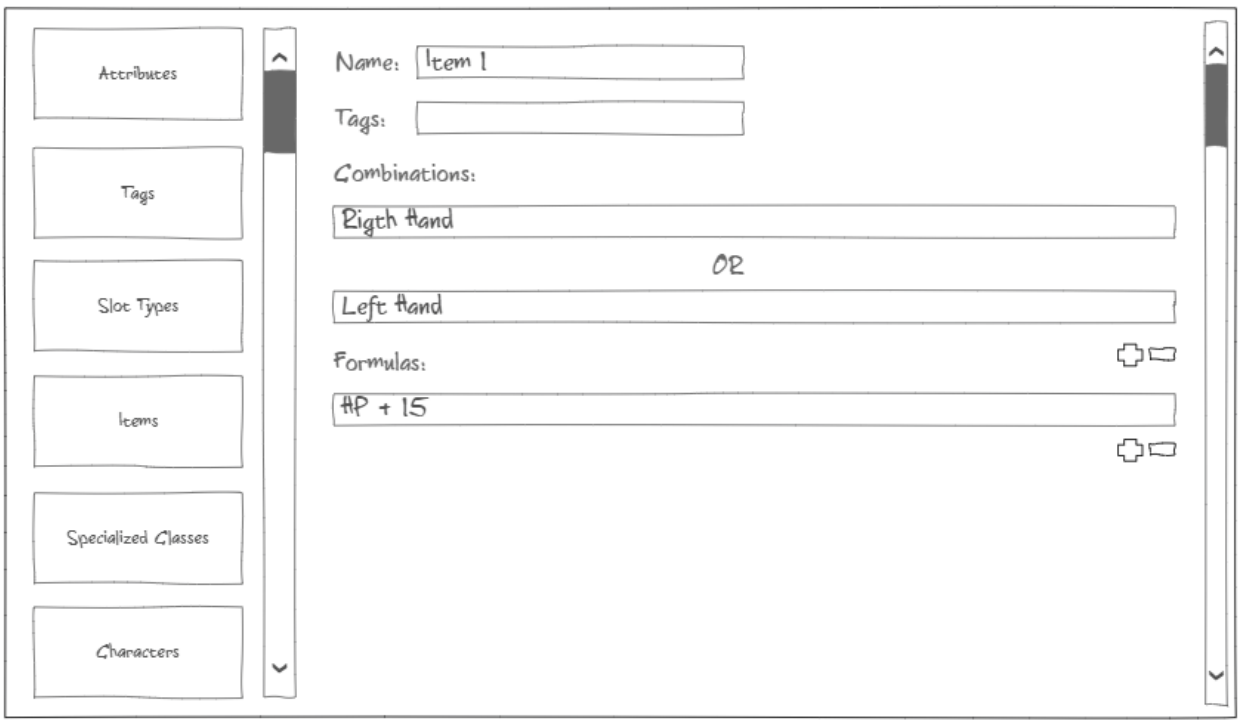


Figura 4.3: Mockup de la ventana de objetos

Tal y como puede observarse en la Figura 4.3, el diseño de la ventana del editor de objetos apareceré la información de un dato en la parte derecha de la ventana. Para acceder a esta información previamente deberá aparecer en la ventana una lista con todos los *objetos* existentes en la base de datos, y al pulsar sobre uno de ellos se mostrará toda la información acerca del objeto seleccionado, permitiendo editar, modificar y eliminar la información del dato. Para eliminar el propio dato, se deberá hacer desde la anterior ventana, la que muestra la lista con todos los objetos.

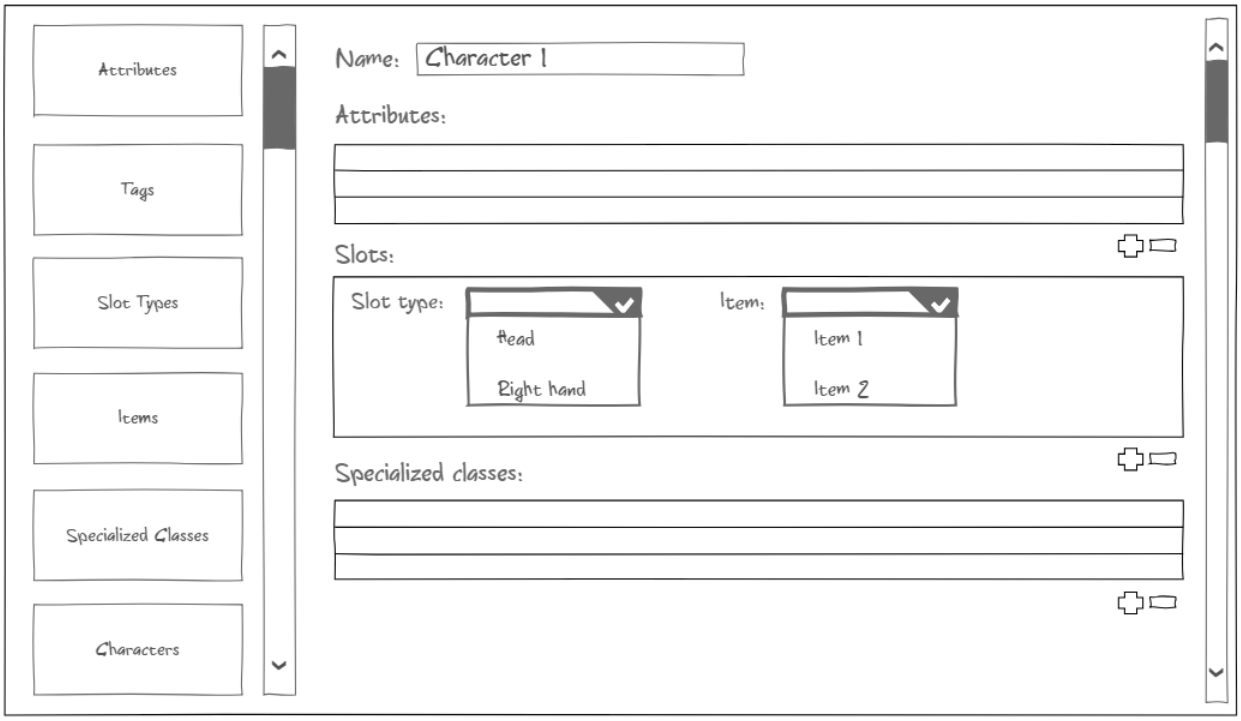


Figura 4.4: Mockup de la ventana de personajes

En la Figura 4.3 puede leerse *combinations*, aquí se especificaran todas las combinaciones posibles de *tipos de hueco* para poder *equipar* el objeto que se este editando.

El diseño de la ventana de *personajes* y *clases especializadas* seguirá un patrón similar. El acceso a cada dato será de la forma descrita anteriormente en *objetos*, dada una lista de datos, al seleccionar un dato, se mostrará toda la información sobre el dato. Al igual que en *objetos*, en esta lista se permitirá la creación de nuevos datos y la eliminación de los datos existentes. Para modificar la información de los datos, en el caso de personajes, tendrá un diseño similar al que aparece en la Figura 4.4.

En la ventana de personajes se deberá permitir añadir *atributos* previamente definidos en la base de datos, seleccionar un conjunto de *tipos de huecos* y seleccionar un conjunto de *clases especializadas*. La ventana de *clase especializada* en el editor de la base de datos será similar, con la excepción de que no se podrá seleccionar una *clase especializada*.

Por último, la ventana de *equipos* deberá tener una primera vista con la lista de todos los equipos existentes, pudiendo crear y eliminar los equipos desde esta lista. Al pulsar sobre un equipo deberá aparecer la información del mismo, con una lista de *personajes*, dando la posibilidad de poder añadir y eliminarlos de la lista desde esa ventana.

Acceso y uso del editor de la base de datos

Con el diseño del editor de la base de datos ya establecido y para poder conocer las formas en las que se debe utilizar la herramienta, se procede a definir como interactuar con la herramienta.

Para acceder al editor, se deberá crear una pestaña en Unity que albergue todas las funcionalidades de la herramienta, debiendo encontrarse *Database editor*, que al seleccionarlo deberá mostrar una ventana nueva con el editor de la base de datos.

En esta ventana, deberán mostrarse todos los elementos de la base de datos en una columna en la parte izquierda de la ventana:

1. **Atributos**. Se permitirá añadir nuevos atributos y eliminar los existentes al pulsar sobre un botón. Al añadirse, aparecerá con un nombre e identificador por defecto y todos los campos vacíos. Se podrá editar el nombre e identificador, añadir una descripción y determinar si el atributo es primario.

2. **Etiquetas**. Se permitirá añadir nuevas etiquetas y eliminar los existentes al pulsar sobre un botón. Al añadirse, aparecerá con un nombre por defecto que podrá editarse.

3. **Tipos de huecos**. Se permitirá añadir nuevos tipos de huecos y eliminar los existentes al pulsar sobre un botón. Al añadirse, aparecerá con un nombre por defecto que podrá editarse.

4. **Objetos**. Se permitirá añadir nuevos objetos y eliminar los existentes al pulsar sobre un botón. Al añadirse, aparecerá con un nombre por defecto. Se podrá editar el nombre, añadir una descripción, añadir un icono representativo, asignar etiquetas existentes en la base de datos, definir fórmulas que modifiquen atributos al usarse y determinar la combinación o combinaciones de huecos necesarios para utilizar el objeto.

5. **Habilidades**. Se permitirá añadir nuevas habilidades y eliminar los existentes al pulsar sobre un botón. Al añadirse, aparecerá con un nombre por defecto. Se podrá editar el nombre, añadir una descripción, seleccionar el tipo de área y objetivo que tendrá y definir fórmulas que modifiquen fórmulas que modifiquen atributos al usarse.

6. **Clases especializadas**. Se permitirá añadir nuevas clases especializadas y eliminar los existentes al pulsar sobre un botón. Al añadirse, aparecerá con un nombre por defecto. Se podrá editar el nombre, añadir atributos existentes en la base de datos, asignar etiquetas existentes, asignar el tipo de huecos que tiene la clase y asignar habilidades.

7. **Personajes**. Se permitirá añadir nuevos personajes y eliminar los existentes al pulsar sobre un botón. Al añadirse, aparecerá con un nombre por defecto. Se podrá editar el nombre, añadir atributos existentes en la base de datos, asignar el tipo de huecos que tiene la clase y asignar la clase o clases.

8. **Equipos**. Se permitirá añadir nuevos equipos y eliminar los existentes al pulsar un botón. Al añadirse, aparecerá con un nombre e identificador por defecto. Se podrá editar el nombre e identificador, seleccionar si será un equipo controlado por el jugador y añadiendo los personajes existentes al equipo.

4.1.4. Gestor del juego

Para gestionar el modo de juego en TRPG Maker se necesita definir las características básicas y necesarias para todo videojuego de rol táctico, pero dejando la posibilidad de usar esta herramienta para otros géneros. Para ello se necesita una interfaz gráfica que de la posibilidad de seleccionar el tipo de juego que se esta desarrollando, actualmente sólo del tipo rol táctico.

Las características básicas para gestionar y dotar de coherencia la juego deberán aparecer de forma visual en el interfaz gráfico de la herramienta, debiendo seleccionar el atributo existente en la base de datos y opciones siguientes:

- Tipo de turno, permitiendo elegir si el turno será de modo aleatorio, o dado por un atributo

- Atributo para turno, seleccionando el atributo que definirá el turno en caso de haber sido seleccionado el tipo de turno de esta manera.

- Atributo de salud, debiendo seleccionar el atributo que representa la saludo o vida de los personajes en el juego.

- Atributo de daño, debiendo seleccionar el atributo que representa el daño de los personajes en el juego.

- Atributo de rango de movimiento, debiendo seleccionar el atributo que representa la distancia de movimiento que el personaje del juego puede realizar.
- Atributo de rango de salto, debiendo seleccionar el atributo que representa la distancia de salto que el personaje del juego puede realizar.
- Atributo de rango de ataque, debiendo seleccionar el atributo que representa la distancia de ataque que el personaje del juego puede realizar.
- Atributo de ataque en vertical, debiendo seleccionar el atributo que representa la distancia de ataque que el personaje del juego puede realizar en vertical.

En el combate del juego, dentro de cada turno, si el personaje puede ser utilizado por el jugador se mostrará un menú en el que se podrán realizar cuatro acciones: moverse, atacar, lanzar un habilidad o finalizar el turno. En función de la opción elegida se realizaran unas acciones u otras y se modificarán los valores de la base de datos si fuera necesario.

Si el equipo no es controlado por el jugador, se encargará la inteligencia artificial de realizar las acciones oportunas.

4.1.5. Conexión con otras herramientas

La conexión con otras herramientas tendrá una interfaz gráfica donde se podrá seleccionar el conector que se quiera utilizar para cada *framework* encargado del apartado visual. Para ello existirá una pestaña en el menú de *Unity* que muestre una ventana para configurar esta información.

En esta ventana, además de elegir el conector deseado, se podrá configurar los distintos parámetros necesarios para este *framework* en concreto. Estas opciones permanecerán separadas del editor principal y serán configuradas de manera separada para mantener la integridad entre la propia herramienta y otros *frameworks*.

Para realizar la conexión con cada *framework* se creará una interfaz que deberá ser implmentada para dar respuesta a cada una de estas acciones:

- Mostrar el área de acción deseada en función de que sea un ataque, un movimiento o una habilidad. Además devolverá la casilla seleccionada por el jugador.
- Realizar un movimiento de un personaje a una posición del escenario.
- Mover la posición de la cámara hacia un personaje en concreto.
- Mostrar una animación sobre un personaje.
- Mostrar una animación concreta de un personaje.
- Colocar un personaje en una posición concreta del escenario.
- Devolver el personaje situado en una posición dada.
- Devolver la posición de un personaje dado.
- Dado un área devolver los personajes situados en la misma.

Se implementará únicamente un conector que realice todas estas acciones con el *framework Isounity*, debiendo seleccionar en la interfaz gráfica de la herramienta las texturas necesarias para representar las celdas donde el personaje puede moverse, las celdas donde el personaje puede atacar, las celdas donde el personaje puede realizar las habilidades y la flecha que indique la posición donde se esta seleccionando con el ratón.

4.2. Implementación

En esta sección se abarcará el cómo se ha implementado la herramienta, según los principios de la Ingeniería del Software [31] ,basándose en diagramas UML de clases y secuencia, así como las tecnologías utilizadas para tal fin.

El proyecto ha sido desarrollado completamente en C#[3], como extensión del entorno de desarrollo Unity.

[3]C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET,

4.2.1. Tecnologías

En este apartado se detallan las tecnologías utilizadas para implementar el proyecto según el análisis realizado previamente.

Unity

Herramienta que ya ha sido comentada en profundidad en el Capítulo 2. TRPG Maker es una extensión de esta tecnología por lo que es un pilar fundamental en la implementación.

Microsoft Visual Studio

Entorno de desarrollo Integrado desarrollado por Microsoft. Soporta varios lenguajes entre los cuales C++, .NET, Java y C# entre otros.

Esta herramienta es la que se ha usado para la programación en C# de los diferentes scripts necesarios para la creación del sistema.

Visual Paradigm

Herramienta de modelado UML que da soporte a múltiples lenguajes de programación y herramientas de desarrollo. Integrada en Microsoft Visual Studio ha sido utilizada para la elaboración de los diagramas dentro del proyecto, permitiendo tanto generar código a través de los mismos como generar diagramas con el código ya existente.

4.2.2. Prototipos

A continuación se detalla la implementación de los prototipos iniciales creados para entender cada una de las herramientas que se iban a utilizar y hacer pequeñas pruebas de concepto sobre las especificaciones iniciales para comprobar la posible implementación de las mismas.

Iso Test

Para entender la herramienta *IsoUnity* [14] y ver todas las posibilidades que nos podía aportar esta herramienta se implementó una pequeña *demo*, o prototipo, llamada *IsoTest*[4] utilizando todas las opciones que nos ofrecía como el dibujado del mapa y sus celdas, la gestión del *input* del usuario, el movimiento de los personajes, los diálogos, el teletransporte entre distintas posiciones del mapa y las animaciones.

Esta *demo* o prototipo (Figura 4.5) consistió en la creación de un escenario, la inserción de tres personajes y las interacciones entre los mismos y con el escenario. Al ser un prototipo no se tuvieron en cuenta la calidad gráfica de las *IsoTextures* o los *IsoDecorations* y se centró principalmente en la funcionalidad del mismo.

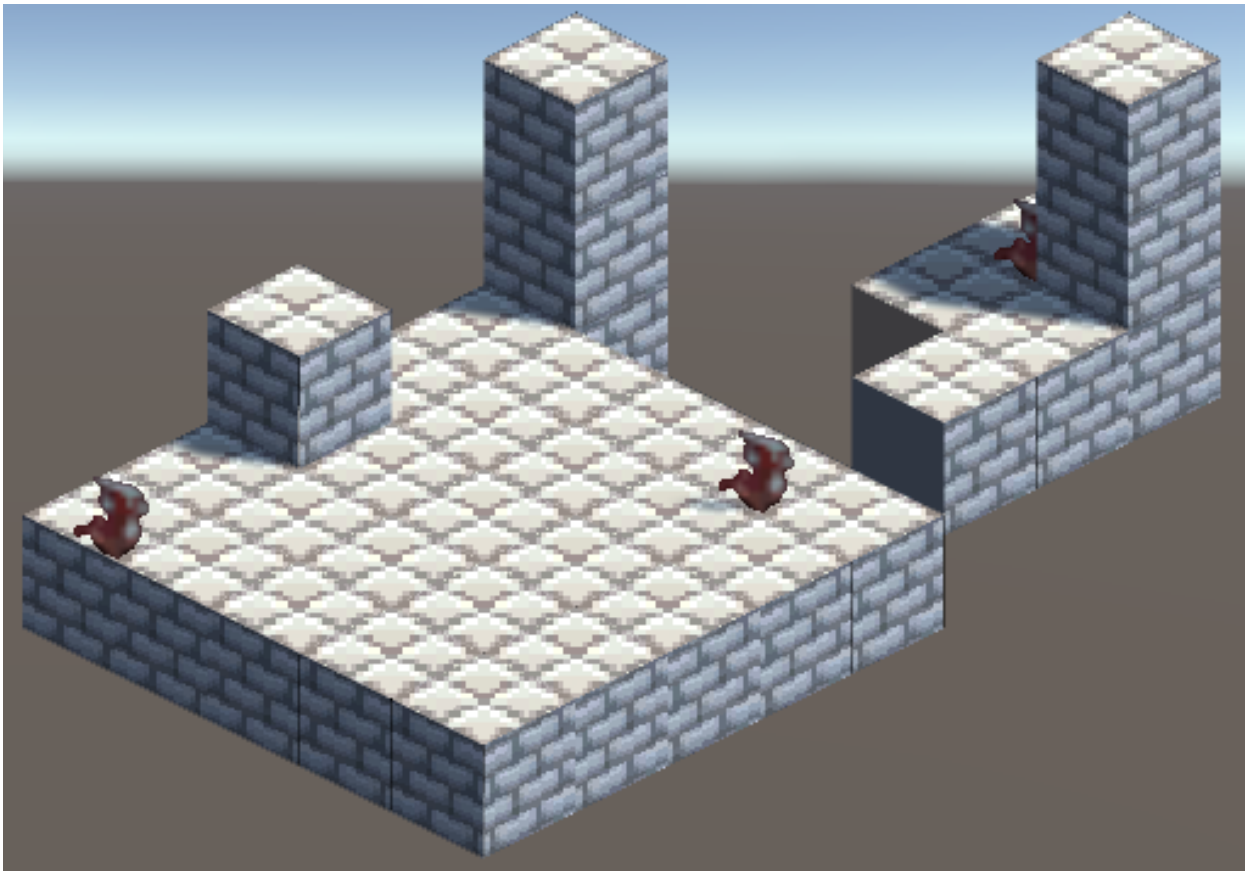


Figura 4.5: Ejemplo del prototipo de *IsoTest* para comprender *IsoUnity*

[4]IsoTest es una *demo*, o prototipo, desarrollado para ver todas las posibilidades que *IsoUnity* puede ofrecer - `https://github.com/WyrnCael/IsoTest`

TRPG Test

Para comprender el funcionamiento del prototipo creado por Druet y Gonzalez para la herramienta *TRPG Maker*, se implementó una demo llamada *TRPG Test*[5], creando un escenario de batalla con *IsoUnity* y definiendo personajes, con sus atributos y todas las necesidades para hacer funcionar la herramienta.

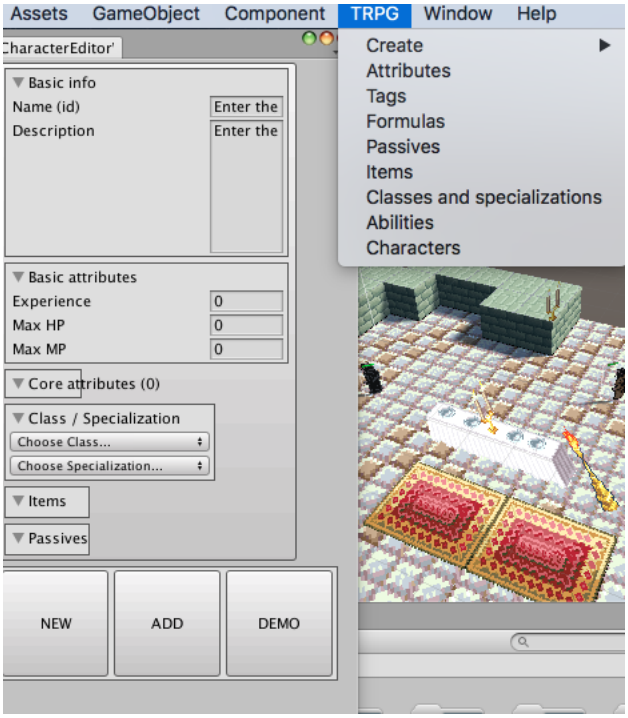


Figura 4.6: Opciones de *TRPG Maker* original y su editor de personajes

En este momento se comprobó que la herramienta era de difícil uso, sin apenas documentación. Se necesitaba realizar los ajustes de una forma muy concreta y con excesiva dependencia de un llamamiento exacto de los elementos, al mismo tiempo, se observó que la herramienta estaba excesivamente vinculada a IsoUnity.

Aun con estos impedimentos se consiguió probar y realizar una demostración de la herramienta.

[5]TRPGTest: `https://github.com/mperez01/TRPGTest`

Se crearon tres personajes, añadiendo atributos, objetos y habilidades. Las diferentes opciones aparecen en la barra de herramientas de Unity, con el nombre *TRPG*, tal y como aparece en la Figura 4.6, teniendo dificultad para comprender como utilizar cada uno de los apartados.

Una vez configurado todo, se ejecuto la prueba, pudiendo sólo atacar a los oponentes en bucle, sin opciones para agregar ninguna inteligencia artificial a los enemigos o finalizar el combate.

Prueba de almacenamiento de datos

Para una primera toma de contacto con la herramienta de desarrollo Unity [7] y el correcto desarrollo del resto de la herramienta se debía comprobar la consistencia de los datos una vez almacenados, como relacionarlos entre ellos y su modificación una vez se compilaba y ejecutaba la herramienta. Por ello se creó un primer prototipo para el inventario y la gestión de los objetos que posee un personaje.

En este prototipo los datos eran almacenados en objetos de tipo *.asset*, una extensión propia de Unity que permite guardar la configuración de elementos dentro de la herramienta siempre que sean *Serializable*. Estos datos eran asignados directamente mediante código provisional en una clase temporal y añadidos a este tipo de objetos.

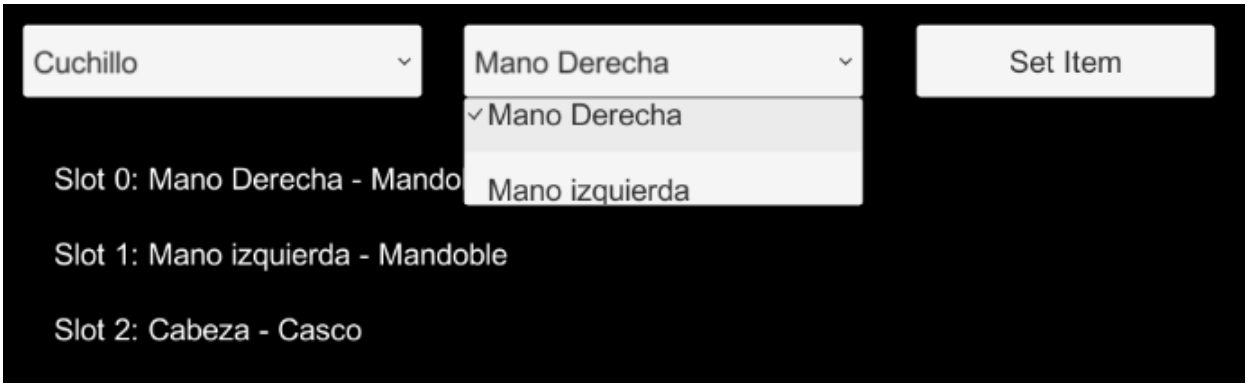


Figura 4.7: Ejemplo del prototipo de la herramienta

Para esta primera toma de contacto se creó un *Inventario* que simplemente contenía una lista de objetos, un *Personaje*, una lista de tipos de huecos donde podrían ser almacenados los objetos y una serie de objetos. Para poder asignar un objeto a un *Personaje* se comprobaba que el objeto pudiese ser equipado según los tipos de huecos que tenía dicho personaje y los requeridos por el objeto en cuestión. Si esta asignación era posible se asignaba el ítem en el hueco o los huecos correspondientes.

Las pruebas realizadas con este prototipo fueron exitosas para la consistencia de los datos y el manejo de los mismos por lo que sirvió como base para empezar con la creación de la base de datos y las asignaciones a la misma.

4.2.3. Base de datos

Como se quería hacer una herramienta totalmente independiente y que estuviese contenida en la herramienta *Unity* sin necesidad de sistemas externos para gestionar la base de datos se decidió utilizar objetos programados en *C#* y almacenarlos en el propio proyecto.

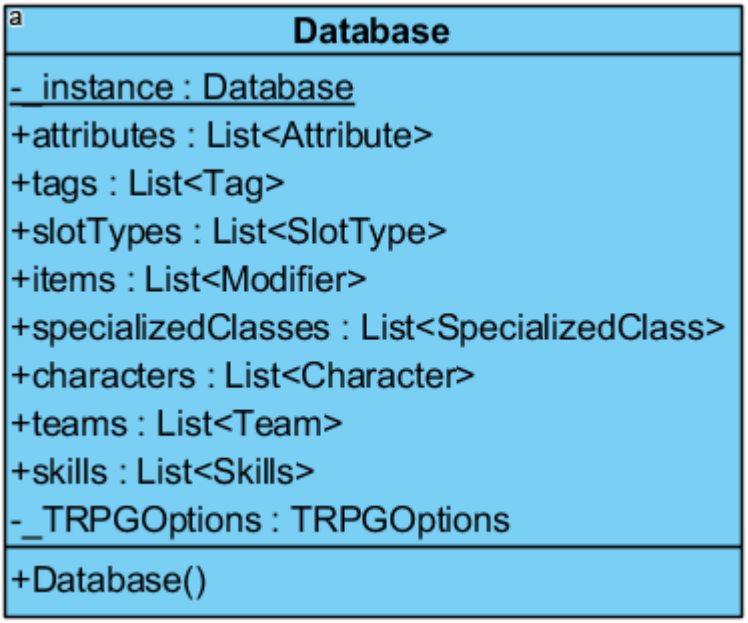


Figura 4.8: Objeto que contiene la información de la base de datos

Para la consistencia de los datos se ha creado un objeto principal llamado ***Database*** en el que se incluye toda la información introducida y es almacenada con la extensión *.asset* proporcionada por *Unity*. Esta extensión permite serializar los datos y almacenarlos para su posterior uso tanto en edición como en ejecución.

Este objeto *Database* (véase Figura 4.8) contiene una serie de atributos que corresponden al resto de objetos de la base de datos. Estos atributos se crearon del tipo *List<>* para que funcionasen como tablas que contuviesen los propios atributos de estos objetos actuando como columnas de estas tablas. Así cada elemento de la lista actúa como una fila de estas tablas.

Cabe destacar que este objeto *Database* sigue el patrón de diseño *Singleton* al ser un objeto único del que deseamos evitar que exista mas de una instancia del mismo.

Los objetos contenidos en esta *Database* guardan relaciones entre si y unos son contenidos dentro de otros (véase Figura 4.9) permitiendo así tener una base de datos consistente y obtener un modelo relacional.

Los objetos contenidos en la base de datos fueron explicados en el apartado 3.4.2 y se implementaron con los atributos indicados.

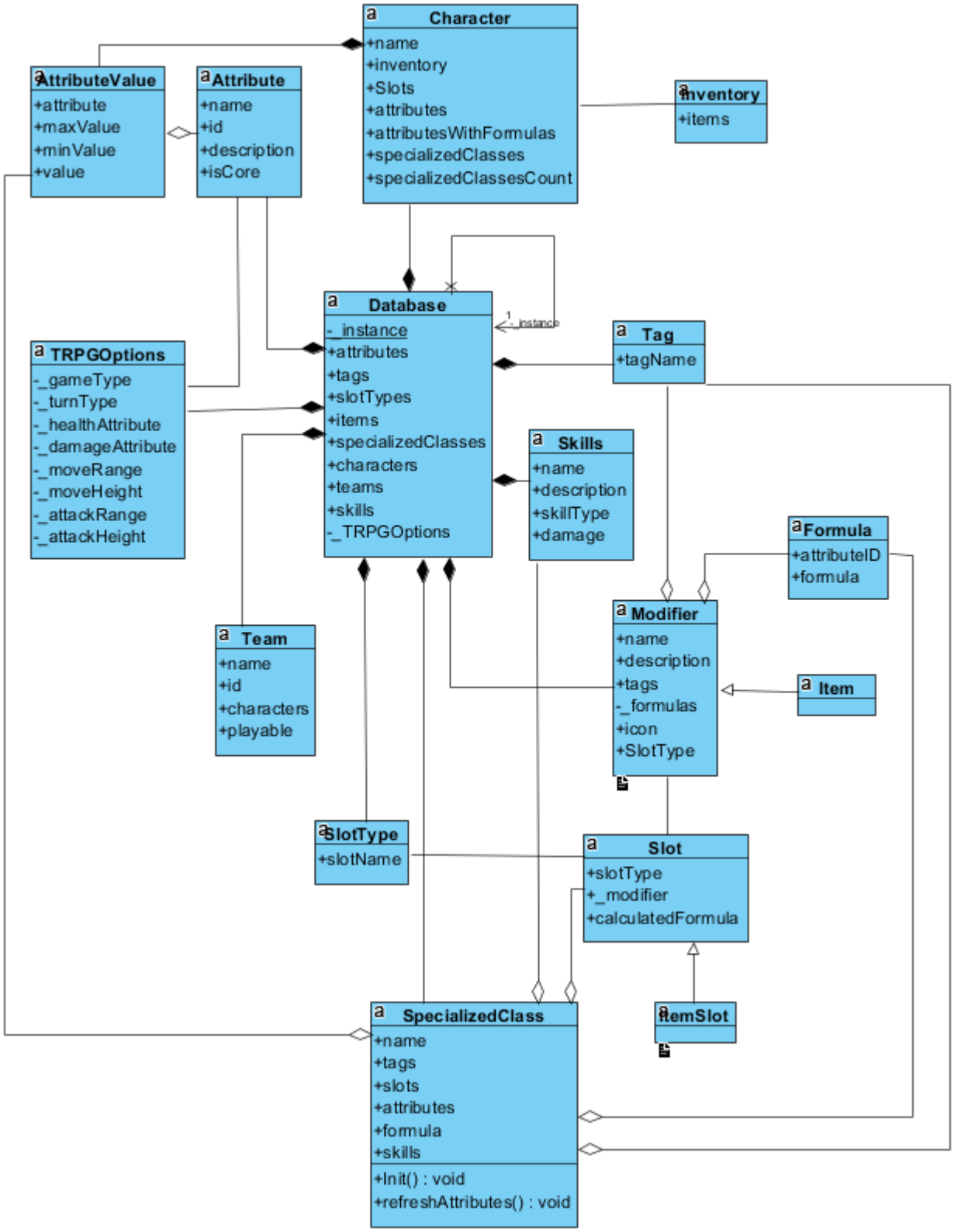


Figura 4.9: Diagrama de clases de los objetos que forman parte de la base de datos

4.2.4. Gestor de la base de datos

El editor de la base de datos se creó utilizando el lenguaje de programación *C#* y la *API* proporcionada por *Unity* para utilizar su editor interno.

Inicialmente creamos un un editor para cada objeto de la base de datos, permitiendo crear los mismos manualmente y modificarlos desde el *Inspector* de la herramienta (véase Figura 4.10). Esto era incompatible con la filosofía que estábamos planteando para la herramienta, la cual debía ser intuitiva y se debían manejar y almacenar todos los datos de forma conjunta, por lo que se decidió crear una ventana principal que manejase toda la información y no permitir la creación de objetos de manera independiente.

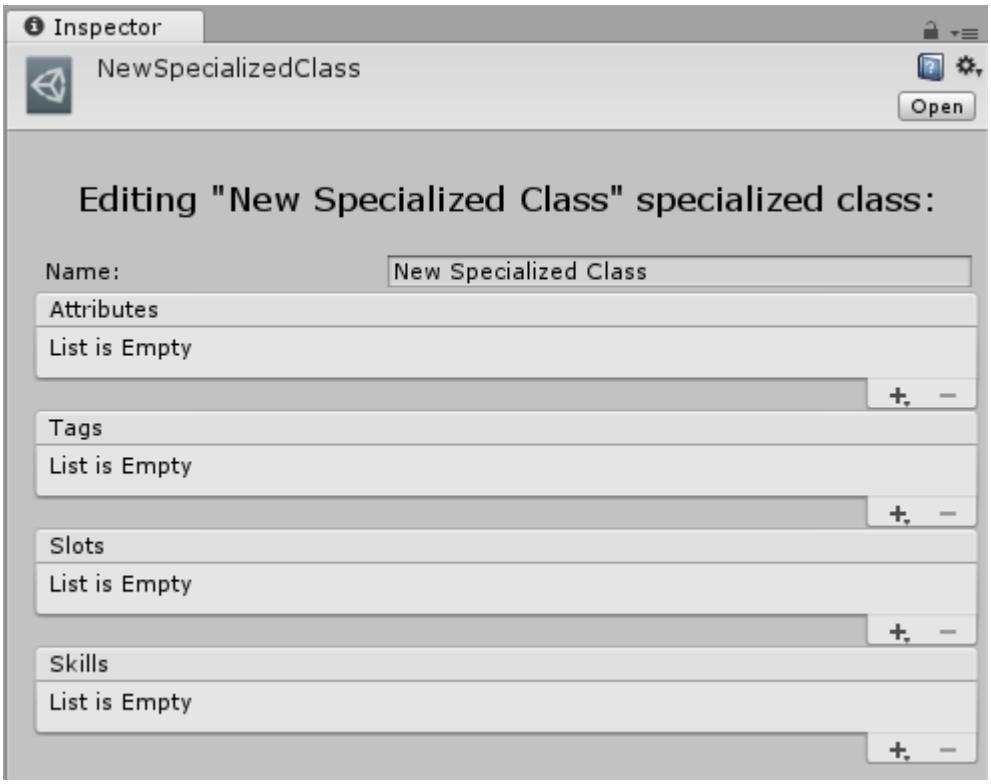


Figura 4.10: Ejemplo de la edición de un objeto desde el Inspector

Se creó entonces una ventana principal (véase Figura 4.11) heredando de la clase *EditorWindow*, proporcionada por *Unity*, y se dividió en dos paneles: un panel izquierdo (o menú) donde se muestran todos los botones con las distintas opciones de la base de datos y un panel derecho donde dinámicamente se muestra la información correspondiente a la tarea que se está realizando actualmente.

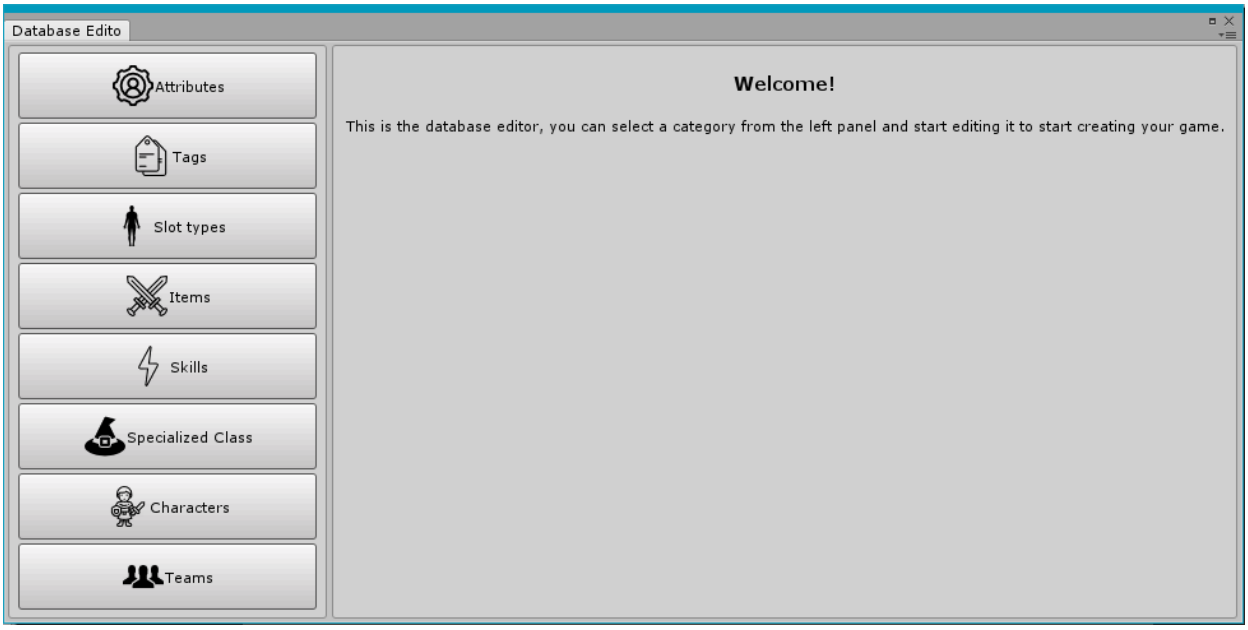


Figura 4.11: Ventana principal del editor de la base de datos

Para el panel izquierdo contábamos con elementos que contenían múltiple información, los cuales se quería que fuesen fácilmente accesible para su edición, por lo que se decidió crear una lista que permitiese añadir, eliminar o reordenar ciertos elementos de una forma rápida y sencilla, para ello se decidió utilizar una *Reorderablelist*[6] cuando estos botones fuesen seleccionados (véase Figura 4.12).

[6]Herramienta no documenta incluida en el código interno de `Unity`, encontrada y documentada por Valentin Simonov - `http://va.lent.in/unity-make-your-lists-functional-with-reorderablelist/`

Figura 4.12: Botón de objeto seleccionado muestra un *Reorderablelist*

Para manejar la información de los objetos se creó un panel especifico para cada objeto que se sitúa en la parte derecha de la ventana general, se creó una clase abstracta (*LayoutWindow*) que implementa una serie de métodos que permiten la creación de un panel y es heredada por cada panel de los objetos de la base de datos. Los objetos que requieren manejar mucha información crean un panel intermedio en el que se muestra una lista de todos los objetos posibles con opción de editarlos o eliminarlos, mientras aquellos que contienen información simple directamente muestran un editor.

Para mostrar la información concreta de cada objeto se crearon clases que heredan de *CustomEditor*, una clase proporcionada por *Unity* para la creación de editores personalizados, que deben ser incluidos en la carpeta *Editor* para ser reconocidos.

Para la edición de información de ***Attributes***, ***Tags*** y ***SlotTypes***, al ser objetos que no requieren de un editor muy desarrollado por no contener gran cantidad de atributos, se decidió mostrar un editor que únicamente incluye un *Reorderablelist* con la lista completa de objetos contenidos en ellos, pudiendo ser añadidos nuevos objetos, editados o eliminados (véase Figura 4.13).

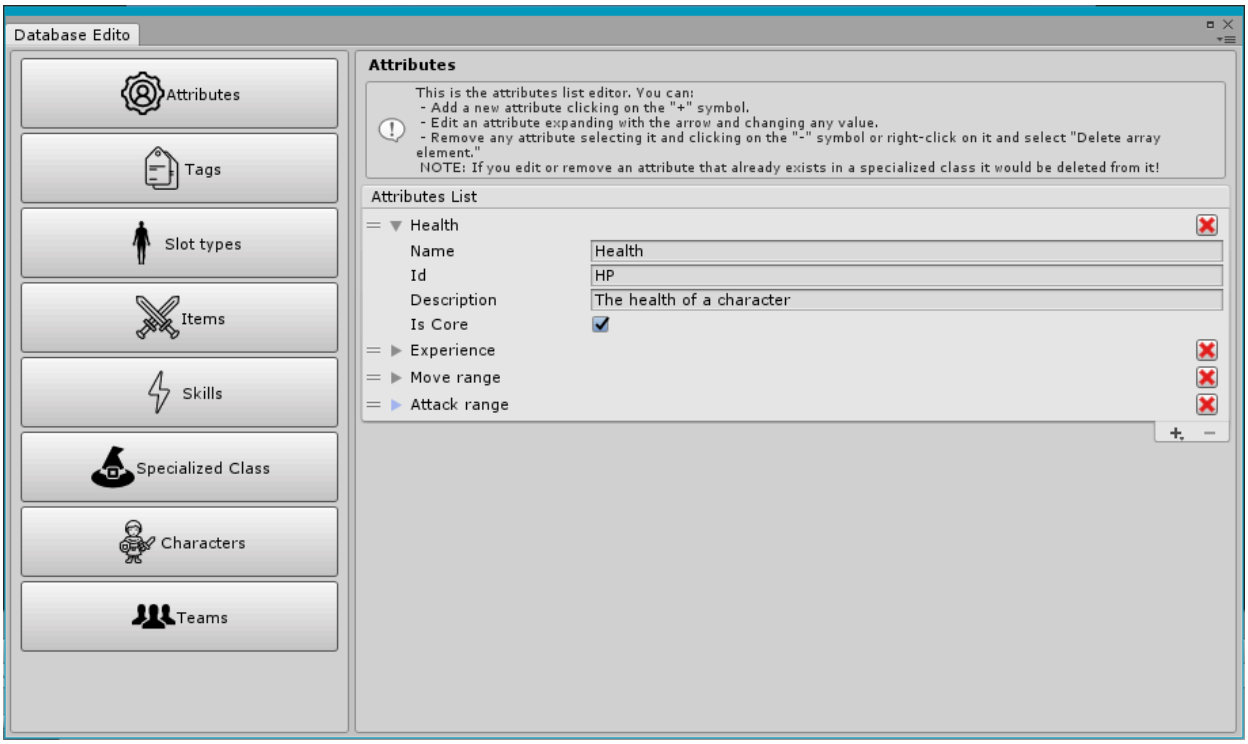


Figura 4.13: Editor de *Attributes* en la base de datos

El editor de ***Items*** fue una de las partes mas complicadas de realizar, al tener que mostrar de una manera intuitiva una de las partes mas complejas de la base de datos. Por un lado estaba la dificultad de añadir multiples *Tags* desde un solo *label* para lo que se utilizó una clase de la herramienta *uAdventure*[7] y se adaptó a las necesidades del proyecto. Esta clase permite, al introducir texto en este campo, desplegar una lista con los *Tags* existentes en la base de datos y seleccionarlos (véase Figura 4.14). En el caso de introducir un *Tag* no existente se muestra un *DisplayDialog* avisando de que no se ha encontrado dicho elemento y si se desea crearlo. También permite la introducción de múltiples *Tags* separándolos por comas.

[7]Herramienta desarollada por *e-ucm* en la Facultad de Informática de la UCM - `https://github.com/e-ucm/uAdventure`

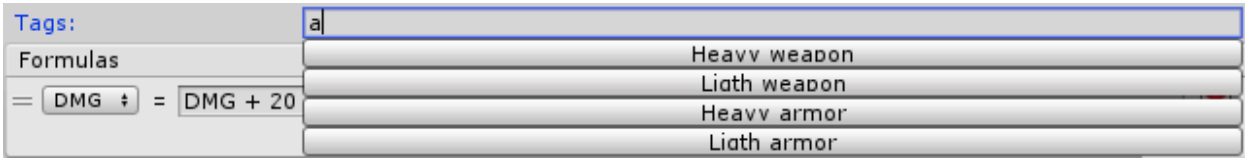


Figura 4.14: Lista desplegable para seleccionar un *Tag* existente

Por otro lado estaba la necesidad de introducir **fórmulas** que modificasen los *Attributes* correspondientes en función de las necesidades del desarrollador. Para ello se utilizó la librería *NCalc*[8] para el cálculo de dichas formulas que se aplicarán a los *Attributes* de cada personaje. Para comprobar que estas fórmulas son introducidas correctamente y que los *Attributes* existen en la base de datos se creó un *parseador* de fórmulas utilizando la idea de la clase *SequenceFormula* de *Puppeteer*[9], validando el *Id* de cada *Attribute* introducido en la fórmula y mostrando un mensaje de error en caso de no existir dicho *Attribute* o no ser valida la expresión matemática (véase Figura 4.15).

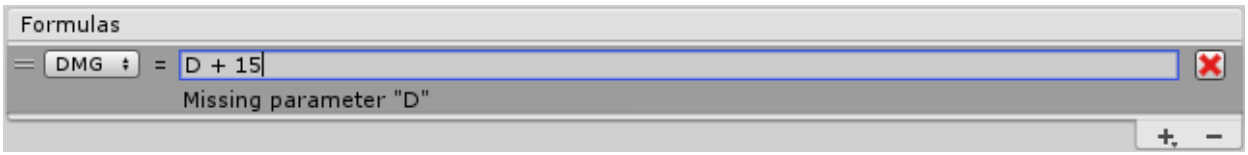


Figura 4.15: Mensaje de error al introducir un *Attribute* no existente en una fórmula

Por ultimo, existía la complejidad de mostrar todas las posibles combinaciones donde este *Item* podía ser equipado, ya que podían existir múltiples opciones disyuntivas inclusivas o exclusivas. En el objeto de la base de datos se decidió crear dos listas anidadas, una primera lista con todas las opciones donde un *Item* podía ser equipado y un lista anidada en cada una de las posiciones en las que de incluyen los *SlotTypes* que este objeto necesita.

[8]Evaluador de expresiones matemáticas para *frameworks* .Net - `https://archive.codeplex.com/?p=ncalc`

[9]Herramienta para Unity desarrollada por Víctor Manuel Pérez Colado - `https://github.com/Victorma/Puppeteer`

Así, si por ejemplo un objeto equiparse en dos *SlotTypes* diferentes, la primera lista contendrá dos elementos con una lista anidada cada uno, y en estas listas internas habrá un único *SlotType*. En caso de poderse equipar únicamente en dos tipos de *SlotTypes* a la vez, habrá un único elemento en la primera lista con dos elementos en su lista interna.

Dada la complejidad interna del funcionamiento de estas listas se decidió crear una interfaz lo más sencilla e intuitiva posible para que el desarrollador no necesitase comprender su uso interno. Esta interfaz consta de un apartado para dichas combinaciones en el que según se van añadiendo se muestra *OR* ó *AND* para entender que tipo de combinación se está usando (véase Figura 4.16).

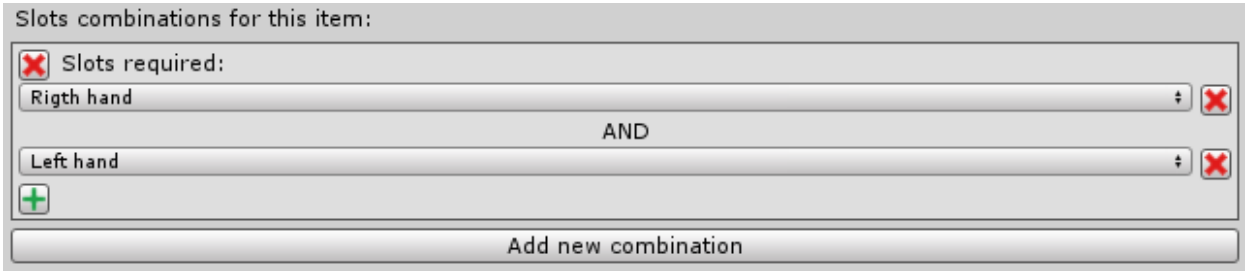


Figura 4.16: Posibles combinaciones para un *Item*

De este modo se creó un editor bastante intuitivo que permite crear *Items* abstrayéndose de la funcionalidad interna de la base de datos. Además se añadió la opción de incluir una imagen y mostrarla en el propio editor para representar el *Item* en el inventario una vez se esté ejecutando el juego (Figura 4.17).

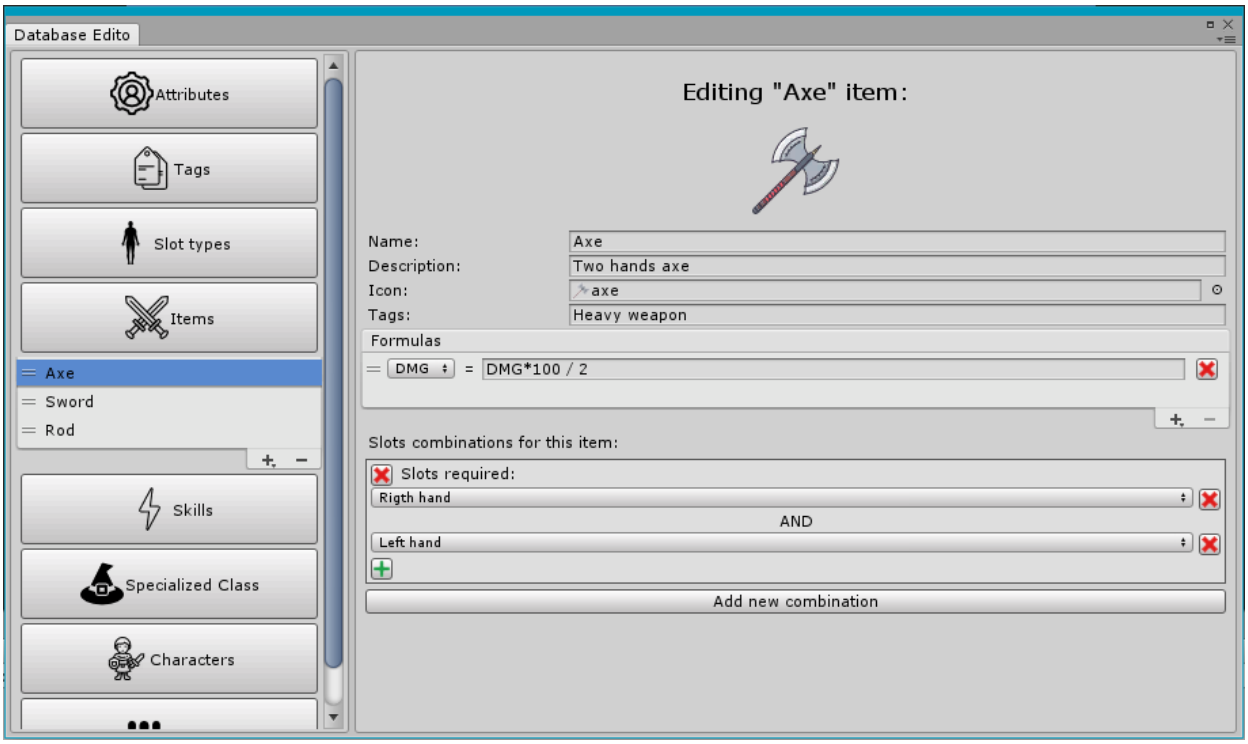


Figura 4.17: Editor de *Items* de la base de datos

Para el editor de ***Skills*** solo se implementó el panel correspondiente heredando del mencionado *LayoutWindow* y se asignó el editor por defecto que utiliza *Unity* para los objetos del tipo *ScriptableObject*. Al ser una clase *Serializable* y tener pocos atributos no era necesario implementar un editor personalizado.

Las ***Specialized Class*** requieren que se haya rellenado bastante información de los apartados anteriores por lo que, una vez insertada esta información, se pueden utilizar para crear clases especializadas específicas. Para asignar los *Attributes* se decidió implementar un *Reorderablelist* y modificarla para que, al seleccionar la opción de añadir uno nuevo, solo de la opción de incluir aquellos que no son *Core* (Figura 4.18).

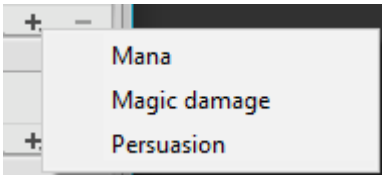


Figura 4.18: Lista desplegable para añadir un nuevo *Attribute* a una *SpecializedClass*

Una vez añadidos estos *Attributes* se permite modificar sus valores, pero no permite editar (y no aparecen) otros parámetros como son el *Nombre*, *Id*, *Descripción*, ni cambiarlo entre *Core* y no *Core* (Figura 4.19).

Figura 4.19: Editor de *Attributes* dentro de una *SpecializedClass*

También se implementaron otras tres *Reorderablelists*, una para asignar los *Tags* que habilitan a esta clase a utilizar determinados *Items* o *Skills*, otra para asignar los *Items* por defecto que llevará esta *Specialized Class* y una última para añadir los *Skills* que llevará por defecto. En estas tres listas se implementó el botón de añadir de la misma forma para que, una vez pulsado, muestre la información ya existente en la base de datos con una presentación similar a la Figura 4.18.

Para los ***Characters*** también se requiere de la información de los apartados anteriores y es utilizada para crear los personajes específicos. Cada personaje cuenta con *Reorderablelist* para los *Attributes*, en los que se incluyen los llamados *Core* (que todo personaje debe llevar obligatoriamente) y son asignados de manera automática, los heredados de la *Specialized Class* y aquellos que el desarrollador desee incluir. Para todo ello el objeto *Character* cuenta con una función *refresh* a la que se llama al instanciar el objeto y cada vez que un *Attribute* o una *Specialized Class* son modificados o añadidos.

Además cuentan con otro *Reorderablelist* para mostrar los *Attributes* que serán utilizados en el juego y que son el resultado de aplicar las fórmulas correspondientes al *Character*. Este cálculo de fórmulas es realizado cuando el valor de un *Attribute* del *Character* es modificado o cuando un *Item* es asignado para mantenerse actualizados.

El editor de *Characters* (Figura 4.20) cuenta con tres *Reorderablelist* más, uno para los *Slots* específicos del personaje con la misma funcionalidad que el de *Specialized Class*, otro para mostrar la información de los *Slots* definidos en las *Specialized Classes* asignadas al que se deshabilitó la edición para ser únicamente informativo y un último para añadir o modificar *Specialized Classes* al personaje.

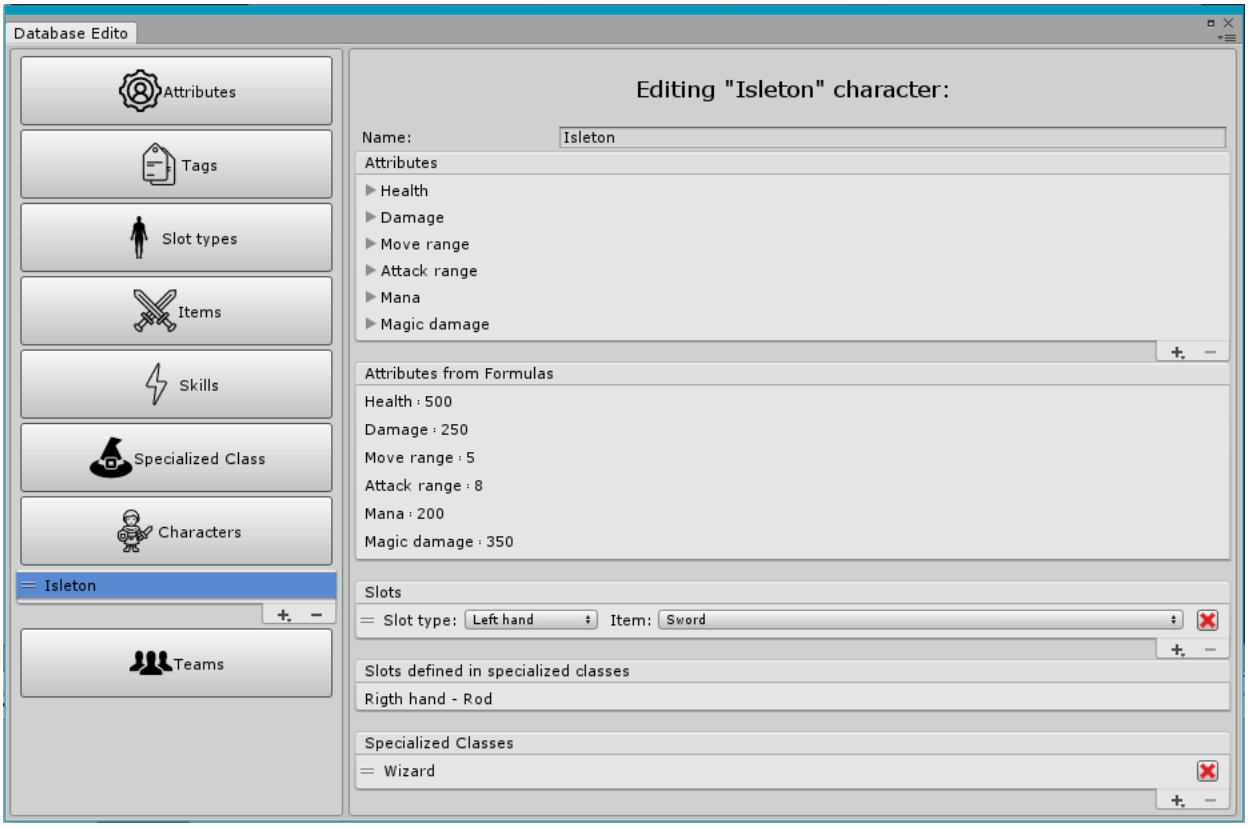


Figura 4.20: Editor de *Characters*

El último apartado del editor es utilizado para manejar la información de los ***Teams***, cuenta con una serie de atributos básicos como son el *Id* y *Name*; un *booleano* que permite elegir si un *Team* es manejado por el jugador o por la inteligencia artificial y un *Reorderablelist* para incluir los personajes que forman parte de ese *Team*.

4.2.5. Gestor del juego

Sistema de combate

La primera necesidad el sistema de combate era la coherencia entre los datos almacenados en la base de datos y el sistema que iba gestionar el juego. Por ello, lo primero que se implementó fue una *EditorWindow* de *Unity* (Figura 4.21) en la que se definió el tipo de juego, todos los tipos de asignación de turnos disponibles y una serie de *Attributes* necesarios para todo juego de rol táctico. Se decidió crear esta ventana para dar libertad al desarrollador de establecer los *Attributes* como desease sin necesidad de depender del idioma o unos *Id's* o *Nombres* predefinidos.

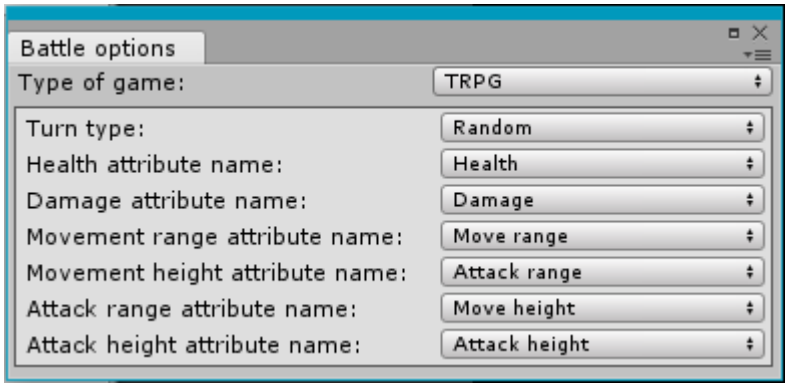


Figura 4.21: Ventana de configuración del combate

Solo se implementaron los tipos de turno *Random*, que genera el turno de forma aleatoria, y *Attribute*, que permite seleccionar el *Attribute* que se desee y un orden creciente o decreciente para gestionar el turno en función del valor del mismo en cada *Character*. Tampoco se implementaron más estilos de juego que el denominado como rol táctico, sin embargo, se dejó abierta la posibilidad de implementar mas estilos de juegos o turnos en un futuro extendiendo esta clase y configurándola para cada necesidad.

Para la gestión del sistema de combate se implementó una clase llamada *GamePlayManager* encargada de la gestión de la información contenida en la base de datos y su utilización para el desarrollo de los combates dentro del juego.

GamePlayManager consta con una serie de métodos públicos que pueden ser llamados para empezar un combate y recibir la información al final del mismo.

El método principal es *StartCombat* el cual, una vez llamado, busca la información de los personajes del escenario y sus respectivos equipos. Se decidió también realizar en este punto una serie de comprobaciones en caso de no haberse configurado correctamente todos los parámetros y avisar al desarrollador antes de iniciar el combate evitando así errores no esperados (Figura 4.22).

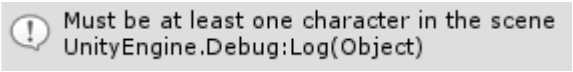


Figura 4.22: Ejemplo de aviso al no configurar correctamente el escenario

Una vez realizadas todas las comprobaciones se añaden los personajes a una lista, según el orden de los turnos seleccionado en el editor, y se llama mediante un bucle a cada uno de los personajes del escenario. Según se van quedando sin vida se van eliminando de la lista a los personajes y se termina el bucle cuando no quedan personajes de alguno de los equipos en dicha lista.

Sistema de turnos

Este sistema se encarga de manejar la información correspondiente a cada turno dentro del combate. Recibe el *Character* que realiza la acción en el turno actual y en función, del tipo de control que tenga, realiza una acción u otra.

Para los *Characters* manejados por el jugador el sistema de combate muestra un menú (Figura 4.23) con las acciones correspondientes. Esté menú se añadió mediante código para evitar que el desarrollador tuviese la necesidad de crear un botón para cada acción y tuviese restricciones a la hora de nombrar dichos botones. No obstante se optó por una implementación sencilla para poder modificarse con facilidad.

Figura 4.23: Menú de acciones *in-game*

Esté menú se añadió mediante un *GameObject* que automáticamente es añadido a la escena y se le agregaron los componentes *EventSystem*, *Canvas*, *Button* y *Text* para su correcto funcionamiento.

La secuencia de acciones implementada para un jugador (véase Figura 4.24) consiste en realizar un movimiento, si se desea, y posteriormente realizar un ataque, lanzar una habilidad o finalizar el turno.

Para el uso de los botones por parte del jugador se implementaron una serie de funciones con llamadas asíncronas para que, una vez pulsado, se llame al conector encargado de realizar las acciones visuales. Así, si el jugador selecciona realizar un movimiento y pulsa un destino el *Character* realizará dicha acción, sin embargo, si antes de seleccionar el destino decide cambiar el tipo de acción se realizará una nueva llamada asíncrona para mostrar las opciones correspondientes a esta nueva acción.

De este modo, el movimiento del *Character* se implementó una función a la que se le pasa como parámetro una función *callback* y, en caso de que el jugador seleccione una celda de destino esta función será la encargada de llamar al conector y realizar el movimiento. Del mismo modo las acciones de ataque y habilidad pasan una función como parámetro que, en caso de seleccionar el destino, devuelven una celda o un *Character* donde se realizará la acción.

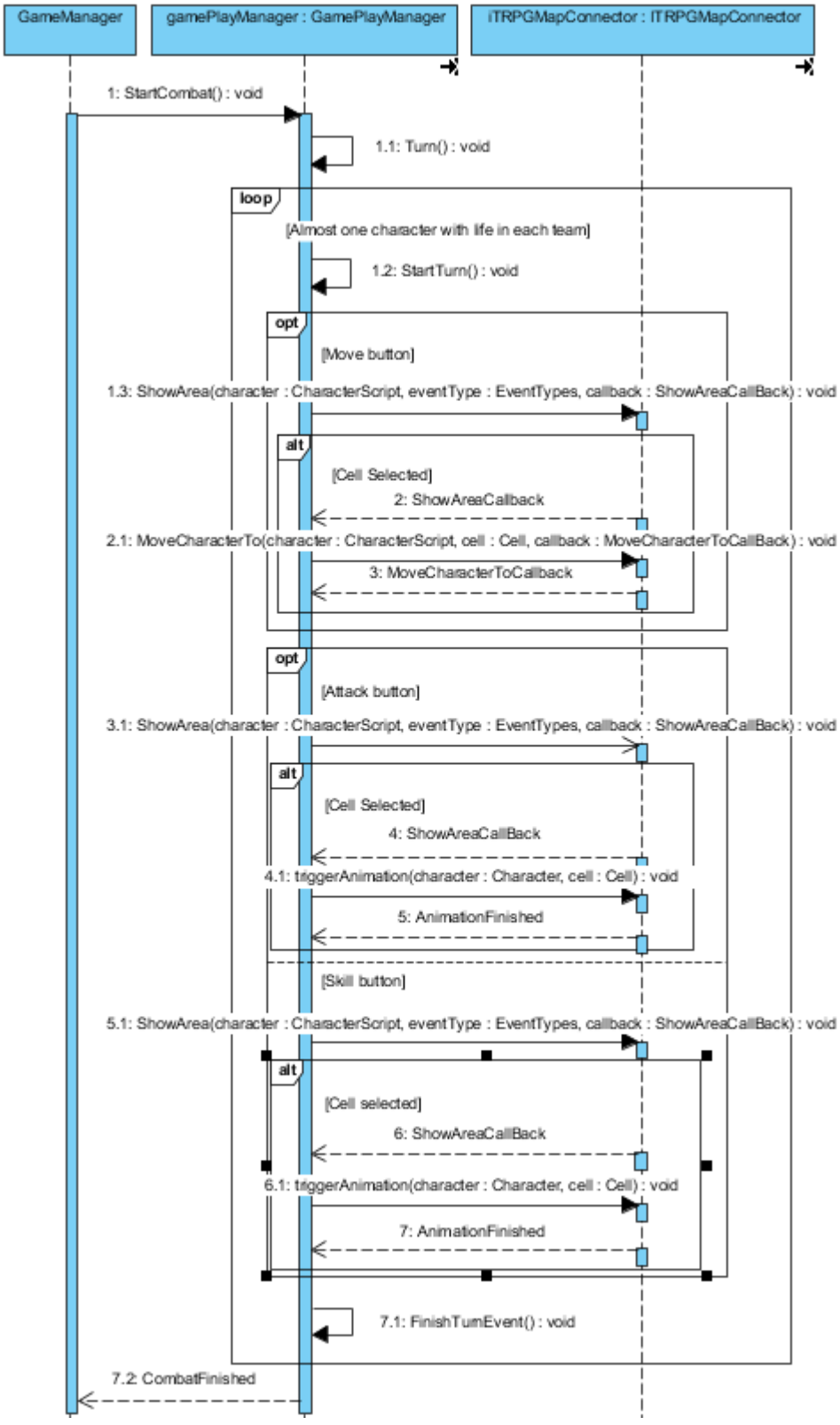


Figura 4.24: Secuencia de un turno para un jugador

Inteligencia Artificial

La inteligencia artificial de los jugadores que no son controlados por el jugador se implementó de un único modo, pero dejando la posibilidad de modificarse o implementar otros modos distintos de una forma sencilla creando nuevos métodos para ello.

El modo implementado se basó en la búsqueda de *Characters* cercanos al actual y el rango de ataque del *Character* actual. Si existe solo uno, será atacado y si existen varios, se atacará al que mas vida tenga en el momento de lanzar la acción. Si no existe ningún *Character* en el rango de ataque se buscará mediante el algoritmo *A Estrella (A*)* [32] y la distancia *Manhattan* [33] al *Character* mas cercano, es decir, al que menos turnos tengan que transcurrir para llegar hasta él. Una vez realizado el movimiento se comprueba de nuevo si existen *Characters* a los que atacar y se realiza la acción si es posible. En caso de no existir ningún *Character* accesible por medio del movimiento o no poder atacar en el turno actual el *Character* se defenderá, es decir, pasará el turno.

Este cálculo se realiza en cada turno por lo que, cuando los *Characters* del jugador o de la inteligencia artificial cambian de posición es calculado de nuevo para comprobar la mejor acción en cada nuevo turno.

4.2.6. Conexión con otras herramientas

Para la conexión con otros *frameworks* o herramientas encargados de la parte visual se decidió crear una interfaz que deberá ser heredada e implementada para cada *framework* en concreto (Figura 4.25), permitiendo así lograr una total abstracción entre la herramienta desarrollada en este proyecto y cualquier otro *framework* para el apartado visual del juego que se desee utilizar. Además permitió evitar así uno de los principales problemas que nos encontramos al inicio de este proyecto y es la actualización o cambio importante en la funcionalidad del *framework* incorporado al proyecto.

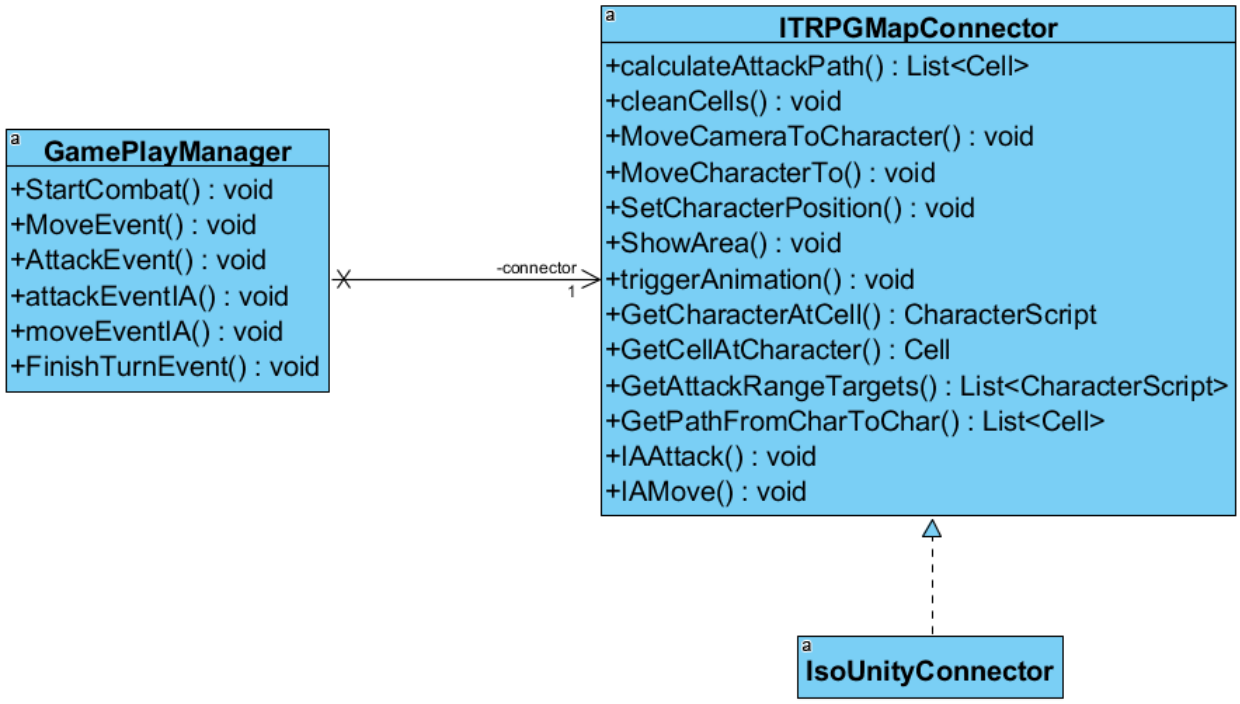


Figura 4.25: Diagrama de clases de la interfaz para conectar con otro *framework*

En este proyecto se decidió utilizar *IsoUnity* como *framework* software para la gestión de la parte visual, pero gracias al diseño de esta implementación puede ser totalmente eliminado del proyecto e incorporado cualquier otro o incluso desarrollar uno propio si se desease.

Para la conexión con *IsoUnity* se heredó e implementó la clase correspondiente (*IsoUnity-Conector*) mediante métodos asíncronos y *GameEvents*, siendo la encargada de realizar las animaciones correspondientes a cada acción, el movimiento de los *Characters* por el escenario, posicionarlos en la posición deseada, mostrar el área de acción o devolver información de las celdas del escenario.

Una de las partes más destacables de este conector es el cálculo de áreas de acción, que se realizó mediante el algoritmo *A Estrella (A*)* y la distancia *Manhattan*, al igual que con la inteligencia artificial de los personajes no jugables. Una vez dibujada este área y en función de la acción a realizar se permitió seleccionar un conjunto de celdas y, en caso de seleccionar alguna, se devuelve la información correspondiente a la misma (Figura 4.26).

Figura 4.26: Representación del área de movimiento y selección de celda de un *Character*

Capítulo 5

Pruebas, resultados y discusión

En este capítulo se mencionan las pruebas realizadas a la herramienta *TRPG Maker* que validan los objetivos y requisitos definidos en el Capítulo 3, así como los resultados de este proyecto.

5.1. Pruebas

Por un lado, los desarrolladores hemos realizado pruebas constantes a cada nueva funcionalidad añadida a la herramienta, por otro, se han presentado dos grandes fechas en el calendario para realizar pruebas con usuarios reales, una sesión pública ante voluntarios y la demostración final.

5.1.1. Pruebas generales

A medida que se iban implementando y añadiendo nuevas funcionalidades a la herramienta, estas iban comprobándose constantemente con la finalidad de encontrar fallos y así poder subsanarlos.

En las fases iniciales de la creación de la base de datos y su editor, las pruebas eran mas sencillas al únicamente realizar prueba de escritura y lectura de datos y su consistencia. A medida que el proyecto fue avanzando se pudieron realizar pruebas mas profundas, comprobando el funcionamiento de las relaciones y los datos con el gestor del juego.

Un buen paso en las pruebas que se realizaron fue la unión del proyecto con *IsoUnity*, que permitió crear escenas y tratar todo el proyecto de una forma más visual, intuitiva y completa. A partir de esta unión se detectaron bastantes fallos en las fases iniciales que se fueron corrigiendo a medida que aparecían.

5.1.2. Sesión pública con usuarios reales

La prueba pública, marcada como hito en el plan de proyecto en el Capítulo 3.2, consistió en el contacto de la herramienta con otras personas en búsqueda de errores. La prueba se realizó a un grupo de 6 personas, todos con aptitudes y conocimiento de Unity y programación. Se les dio acceso a un ordenador con la herramienta en ejecución, se les facilitó un manual de usuario (véase Apéndice D) así como una guía para la prueba (véase Apéndice E).

La prueba consistió en permitir a los participantes modificar la base de datos, añadiendo o editando los elementos existentes. Una vez configurado los datos, el participante podría realizar un combate contra la IA, añadiendo el personaje al escenario de Unity y ejecutando la aplicación. Al terminar la prueba, se les pidió a los participantes que rellenasen un formulario, creado mediante Google Forms, con la finalidad de conocer su opinión sobre la herramienta.

El formulario se dividió en 5 temas, permitiendo seleccionar una respuesta predefinida o escribir una respuesta propia. El objetivo era conocer la experiencia de usabilidad, la validez del diseño y de la estructura organizativa de la herramienta, así como saber si los participantes usarían la herramienta para desarrollar sus propios proyectos, y obtener su opinión general sobre el proyecto TRPG Maker.

Antes de entrar en detalle a los resultados obtenidos en el formulario, cabe destacar que en el transcurso de la prueba, la herramienta falló y quedo inutilizada cuando lo estaba utilizando uno de los usuario.

Este fallo grave permitió conocer un problema existente y así poder arreglarlo a los pocos días. El error consistía en el cambio de nombre de los atributos, dado que al modificarlo no actualizaba la base de datos ni sus referencias con el resto de objetos y estas no se encontraban.

5.2. Resultados

El principal resultado de este proyecto es la herramienta desarrollada que se encuentra en el repositorio de GitHub: `https://github.com/Narratech/TRPGMaker`. Los resultados de las pruebas generales, de la sesión pública y la demostración final se describen a continuación.

5.2.1. Pruebas generales

Los principales fallos en estas pruebas generales fueron con el editor de la base de datos, mostrando errores cuando se eliminaba algún dato en concreto, se añadía un objeto vacío o pequeños errores que no se detectaron a la hora de implementarlo. Por otro lado en el gestor del juego y la conexión con *IsoUnity* no se detectaron prácticamente problemas ya que estos eran subsanados a medida que se iban implementando, ya que las pruebas podían realizarse con cada nueva funcionalidad del código.

5.2.2. Sesión pública con usuarios reales

A excepción del problema que tuvimos con un usuario al que se le bloqueó la herramienta, en las demás pruebas la herramienta funcionó correctamente, arrojando el formulario que pasamos a los probadores, los siguientes resultados:

- **Usabilidad**. Los resultados fueron dispares, aunque se pudo observar la necesidad global en mejorar y distribuir mejor la información que aparece en pantalla.
- **Diseño**. Por lo general, el diseño de la herramienta fue de agrado a los participantes.

- **Estructura de la herramienta**. Por lo general, la estructura de la herramienta pareció correcta a los participantes.
- **¿Usarías esta herramienta?**. A la pregunta, la mayoría de los participantes respondieron de modo afirmativo.
- **Opinión general**. La opinión general por parte de los participantes de la herramienta fue positiva.

Los resultados completos se encuentran en el Apéndice F.

Con la información obtenida tras la prueba, se realizaron diversos cambios en la herramienta para su mejora, como el añadido de un botón más representativo para eliminar elementos de la base de datos.

- Se añadieron botones con el símbolo x para poder borrar elementos y mejorar así la visibilidad de esta opción.
- Se cambió el diseño para las combinaciones de los objetos, mejorando su visualización.
- *Attribute* se cambió y se añadió un nuevo *Attribute Value* que es el que se añade a los objetos y en los que se guardan los valores y una referencia al *Attribute* original
- Se resolvió un error que se generaba al añadir objetos vacíos

El resultado final de TRPG Maker y de este proyecto es una herramienta funcional, que permite la creación de videjuegos de rol tácticos.

En los próximos capítulos se discutirá y llegará a la conclusión sobre el resultado, así como las posibles ampliaciones a la herramienta.

5.2.3. Demostración de la herramienta

La demostración consiste en un pequeño juego realizado con la herramienta en el que se puede realizar un combate entre dos equipos, uno controlado por el jugador y otro por la inteligencia artificial, en un mapa creado con la herramienta *IsoUnity* (véase la Figura 5.1)

Los dos equipos cuentan con varios personajes, creados con los datos existentes en la base de datos, siendo cada uno de los personajes único por la configuración y asignación de elementos.

Cuando el jugador tenga control de un personaje de su equipo aparecerá en pantalla varias opciones que le permiten realizar en el juego: mover, atacar, habilidad o pasar turno. Si se selecciona mover, aparecerá en el mapa del juego un área de color azul indicando la posición donde se puede mover el personaje, después de la animación de movimiento, se podrá pasar turno, atacar o usar una habilidad, deshabilitando la opción de movimiento.

Si se selecciona atacar, aparecerá un área de color rojo en el mapa mostrando donde puede atacar el personaje, si en ese área se encuentra un personaje enemigo y se pulsa sobre él, se le infligirá un daño. Este daño es el total del atributo *daño* después de realizar las fórmulas existentes en el personaje. La salud del enemigo atacado se verá mermada según el daño. En el caso de que la vida del enemigo atacado se quedase a cero, éste se quedaría fuera de combate y realizará una animación de muerte.

Figura 5.1: Captura de la demostración de la herramienta

5.3. Discusión

Con los requisitos fundamentales de la herramienta ya realizados, y con la posibilidad de poder utilizarla de manera correcta y poder realizar simples juegos con ella, se ha tenido conciencia de diversas mejoras y ampliaciones para poder llegar a tener una herramienta de desarrollo completa.

Comparando con el prototipo del que se partía, se ha logrado hacer una estructura más sólida y genérica, con menor número de líneas de código y aumentando su funcionalidad, con inteligencia artificial, animaciones, muerte de los personajes, definición y uso de fórmulas, uso de habilidades, finalización del combate y una estructura más visual y accesible de la herramienta. Además se ha logrado implementar *TRPG Maker* de un modo completamente independiente de *IsoUnity*, permitiendo adaptarse a otras herramientas de manera sencilla, pero debiendo implementar la interfaz definida para ello.

Además de encontrar mejoras y ampliaciones, la *herramienta* actual se encuentra en una posición dentro del mercado muy aventajada, ya que permite definir los elementos necesarios para realizar un videojuego de rol táctico, o cualquier videojuego con características y elementos similares, debido a la filosofía que se ha usado en este proyecto para no cerrarse sólo a un género, sino ser lo suficientemente ambiguo como para poder ser usado de la manera deseada por el desarrollador.

Esta filosofía abierta en el proyecto ha estado presente desde el comienzo, se quería realizar una herramienta enfocada a realizar videojuegos de rol tácticos, pero sin impedir su uso a otros géneros. Por ello se ha dejado la puerta abierta en la *herramienta* a la posibilidad de implementar nuevas funcionalidades para diferentes tipos de juego. Puede verse esta intención en apartados como *"battle options"* en la que se debe especificar el tipo de juego.

Al igual que se quería permitir el uso de la herramienta facilitando la posibilidad de ampliación según el tipo de juego que se quiera crear, se definió una interfaz para permitir el uso de la herramienta deseada para *gestionar el juego*, desde la pestaña *"connector options"*. Permitiendo desde aquí seleccionar con que herramienta se conectará TRPG Maker, teniendo como ejemplo de uso de la interfaz la conexión con la herramienta IsoUnity, dejando totalmente abierta la posibilidad de ampliación para el uso con la herramienta que se desee.

Dadas las referencias descritas en el 2, en concreto Final Fantasy Tactics y la actual versión de TRPG Maker, podemos recrear una batalla al estilo de este videojuego con algunas salvedades, como ver el estado del personaje, la experiencia que se obtiene en combate y todo lo relacionado con ello. Pero sí es posible crear una escena con varios personajes de diferentes equipos, definir habilidades, atributos y objetos, y equipar a los personajes con objetos que modifiquen el valor de sus atributos. En la batalla se permite mover a los personajes seleccionando la casilla dentro del rango del movimiento del personaje, atacar y usar habilidades. El enemigo o personaje no jugable ataca a aquellos personajes que no pertenezcan a su equipo.

Para realizar este proyecto se ha necesitado mucho tiempo para aprender como añadir extensiones a Unity, ya que la única experiencia que existía al comienzo del proyecto era con la creacción de juegos usando Unity, y la creacción de extensiones y editores para Unity cambia considerablemente en relación a su uso para el desarrollo de videojuegos.

Además de necesitar un buen periodo de aprendizaje en Unity, comprender el funcionamiento de la herramienta IsoUnity fue costoso debido a la poca documentación que se encontró sobre ella y su complejidad, se realizó una primera prueba creando una pequeña escena descrita en el Capítulo 4.2.2, pero al tener contacto con el co-director y desarrollador de la herramienta Victor Manuel Pérez Colado se empezó a comprender al completo todas las funcionalidades de las que dispone.

Con el estudio del prototipo TRPG Maker también tuvimos un periodo de prueba, al elegir este proyecto teníamos la intención de actualizar la herramienta, no rehacerla por completo, y ese periodo en el que tuvimos que aprender a utilizar la herramienta y buscar los requisitos para la nueva versión hizo que invirtiéramos gran cantidad tiempo en trabajo que finalmente no hemos utilizado.

Capítulo 6

Conclusiones

Este proyecto comenzó con la intención de ampliar y mejorar la herramienta un prototipo ya existente, pero la escasa documentación y lo ligado al código de otras herramientas que se encontraba esta primera versión, hizo que el proyecto sufriese un importante cambio de enfoque. Tras mucho trabajo de investigación y pruebas con el prototipo se procedió a rediseñar y reescribir todo el código, montando una nueva arquitectura y buscando la forma de que **TRPG Maker** fuese lo más independiente que fuera posible de otras herramientas, exceptuando Unity, el propio entorno de desarrollo. Para ello se tomó la herramienta anterior como una referencia a los requisitos necesarios, se mantuvieron abundantes reuniones con el director y co-director del proyecto para profundizar en los requisitos, así como constante contacto entre los integrantes del grupo para diseñar y especificar la herramienta.

Tras esta primera *crisis* sobre el porvenir del proyecto, el trabajo siguiente consistió en definir y especificar que es necesario para todo videojuego de rol táctico, y tras ese proceso de investigación comenzó la implementación de la herramienta.

Se ha conseguido implementar la base de datos, así como su editor, en una ventana visual e intuitiva, permitiendo crear personajes completos, con *atributos*, *etiquetas*, *objetos*, *habilidades* y *tipos de huecos*, que en su conjunto definen al personaje. También se ha conseguido implementar *equipos*, diferenciando si son jugables o no, desde el propio editor de la base de datos.

Uno de los mayores problemas que existía en el prototipo TRPG Maker, era lo muy acoplado que estaba a la herramienta IsoUnity, en este proyecto se ha conseguido seguir utilizando IsoUnity, pero de modo totalmente independiente, permitiendo poder utilizar la herramienta que se desee como complemento a TRPG Maker. Para ello se ha implementado un conector, explicado en el Capítulo 4, de forma que se puede conectar a cualquier otra herramienta o incluso a una propia del desarrollador simplemente implementando una interfaz de *C#* existente en el código de la herramienta.

También se ha conseguido implementar un gestor de juego, que permite realizar combates por turnos entre equipos que se encuentren en el mapa, con la existencia de inteligencia artificial para los equipos que no sean controlados por el jugador.

Se ha podido crear una escena de ejemplo jugable que ilustre la utilidad y las capacidades de esta herramienta, consistiendo en un combate entre dos *equipos* con varios personajes en cada uno de ellos, donde se permite realizar las acciones existentes actualmente en combate, tales como atacar, mover, lanzar habilidades y pasar turno. La demostración permite el uso de uno de estos equipos a un jugador, dejando el uso del otro equipo a la inteligencia artificial implementada. La partida concluye al derrotar a todos los enemigos del equipo contrario, dando como resultado la victoria del jugador, o dejando a cero la vida de todos los personajes jugables, con el resultado de derrota para el jugador.

Con todo lo anterior implementando, queda abierta la posibilidad de mejoras en la herramienta. La base de datos y el editor funcionan correctamente, pero se le podría dar más profundidad a los datos que puede almacenarse en la base de datos. A nivel jugable, puede añadirse mucho más fondo, ya que actualmente consiste en simples batallas. Se necesitará también implementar en futuras ampliaciones toda lo relacionado con las *habilidades pasivas* y la gestión de niveles y experiencia. Además existe la posibilidad de poder portar la herramienta a otros *engine* como Unreal Engine.

En definitiva, se ha creado una herramienta con necesidad de ampliaciones futuras, pero con las suficientes funcionalidades para ser utilizada actualmente en videojuegos simples y la gran proyección que tiene la herramienta si se realizan mejoras y ampliaciones en el futuro.

Aportaciones individuales de los autores

1. Marcelino Pérez Durán

El comienzo del proyecto se baso en probar el prototipo TRPG Maker existente y la herramienta IsoUnity, para ello cada uno probó una de esas herramientas. Me encargué de la herramienta TRPG Maker en su totalidad, con ello conocí las características que debe tener una herramienta destinada a realizar videojuegos de rol táctico, pero también, discutiéndolo con mi compañero, comprendimos que existía la necesidad de reimplementar la estructura de la herramienta.

Además de probar TRPG Maker, investigué sobre la herramienta RPG Maker, probándola y aprendiendo sus funcionalidades para obtener referencia y especificaciones en la herramienta que hemos realizado, encontrando muchas características necesarias. Asimismo probé varios videojuegos TRPG para conocer sus funcionalidades y disponer información para el *estado de la cuestión* de la memoria, en concreto Final Fantasy Tactics, en el que estudie las características que tenía, y junto a haber probado otros juegos, encontré similitudes entre ellos para conocer las características básicas existentes en este género, y buscar el modo de *generalizar* las características propias de cada uno de ellos.

Junto a Juan José, definimos los requisitos de la herramienta, el comportamiento y el modo de conectarlo a IsoUnity o a cualquier otro *framework*. Para ello mantuvimos reuniones con el director y co-director, y muchas conversaciones y reuniones entre nosotros dos para especificarlo y buscar el mejor modo en el que se debería realizar. Realizamos diagramas de secuencia para definir las interacciones que debería tener TRPG Maker con otras herramientas para desacoplarlas completamente.

En la implementación, ayudé en dar forma a la base de datos y al editor de la misma, en especial con los elementos *tags* y *slot types*. Revisé y probé la herramienta en cada nueva funcionalidad añadida. Implementé el *sistema de turnos* de la herramienta para el combate.

En las presentaciones que tuvimos para mostrar el estado del proyecto al director y miembros de Narratech, marcados como hitos, me encargué de construir una escena en Unity para mostrar una demo de la herramienta, con elementos visuales, iluminación y partículas para darle mayor profesionalidad a TRPG Maker.

Para la realización de la prueba guiada con usuarios reales, redacté el documento que se entregó a los usuarios para realizar la prueba, así como la *guía del usuario* de la herramienta TRPG Maker, explicando el funcionamiento completo de la herramienta en el momento de la prueba.

Respecto a la memoria, me encargué de redactar la *introducción*, el *estado de la cuestión*, leyendo y buscando información sobre herramientas y videojuegos y *objetivos y especificaciones*, describiendo la metodología usada para llevar a cabo el proyecto, así como definiendo los objetivos y la especificación de requisitos a partir de documentos y textos que junto a mi compañero, redactamos en las reuniones con el director y codirector, y en reuniones propias, también basándome en las herramientas y videojuegos que probé para conocer las especificaciones y necesidades en los TRPG. En el capítulo 4, me encargué de redactar el *análisis y diseño* de la herramienta, y realizando los casos de usos que ahí se muestran. También me encargué de redactar el capítulo *pruebas y resultados*, narrando la prueba realizada a usuarios reales, así como el resultado de la misma. En los capítulos 6 y 7 redacte la discusión y conclusión de este proyecto.

El conjunto de la memoria (texto, figuras y bibliográfia) fue revisado en equipo, discutiendo sobre el estilo, la maquetación en LaTex y la redacción.

2. Juan José Prieto Escolar

Al comienzo del proyecto me encargué del estudio de videojuegos de rol táctico así como del estudio de otras herramientas para la realización de los mismos, siendo especialmente relevantes el estudio de *RPG Maker*, *IsoUnity* y el prototipo de *TRPG Maker*.

Probé en su totalidad la herramienta *IsoUnity*, conociendo todas las características que este *framework* nos podía ofrecer, creando una pequeña prueba de concepto sobre el mismo, conociendo a fondo su funcionamiento, estudiando la manera de añadirlo al proyecto y discutiendo con mi compañero la posibilidad de mantenerlo.

Para definir los requisitos de la herramienta mantuvimos múltiples reuniones con el director y el coodirector, habiendo asistido y participado de forma activa en todas ellas. Junto con mi compañero Marcelino mantuvimos conversaciones y reuniones para definir el diseño de la herramienta, así como los requisitos que veíamos fundamentales y la forma de realizarlos.

Para la implementación de la base de datos estudié en profundidad el funcionamiento de la existente en el prototipo de *TRPG Maker*, usando las ideas planteadas en el mismo para crear una base de datos que guardase toda la información de la herramienta y las relaciones entre los distintos datos. Junto a mi compañero realizamos una ardua tarea para diseñarlo de la forma mas eficiente posible evitando la redundancia de los datos almacenados y finalmente implementé la misma.

Realicé una amplia investigación sobre el editor de *Unity*, leyendo la documentación de la propia herramienta y probando otros *frameworks* que hacen uso del mismo como se explica en el capitulo 4.2.4. Tras esta investigación diseñé y realicé un editor intuitivo utilizando la información recabada, siendo una de las partes mas difíciles de desarrollar por la falta de conocimientos sobre el editor y la dificultad de relacionar todos los datos de una forma sencilla y muy visual.

Junto con mi compañero realizamos reuniones con el codirector para añadir al proyecto *IsoUnity* de la forma mas independiente posible. Tras estas reuniones realicé el conector encargado de manejar la información enviada y recibida sobre este *framework*, realizando múltiples pruebas iniciales para comprobar el correcto funcionamiento de este conector. Me encargué tanto de conectarlo con *IsoUnity* como de conectarlo con nuestra propia herramienta.

Realicé gran parte del gestor del juego, encargándome de vincular la información almacenada en la base de datos con el sistema de combate, mostrar el menú al usuario y realizar las acciones correspondientes al combate utilizando el conector. También diseñé tanto el uso de fórmulas en el editor como su correcto cálculo a la hora de añadirlas a un personaje o utilizarlas en el gestor del juego, para ello realicé una amplia investigación sobre la herramienta *NCalc* y su uso en otros proyectos.

Por último, me encargué de implementar la inteligencia artificial explicada en el capitulo 4.2.5, el uso de animaciones según la acción realizada y la barra de vida que aparece sobre los personajes cuando se ejecuta la herramienta.

Para comprobar el correcto funcionamiento de la herramienta me encargué de realizar múltiples pruebas con cada nueva funcionalidad y junto a los resultados obtenidos por mi compañero revisé y corregí los fallos detectados, o se diseñó y analizó de nuevo e implementé una versión más estable de estas funcionalidades.

En las reuniones marcadas como hitos para presentar la herramienta me encargué de crear versiones funcionales de la herramienta y que se pudiesen presentar ante los miembros de Narratech sin fallos en las partes ya implementadas. Para la realización de la prueba guiada con usuarios reales también me encargué de implementar una versión lo mas parecida posible a la versión final de la herramienta y a explicar el funcionamiento de la misma a los participantes.

Respecto a la memoria me encargué de realizar los diagramas, *mockups* y capturas de la herramienta mostradas en los capítulos 3 y 4. Además redacté toda la parte de implementación del capítulo 4 y pequeñas partes de otras secciones. Revisé todo el documento cambiando partes que vi necesarias y comentando a mi compañero como plantear ciertos capítulos o secciones. Junto con mi compañero discutimos el estilo y la maquetación del documento y realizamos una amplia búsqueda de imágenes y referencias utilizadas durante el desarrollo de la herramienta.

Bibliografía

[1] Steven L. Kent. *The Ultimate History of Video Games*. Three Rivers Press, 2001. ISBN 978-0-7615-3643-7.

[2] Angelo M. D'Argenio. Gaming literacy: Spacewar! the first video game worthy of the name. *Game Crate*, 6 2017. URL `https://www.gamecrate.com/gaming-literacy-spacewar-first-video-game-worthy-name/16643`.

[3] Scott Adams. Spacewar! hand coded manuscript, 1975. URL `http://exoticsciences.com/swgmanu.html`.

[4] Fusajirō Yamauchi. Nintendo, 1889. URL `www.nintendo.es/`.

[5] Jeremy Parish. Five critical moments in platform game history. *US Gamer*, 7 2014. URL `https://www.usgamer.net/articles/five-critical-moments-in-platform-game-history`.

[6] Ken Williams and Roberta Williams. Sierra entertainment inc, 1979. URL `http://www.sierra.com/`.

[7] Unity Technologies. Unity, 2005. URL `http://unity3d.com/es`.

[8] ASCII Corporation and Enterbrain. Rpg maker. URL `www.rpgmakerweb.com`.

[9] Epic Games. Unreal engine, 1998. URL `https://www.unrealengine.com`.

[10] Gabe Newell. Steam, 2003. URL `https://store.steampowered.com/`.

[11] CD Projekt. Gog.com, 2008. URL `https://www.gog.com/`.

[12] Javier Druet Honrubia and Luis Alfonso González de la Calzada. *Desarrollo de una Herramienta de Creación de Videojuegos de Rol Táctico para Escenarios Isométricos Compuestos por Bloques*. Facultad de Informática de la Universidad Complutense de Madrid, 2017. [Trabajo fin de Grado].

[13] Iván José Pérez Colado and Victor Manuel Pérez Colado. *Un conjunto de herramientas para Unity orientado al desarrollo de videojuegos de acción-aventura y estilo retro con gráficos isométricos 3D*. Facultad de Informática de la Universidad Complutense de Madrid, 2014. [Trabajo fin de Grado].

[14] Iván José Pérez Colado and Victor Manuel Pérez Colado. Isounity, 2014. URL `https://github.com/Victorma/IsoUnity`.

[15] Square Co. Ltd. Final fantasy tactics, 1997. [PlayStation].

[16] Square Enix. Final fantasy tactics advance, 2003. [Game Boy Advance].

[17] TOSE and Square Enix. The war of the lions, 2007. [PlayStation Portable, Android e iOS].

[18] Square Enix. Advance 2: Grimoire of the rift, 2008. [Nintendo DS].

[19] Square Enix. Square enix, 2003. URL `http://www.square-enix.com/`.

[20] Nintendo and Intelligent Systems. Fire emblem heroes, 2017. URL `https://fire-emblem-heroes.com/es/`. [Android e iOS].

[21] Intelligent Systems. Fire emblem: Rekka no ken, 2003. [Game Boy Advance].

[22] Stoic Studio. The banner saga, 2014. [Multiplataforma].

[23] Bob Charrette, Paul Hume y Tom Dowd. Shadowrun, 1989.

[24] Intelligent Systems. Advance wars, 2001. [Game Boy Advance].

[25] Blizzard Entertainment. Diablo, 1996. URL https://eu.diablo3.com/es/.

[26] Enterbrain. Rpg maker vx, 2012. URL https://www.rpgmakerweb.com/products/programs/rpg-maker-vx-ace.

[27] Vindicator. Rpg maker vx ace database guide 0.6, 2013. URL https://steamcommunity.com/sharedfiles/filedetails/?id=123520832.

[28] Juan Carlos García Romero. URL https://programavideojuegos.blogspot.com/2013/11/.

[29] Roger S. Pressman. *Ingeniería del software: Un enfoque práctico*. McGraw-Hill, 7th edition, 2010. ISBN 978-607-15-0314-5.

[30] Kent Beck, James Grenning, Robert C. Martin, Mike Beedle, Jim Highsmith, Steve Mellor, Arie van Bennekum, Andrew Hunt, Ken Schwaber, Alistair Cockburn, Ron Jeffries, Jeff Sutherland, Ward Cunningham, Jon Kern, Dave Thomas, Martin Fowler, and Brian Maric. *Manifesto for Agile Software Development*. Agile Alliance, 2010.

[31] Ian Sommerville. *Ingenieria del software*. Pearson Educación, 2017. ISBN 978-607-32-0603-7.

[32] Amit Patel. *Amit's A* Pages*. Red Blob Games. URL http://theory.stanford.edu/~amitp/GameProgramming/.

[33] Stuart J. Russell y Peter Norvig; traducción Juan Manuel Corchado Rodríguez. *Inteligencia artificial: un enfoque moderno*. Pearson Educación, 2nd edition, 2011. ISBN 978-84-205-4003-0.

Apéndice A

Title (in English)

Development of a Unity Toolkit for the Creation of Role-Playing Computer Games using Turn-Based Combat

Apéndice B

Introduction (in English)

If something characterizes the Video Game Industry in this last decade, it is its incessant production of titles [1],no longer developed only by large producers (known as AAA) but also by countless independent studies. Nowadays, video games are published simultaneously on all types of platforms (PC, consoles, smartphones ...) and most of them meet very demanding software, audiovisual and design quality requirements.

Video game developers, if they want to be successful in this complex sector, must be competitive and for this it is necessary that they get cheaper production costs, both in time and effort and complexity, of software development projects. For this, fortunately, there are tools that allow creating multiplatform videogames of quality, with relatively little effort and with relative speed.

It is true that today there are integrated development environments, such as Unity, and well-established tools for creating video games that belong to the most popular genres and are easily appealing to the public, such as 2D platform games or "*shooters.*"[2] in the first person in 3D. However, there are not so many tools nor so powerful in other genres that were not usually addressed by the most humble economic development studies. This is the case of tactical RPGs.

[1]Distribution of games released on Steam between 2004 and 2016, by release year - `https://www.statista.com/statistics/750099/steam-games-release-annual-distribution/`

[2]First Person Shooter Video Game

1. Tactical role games

The use of technology for leisure has been with us for more than 50 years. Since the physicist William Higinbotham at the Brookhaven National Laboratory, New York, invented a game similar to ping-pong on an oscilloscope, until Nolan Bushnell, May 24, 1972, the founder of the company Atari Inc, published "Pong", the game that would launch the video game industry, spent more than 10 years.

Then began the development of domestic platforms to play video games, the first video consoles such as the Atari 2600; with games such as Lunar Lander, Space Invaders, Pac-Man and Dragonstomper (first role-playing game in console, 1982).

The first computer role-playing video game can be considered "Dungeons and Dragons", in 1980, an adaptation of the famous board game.

Role-playing games are divided into several sub-genres, being the tactical role-playing game or TRPG (from English *Tactical Role-Playing Game*) one of the most affected in terms of lack of modern authoring tools oriented towards multiplatform production. This subgenre is characterized by structuring its movements and actions, such as those of combat, by turns and by dividing the stage into squares, often using an isometric or similar perspective.

The *turn* in this type of video game is key, since the way in which the actions are carried out makes this type of video games different from others. Each turn allows the player to think about how to move his team through the squares of the game board. Usually it is paused, since this type of games does not usually penalize the time destined to execute an action, and it is possible to be the desired time meditating the play that is wanted to realize.

Sagas of important videogames like Final Fantasy Tactics or Fire Emblem have been in the memory of the fans and they are still publishing new deliveries, and new games like Banner Saga, without trying to change the mechanics already defined by the classics, have triumphed, making clear that there is a market for this genre.

2. Development environments and authoring tools

As the technologies have been improving, the videogame sector has evolved with an increase in functionality, graphics and, ultimately, offering games of much greater complexity. This complexity is reflected in much more work to develop content and in turn, much higher development cost.

Spacewar !, considered by many the first computer game [2], was programmed in assembly code [3], while some of the Atari game consoles used the BASIC language to program their games. Languages that to make a video game is hard and expensive in time and results.

With these first video games on the table, the creators began to develop new forms of game. The popular Nintendo [4] redefined the standards of video games platforms with Super Mario Bros [5], Sierra Entertainment [6], for example, did something similar in his time with the bases of graphic adventures with King's Quest. Companies in the sector created and created their products, mostly with graphic engines and their own tools, to which the general public and independent developers do not have access.

Developing a videogame today to trade in the market involves a lot of risk and a large initial capital, and only companies with greater financial capacity can produce large projects, similar to what happens in the film industry. Nowadays, we are experiencing an irruption of independent developers, who create videogames without the economic pretensions of big productions, but who risk in the playable, even in the narrative or looking for new horizons in the video game as a medium. These companies can exist thanks, in part, to tools such as Unity [7], a development environment that offers a huge amount of tools to the independent developer, which has a free version and greatly facilitates the entire development process of a video game.

Today we have tools that facilitate the creation of video games according to the type of game we want to play, although we find genres where there are more development aids and others where there is less. In RPG Maker [8], a tool for the development of this type of videogames exists, although with results of graphic quality far below tools such as Unity or Unreal Engine [9].

The development, publication and physical distribution of a video game is a more expensive job than it seems, but access to video games from the customer's point of view could not be easier thanks to computer download platforms such as Steam [10], GOG [11] and similar, or the digital stores of video consoles, which allow you to buy and download all the titles you want in just a few clicks. This accessibility has led to the growth of independent studies, making the distribution of video games of a certain niche somewhat easier for small studies.

On the one hand the authoring tools that facilitate the production of video games and on the other the platforms that enable the distribution of them, has caused this boom for the so-called ïndie world.ºf the independent development of video games.

3. Purpose of the work

This work pretend contribute to the existence of useful and complete functional tools to develop tactical role videogames.

It is sought that developers without much experience or extensive knowledge in programming, can create this type of games in a simple and intuitive way. To do this, it is proposed to create a tool, with an important visual component and without just writing code, by means of which the creator can define various parameters (attributes, abilities, objects, weapons, and the like) in a consistent and accessible way.

The proposed tool will be based on what is probably the best known videogame development environment today, Unity, following a proof of concept that two students of the Faculty, Javier Druet and Luis Alfonso González [12], have already anticipated. integrating our work with another tool that has been created here and that increasingly becomes more relevant: IsoUnity [13].

Once the tool is done, we consider it important to create a demonstration, a test of a game made entirely with our tool to demonstrate the capabilities of it.

The type of functionalities that we look for are:

Characters with different classes, properties and attributes

Objects and weapons that modify the properties of the characters

Inventory of objects and weapons

System of turn-based combat by default (without preventing the developer from using another if he wanted to)

In addition, the project is planned with the intention of facilitating and allowing successive enlargements of future students or any other developer who loves free software who wishes to contribute.

4. Work Structure

This introductory chapter is followed by a review of the state of play in the field of turn-based role-playing and its authoring tools in Chapter 2. Chapter 3 details the objectives and specification of the tool to be developed, and Chapter 4 groups together the analysis, design and implementation of the software. In Chapter 5, we present the tests carried out on an example set, with the results obtained, and discuss the positive and negative aspects of the project. We conclude in Chapter 6, recapitulating what the tool can do now and what the plans are for improving it in the future.

Apéndice C

Conclusions (in English)

This tool is the result of a first contact, as it has been explained in different chapters of this report, of a tool made by a previous *Final Degree Project*. This project started with the intention of expanding and improving the existing tool, but the lack of documentation, as well as the link that this first version was to the IsoUnity tool, made the project focus, after hard work of research and reverse engineering , in rewriting all the code, structuring it again, and looking for a way to make **TRPG Maker** completely independent of any tool (except, obviously, Unity, since it is still an extension of it). To this end, the previous tool was taken as a reference to the necessary requirements, meetings were held with the director and co-director of the project to deepen the requirements, as well as constant contact between the members of the group to design and specify the tool.

After this first *crysis* about the future of the project, the following work consisted of defining and specifying what is necessary for any tactical role videogame, and after that research process began the implementation of the tool.

It has managed to implement the database, as well as its editor, in a visual and intuitive window, allowing to create complete characters, with *atributes*, *labels*, *objects*, *abilities* and *slot types*, which together define the character. It has also been possible to implement *computers*, differentiating whether they are playable or not, from the database editor itself.

One of the biggest problems that existed in the prototype TRPG Maker, was how closely coupled it was to the IsoUnity tool, in this project it has been possible to continue using IsoUnity, but in a totally independent way, allowing to use the tool that is desired as a complement to TRPG Maker. To this end, a connector has been implemented, explained in Chapter 4, so that it can be connected to any other tool or even to the developer's own, simply by implementing an interface.

It has also managed to implement a game manager, which allows turn-based combat between teams that are on the map, with the existence of artificial intelligence for teams that are not controlled by the player.

With all the above implemented, is open the possibility of improvements in the tool. The database and the editor work correctly, but you could give more depth to the data that can be stored in the database. At playable level, much more background can be added, since at the moment it consists of simple battles. There is also the possibility of being able to port the tool to others *engines* as Unreal Engine.

Apéndice D

Guía del usuario

En el siguiente documento se pondrá a disposición del usuario una guía de uso de la herramienta TRPG Maker.

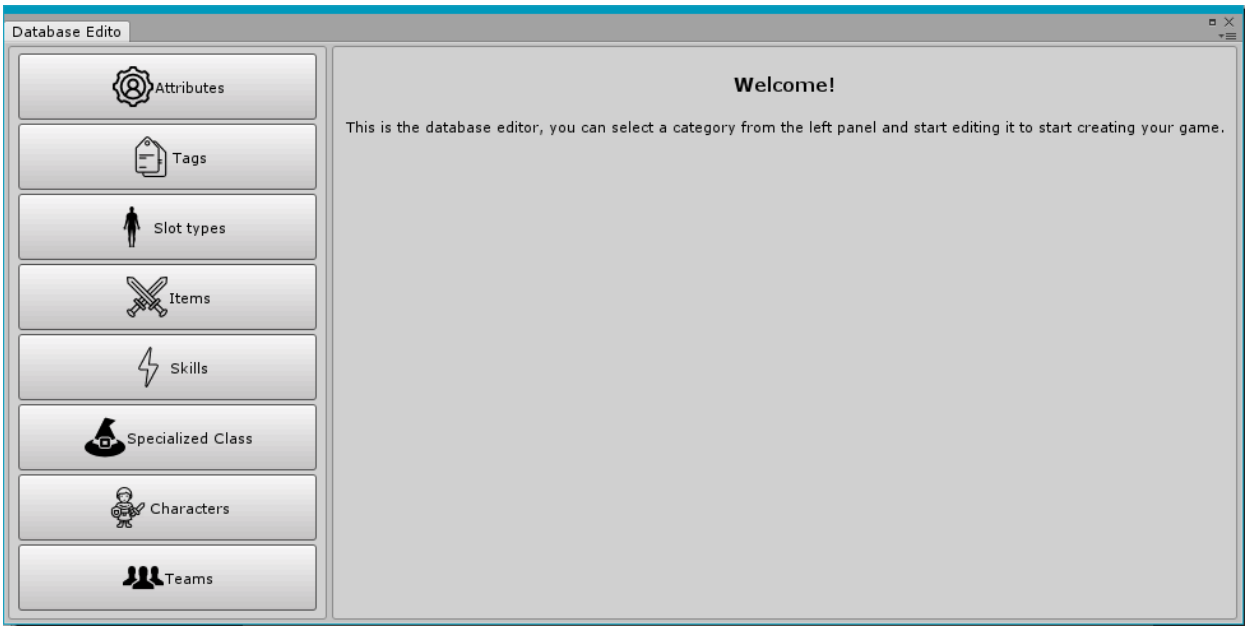


Tabla de contenido

1. Configuración inicial y puesta en marcha
2. Añadir información a la base de datos
3. Opciones de batalla y de conexión

1. Configuración inicial y puesta en marcha

Descargar el repositorio *https://github.com/Narratech/TRPGMaker* con la última versión del proyecto en la rama "master" y Unity, en su versión 2017.3.0f3. El repositorio contiene TRPG Maker y la herramienta IsoUnity.

Abrir la carpeta descomprimida del respositorio como proyecto de Unity (Existe una escena de prueba en la carpeta "Scenes" con el nombre test). Crearemos una escena con la herramienta IsoUnity como base de nuestro juego, para ello debemos añadir a la escena el componente "IsoUnity Game" haciendo click con el botón derecho en la parte de jerarquía de Unity. Del mismo modo debemos añadir a la escena un IsoUnity Map.

Al IsoUnity Game debemos, desde el inspector, seleccionar el mapa que vayamos a gestionar (el anteriormente añadido), añadir el script "IsoUnityOptions" y el script GamePlayManager. Para un correcto funcionamiento de la cámara del juego es recomendable marcar "Smooth" en Camera Mode en game.

Para crear el mapa del juego, pinchando en "Map" en la jerarquía y en "Edit" en el inspector, pudiendo crear casillas en la escena de juego a modo de mapa, cambiando la altura en "Grid Height".

Usando IsoUnity, trataremos los personajes como "Decorations", para añadir un personaje al terreno de juego debemos pinchar en la jerarquía sobre "map" y después en el inspector en "decoration" y añadir desde ahí el personaje que queramos en el juego.

Debemos añadir al "decoration" del personaje los scritps "mover", "entity", "character script" y "talker".

- mover", "entity" y "talker" son scripts propios de IsoUnity
- "character script" pertenece a TRPGMaker. En este script debemos añadir el personaje que queremos de la base de datos de TRPGMaker y el equipo al que vaya a pertenecer.

De esta manera podremos crear un mapa con diferentes personajes. Para añadir información sobre los atributos, equipo, y demás elementos, debemos hacerlo mediante el "database editor" de TRPGMaker, que se explica a continuación.

2. Añadir información a la base de datos

TRPGMaker cuenta con un editor de su base de datos, para acceder a él debemos pinchar sobre la barra de herramientas en TRPGMaker y en "Database editor". Aparecerá una ventana nueva, donde en la parte izquierda aparecerán secciones de la base de datos que se pueden agregar, eliminar o modificar, que, haciendo click sobre ello aparecerá a la derecha información al respecto de esa sección de la base de datos.

Las secciones de la base de datos de TRPGMaker son las siguientes:

1. **Attributes**

   Attributes, o atributos. Aquí podremos editar la lista de atributos, añadiendo nuevos atributos, pudiendo editar y eliminar los atributos existentes.

   Los atributos están formados por un Nombre, un ID, una descripción, un valor máximo y mínimo y un valor. Además, tendremos que seleccionar si es "core" o no. Si se selecciona que es core, este atributo aparecerá en todos los personajes por defecto.

2. **Tags**

   Tags, o etiquetas. En esta sección se pueden añadir o eliminar etiquetas, también se podrá editar el nombre de los mismos. Las etiquetas sirven para definir objetos y valores.

3. **Slot types**

   Slot types, o tipo de hueco. En esta sección se pueden añadir o eliminar slots types, también se puede editar el nombre de los mismos.

   Los Slot Types sirven para definir los "huecos" de un personaje y donde pueden ir equipados los objetos.

4. **Items**

   Items, u objetos. En esta sección se pueden añadir, eliminar y editar objetos. Un objeto consiste en:

   a) Nombre

   b) Descripción

   c) Icono

   d) Tags

   e) Numero de combinaciones del objeto. En este apartado se pueden definir los "Slot types" necesarios para usar el ítem, pudiendo hacer combinaciones de varios Slot (dos manos) o pudiéndose añadir a solo un slots con varias posibilidades (mano derecha o mano izquierda)

   f) Fórmula

5. **Skills**

   Skills, o habilidades. En esta sección se pueden añadir, eliminar y editar habilidades. Una habilidad consiste en nombre, descripción, el tipo de habilidad (en área, proyectil, un solo objetivo, área desde el objetivo) y el daño que realiza dicha habilidad.

6. **Specialized Class**

   Specialized class, o clase especializada. En esta sección se puede añadir, eliminar y editar clases especializadas.

   Una clase especializada consiste en un conjunto de Attributes, un conjunto de Tags, un conjunto de Slot Types, un conjunto de Skills y una fórmula.

   Todos los conjuntos que se mencionan en "Specialized Class" son los existentes en la Base de datos de TRPGMaker de las secciones anteriores.

7. **Characters**

   Characters, o personajes. En esta sección se puede añadir, eliminar y editar personajes.

   Un personaje consiste en un conjunto de Attributes, un conjunto de Tags, un conjunto de Slot Types, un conjunto de Skills y un conjunto de Specilized Classes.

   Todos los conjuntos que se mencionan en "Characters" son los existentes en la Base de datos de TRPGMaker de las secciones anteriores.

8. **Teams**

   Teams, o equipos. En esta sección se pueden editar, crear o eliminar equipos.

   Un equipo consiste en un conjunto de personajes, un ID y un nombre del equipo. También tendremos que seleccionar si es "Playable", para poder usar ese equipo en el juego o de lo contrario, es un equipo controlado por la IA. (Por defecto, "Playable" esta deseleccionado)

3. Opciones de batalla y de conexión

TRPGMaker está optimizado en esta versión para ser usado junto a la herramienta IsoUnity, pero está abierto a poder ser usado con otras herramientas. Para ello cuenta con dos opciones que facilitan esta tarea para el desarrollador:

1. **Battle options.** Pinchando sobre TRPGMaker ->Battle options en Unity, aparecerá una ventana donde podemos seleccionar los atributos y valores que queramos para nuestro juego. Siendo:

   a) *Turn type*. Donde elegiremos si queremos que el tipo de turno sea aleatorio o por el valor de un atributo de los personajes.

   b) *Health attribute name*. Para seleccionar el atributo que sea la "vida" de los personajes del juego.

   c) *Damage attribute name*. Para seleccionar el atributo que contenga el valor del "daño" realizado por los personajes en el juego.

d) *Movement range attribute name*. Para seleccionar el atributo que defina el rango de movimiento de los personajes.

e) *Movement height attribute name*. Para seleccionar el atributo que defina el rango de altura que los personajes pueden saltar.

f) *Attack range attribute name*. Para seleccionar el atributo que defina el rango de ataque de los personajes.

g) *Attack height attribute name*. Para seleccionar el atributo que defina el rango de altura del ataque de los personajes.

Véase que estas opciones deben ser definidas por el usuario en la base de datos de TRPGMaker. Siendo atributos básicos para un juego de rol táctico.

2. **Connector options.** Pinchando sobre TRPGMaker ->Connector options en Unity, aparecerá una ventana donde podremos seleccionar el conector que queramos usar con TRPGMaker. Actualmente solo esta definida la conexión con la herramienta IsoUnity, con "IsoUnityConnector", dejando abierta la posibilidad de añadir un mayor número de herramientas en el futuro. En "connector options" seleccionando IsoUnityConnector podemos elegir:

   a) *Move Cell*. Para seleccionar la imagen en la que se quiera ver la celda en la que se quiere realizar un movimiento de los personajes.

   b) *Attack Cell*. Para seleccionar la imagen en la que se quiera ver la celda en la que se quiere realizar un ataque de los personajes.

   c) *Arrow Decoration*. Para seleccionar la forma en la que se quiera ver la flecha de selección en la celda correspondiente del mapa.

Apéndice E

Pruebas guiadas con usuarios reales

Fecha: 3 de mayo de 2018, de 16:00 a 19:00 horas

Lugar: Laboratorio 3 de la Facultad de Informática de la Universidad Complutense de Madrid (Madrid, España)

TRPG Maker es una herramienta para Unity desarrollada para ayudar a los desarrolladores en la creación de videojuegos de rol tácticos. Para ello, usamos la herramienta IsoUnity para la creación de escenarios y gestión de juego.

Esta prueba consiste en crear uno o varios personajes para derrotar a un malvado y peligroso enemigo que se encuentra en el mapa del juego. Para ello es necesario:

1. **Crear uno o varios personajes**.

   Este personaje debe ser añadido a la base de datos de TRPGMaker. Para acceder a ella hay que pinchar sobre la barra de herramientas en "TRPGMaker" ->Database editor. En el editor de la base de datos se pueden añadir distintos atributos, tags, ítems, etc. (Skills, no funciona, así que añadir o quitar datos de esta sección no cambiará nada en el juego). Una vez añadido todo lo deseado, en "Characters" le daremos formaremos al personaje como deseemos, y posteriormente, lo añadiremos al equipo "playable" para poder derrotar al enemigo.

   Una vez añadido a la base de datos TRPGMaker debemos añadirlo a la escena de Unity. Para ello se tendrá que usarse la herramienta IsoUnity para añadir un personaje visual. Pinchando sobre "map" en la jerarquía de la escena, y posteriormente añadir una "decoration" con el personaje que se desee crear.

2. **Añadir/editar la base de datos**.

   Una vez creado el personaje, este puede ser editado. Si se quiere cambiar un ítem, o añadir un arma al personaje, este es el momento. Simplemente vuelve a entrar en el editor de la base de datos, y crea, añade o edita al personaje para la batalla.

3. **Luchar contra el enemigo**.

   Preparad todo para la batalla. El objeto del personaje en el inspector debe contener los scripts: "entity", "mover" (de IsoUnity) y "Character Script". En character editor debemos seleccionar al personaje que hayamos creado en la base de datos, junto a su equipo. Debemos asegurarnos de que nuestro personaje está añadido a un "team" que tenga seleccionado "playable" (y que el enemigo pertenezca a uno que no lo tenga, ya que de lo contrario no será enemigo). ¡OJO! Skills no está en funcionamiento

   Cerciorándonos de que todo esta correcto, pulsamos PLAY y que comience la batalla, pudiendo movernos, atacar o pasar turno.

   ¡Esperamos que salgáis victoriosos de la batalla!

NOTA: No está implementado el final del combate, una vez el enemigo tenga "0" vida el juego continuará como si aun estuviera en combate, aunque este personaje ya no realizará ninguna acción de ataque/movimiento.

Sería de gran utilidad el realizar esta encuesta:

https://goo.gl/forms/LWpGHZENvfat6s5W2

Muchas gracias

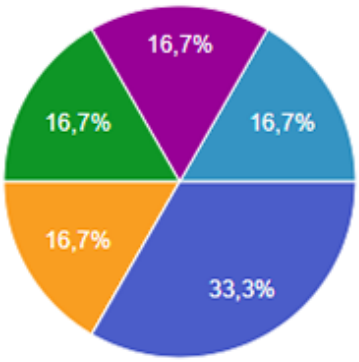
Apéndice F

Resultados de las pruebas guiadas con usuarios reales

A continuación se muestran los resultados de la encuesta realizada durante las pruebas guiadas con usuarios reales y su respuestas a la misma.

Usabilidad

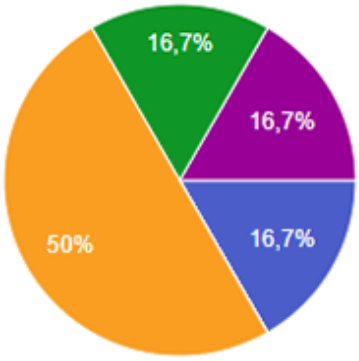
6 respuestas



- Necesita ser más intuitivo
- No tengo ni idea de por donde empezar
- Todo bien
- Es intuitivo pero si no has usado nunca unity te ves un poco perdido
- Bastante intuitivo visualmente (iconos), pero puede venirle bien al...
- Es intuitivo, pero algún menu tiene demasiados datos aglomerados

Diseño

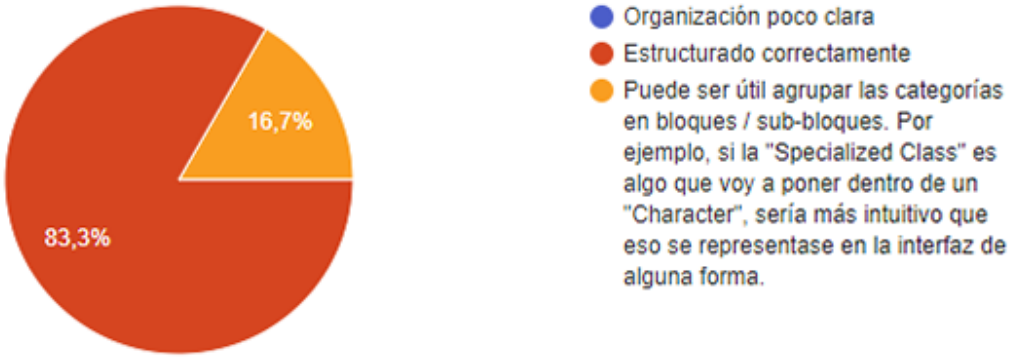
6 respuestas



- Iconos poco representativos
- Diseño del interfaz in-game poco desarrollado (Ten en cuenta que es una prueba/test)
- Diseño claro e iconos representativos
- Los iconos están bien escogidos, pero todos siguen el paradigma del Rol de fantasía. Puede ser interesa...
- En general bien, pero hay iconos que deberían ser más grandes

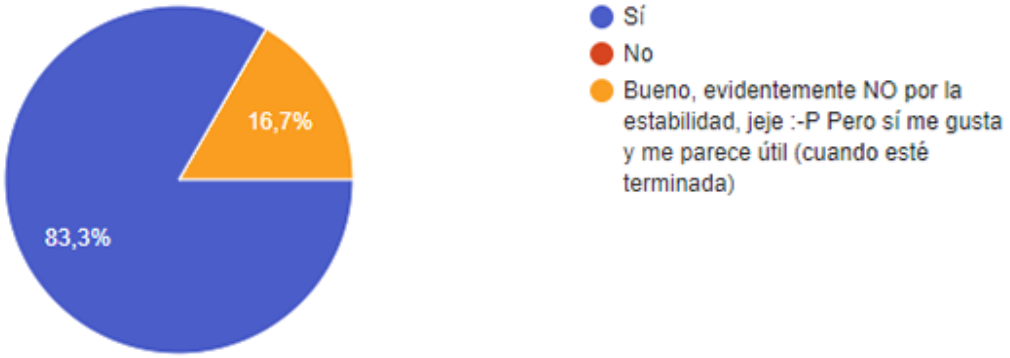
Estructura de la herramienta

6 respuestas



¿Usarías esta herramienta?

6 respuestas



Opinión general

6 respuestas

Ojo a las erratillas: New Attrbiute 007. El - para quitar atributos lo veo poco intuitivo Me deja loco lo de las combinations... no es fácil saber qué es Deshabilitar numerito de las combinations (bueno, el campo es que no se actualiza) Creé un personaje Ironman, le metí clase especializada Butanero... y petó Cuando creas un new character me mantenía el nombre del anterior (bueno, por sólo unos segundos...) Las líneas más claras y más oscuras al seleccionar Edit de un personaje o un team me resultan confusas (parece que algo está seleccionado!) De Thor me ha quitado la erre al final (??? la última letra no la coge) Creemos que si no tiene team un personaje, explota todo (pero lo que pasa en realidad es que NO ME GUARDA EL TEAM!) Ojo que si cambias el nombre de un atributo de la BBDD que esté en battleoptions, PARECE que te lo cambia, pero luego resulta que no lo tienes y creemos que casca el juego por eso Volví a cambiar Health de Salud y luego a Health... y se acabaron duplicando Cuando él me mata me quita vida todo el rato (infinitos turnos), y no se entiende mucho ese número X/Y que sale sobre el personaje (me decía 200/100, que tengo 200 de vida y hay 100 de total)
Me ha gustado, este tipo de herramientas me parece bastante útil para facilitar a la gente que quiere empezar, sin la complicación de que tenga que escribir código o al menos reducirlo, de tal forma que con unos minimos conocimientos de Unity pueda empezar a crear algo.
Creo que es una idea con potencial real dentro de la industria del videojuego: ya sea a nivel académico, amateur o incluso profesional. Hoy en día cada vez hay más demanda de herramientas que faciliten la vida a aquellas personas que no tengan un background técnico y quieran hacer un videojuego. Haberla enfocado hacia un público específico, el de los TRPGs, me parece un acierto y yo seguiría incluyendo features en esa línea (siempre con la máxima abstracción posible, pero teniendo ese género en la cabeza). Si no está incluido, creo que aparte del conector propio (que me parece una gran idea también), puede ser muy interesante y útil que el desarrollador pueda exportar sus creaciones en formatos de etiquetas tipo XML, Excel, HTML, etc. Ya que si hablamos de que es una herramienta muy accesible que cualquiera puede usar, también debe de dar las máximas facilidades posibles para exportar los datos en diferentes formatos para adaptarse al máximo número de usuarios posibles. Como locura final y pensando en un largo plazo, la herramienta ganaría una flexibilidad casi infinita si pudiera generar archivos con clases o métodos en C# correspondiente a los que el usuario hubiera creado. De este modo un programador podría editar esos archivos manualmente y añadir/modificar lo que quisiera pero sobre una base ya creada, lo que le ahorraría mucho trabajo.

Opinión general

6 respuestas

Es una herramienta muy intuitiva, solo le falta que los menus sean más “agradables a la vista”
Me parece una gran idea. Yo antes de entrar a la carrera siempre buscaba herramientas para hacer juegos. Igual sería más intuitivo que pudieras arrastrar cosas a las ventanas como ofrecen otros editores, pero es cierto que esta herramienta es para hacer juegos de rol y eso implica una mayor dificultad que da lugar a que la interfaz sea complicada. Pero que sea complicada es porque la herramienta es bastante completa y ofrece muchas cosas. Muy top
Está bastante bien, y sólo hay pegas en cuestiones ya planteadas por vosotros (la coherencia de las restricciones en las armas) o en cuestiones a ser desarrolladas (coherencia en los tags de las armas que puedes asignar según la clase de cada personaje)