

---

# **Creación de Videojuegos de Rol Táctico mediante Herramientas de Desarrollo para el Usuario Final**

*Creation of Tactical Role-Playing Games using Tools  
for End-User Development*

---



*Trabajo de Fin de Máster realizado por*

**José Javier Cortés Tejada**

*Dirigido por*

**Prof. Dr. Federico Peinado Gil**

Máster en Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid

*Curso académico*

**2019 - 2020**

*Convocatoria*

**Septiembre**



# Resumen

La industria del videojuego ha crecido exponencialmente en los últimos años. Como consecuencia de esta evolución, los entornos de desarrollo se han convertido en herramientas cada vez más complejas centradas en las características comunes de los videojuegos comerciales modernos.

Este fenómeno ha propiciado la aparición de herramientas que no requieren conocimientos técnicos para que los usuarios las usen, haciendo el desarrollo de videojuegos accesible a todo el mundo. Debido al alto interés en este tipo de software hemos decidido desarrollar *TRPG Maker*, una herramienta intuitiva y autocontenido enfocada en el desarrollo de un género concreto de videojuegos para el usuario final.

El diseño de esta aplicación se centra en facilitar la creación de videojuegos de rol táctico a aquellos usuarios sin conocimientos de programación o sin interés en la parte técnica del desarrollo de videojuegos.

Junto con el desarrollo de la herramienta hemos llevado a cabo un proceso de diseño y desarrollo basándonos en los principios de desarrollo para el usuario final y hemos realizado una primera validación experimental con usuarios reales para determinar el grado de utilidad y comodidad del prototipo, así como para detectar fallos y descubrir posibles mejoras para esta herramienta. Por último, la realimentación obtenida ha sido usado para crear una versión del producto más completa, comparable en rasgos generales con herramientas similares que encontramos en el mercado, se ha publicado y puesto a disposición de la comunidad para su uso gratuito.

## Palabras clave

Videojuegos, Juegos de Rol, Herramientas, Usabilidad, Diseño de Videojuegos



# Abstract

Video game industry has grown exponentially in the last years. As a consequence of this evolution, development environments have become more complex toolkits focused on the many features that are common in modern commercial video games.

This phenomenon has led to the emergence of tools that do not require technical knowledge for users to use them, making video game development accessible to everyone. Due to the high interest in this type of software we have decided to develop *TRPG Maker*, an intuitive and self-contained tool focused on the development of a specific genre of video games for the end user.

The design of this application focuses on facilitating the creation of tactical role games to those users without programming knowledge or interest in the technical part of video game development.

Along with the development of the tool we have carried out a design and development process based on the principles of end-user development and we have made a first experimental validation with real users to determine the degree of usefulness and comfort of the prototype, as well as to detect failures and discover possible improvements for this tool. Finally, the feedback obtained has been used to create a more complete version of the product, comparable in general features with similar tools that we find in the market, which has been published and made available to the community for free use.

## Keywords

Video Games, Role-Playing Games, Tools, Usability, Video Game Design



# Índice

<b>Resumen</b>	<b>III</b>
<b>Abstract</b>	<b>v</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Objetivos . . . . .	3
1.3. Alcance del proyecto . . . . .	3
1.4. Estructura del documento . . . . .	4
<b>2. Estado de la cuestión</b>	<b>7</b>
2.1. End user programming . . . . .	7
2.1.1. Visual programming . . . . .	8
2.1.2. Programming by demonstration . . . . .	9
2.1.3. Programming with text . . . . .	10
2.1.4. Programming by specification . . . . .	11
2.2. End user software engineering . . . . .	12
2.2.1. Requisitos . . . . .	13
2.2.2. Diseño y especificación . . . . .	14
2.2.3. Reutilización . . . . .	15
2.2.4. <i>Testing</i> y verificación . . . . .	17
2.2.5. <i>Debugging</i> . . . . .	18
2.3. Herramientas profesionales . . . . .	19
2.3.1. RPG Maker . . . . .	19
2.3.1.1. Editor de mapas . . . . .	19
2.3.1.2. Editor de eventos . . . . .	20
2.3.1.3. Editor de bases de datos . . . . .	21
2.3.2. Construct . . . . .	22
2.3.3. GameMaker: Studio . . . . .	23
2.4. Herramientas educativas . . . . .	24
2.4.1. MIT App Inventor . . . . .	24

2.4.1.1. Diseño de pantallas . . . . .	25
2.4.1.2. Editor de bloques . . . . .	26
2.4.2. Scratch . . . . .	27
<b>3. Herramientas de desarrollo</b>	<b>31</b>
3.1. Unreal Engine . . . . .	31
3.1.1. Blueprints . . . . .	31
3.1.2. C++ . . . . .	32
3.1.3. Clases básicas en un proyecto de Unreal Engine . . . . .	34
3.2. Otras herramientas de desarrollo . . . . .	35
3.2.1. Mixamo . . . . .	35
3.2.2. Aseprite . . . . .	35
<b>4. Metodología y gestión del proyecto</b>	<b>37</b>
4.1. Primer hito . . . . .	38
4.1.1. Tareas del editor de escenarios 3D . . . . .	39
4.1.2. Tareas del editor de bases de datos . . . . .	42
4.1.3. Tareas del editor de conversaciones . . . . .	44
4.2. Segundo hito . . . . .	45
4.2.1. Tareas del editor de escenarios 2D . . . . .	45
4.2.2. Tareas del editor de mapas . . . . .	47
4.2.3. Tareas del gestor de colas . . . . .	48
4.3. Planificación temporal . . . . .	49
<b>5. Desarrollo del sistema</b>	<b>51</b>
5.1. Arquitectura del sistema . . . . .	51
5.1.1. Editor de niveles . . . . .	53
5.1.1.1. Clase GeometryComp . . . . .	57
5.1.1.2. Clase MovementComp . . . . .	58
5.1.1.3. Clase UndoRedoComp . . . . .	59
5.1.1.4. Clase DataLoaderComp . . . . .	60
5.1.1.5. Clase ObjectAllocatorComp . . . . .	60
5.1.2. Acciones y gestión del editor de niveles . . . . .	61
5.1.3. Editor de bases de datos . . . . .	63
5.1.4. Editor de mapas . . . . .	64
5.1.5. Editor de conversaciones . . . . .	65
5.1.6. Gestor de colas de eventos . . . . .	66
<b>6. Pruebas y resultados</b>	<b>69</b>
6.1. Pruebas con usuarios reales . . . . .	69
6.2. Comparativa con otras herramientas . . . . .	72
6.3. Artículo de investigación . . . . .	73

<b>7. Conclusiones</b>	<b>75</b>
<b>Bibliografía</b>	<b>77</b>
<b>A. Resultados de la evaluación</b>	<b>81</b>
<b>B. Introduction</b>	<b>89</b>
B.1. Motivation . . . . .	89
B.2. Objectives . . . . .	90
B.3. Project scope . . . . .	91
B.4. Document organization . . . . .	91
<b>C. Conclusions</b>	<b>93</b>



# Índice de figuras

2.1.	Captura de pantalla del editor de LabVIEW. . . . .	9
2.2.	Captura de pantalla de la herramienta CoScripter. . . . .	10
2.3.	Ejemplo de un error de tipos en GNOME. . . . .	11
2.4.	Ejemplo de funcionamiento de la herramienta Metafor. . . . .	12
2.5.	Captura de pantalla de la herramienta Damask. . . . .	16
2.6.	Ejemplo de la metodología WYSIWYT en la validación de las celdas de una hoja de cálculo. . . . .	17
2.7.	Captura de pantalla de la herramienta de <i>debugging</i> prototipada para Alice. . . . .	19
2.8.	Editor de mapas de RPGMaker. . . . .	20
2.9.	Editor de eventos de RPGMaker. . . . .	21
2.10.	Editor de bases de datos de RPG Maker. . . . .	21
2.11.	Hojas de eventos de Construct. . . . .	22
2.12.	Ejemplos de eventos disponibles en Construct. . . . .	22
2.13.	Ejemplo del sistema <i>point and click</i> de GameMaker: Studio. .	23
2.14.	Opciones del editor de GameMaker: Studio. . . . .	23
2.15.	Ejemplo de conversión entre código GML y código visual. .	24
2.16.	Proceso de creación de una app con MIT App Inventor. . .	25
2.17.	Interfaz de la opción diseño de pantallas de MIT App Inventor.	25
2.18.	Interfaz de la opción editor de bloques de MIT App Inventor.	26
2.19.	Interfaz gráfica de la herramienta Scracth. . . . .	27
2.20.	Fragmento del clásico juego <i>pong</i> codificado con Scracth. .	28
2.21.	Fragmento del clásico juego <i>pong</i> codificado con Scracth. .	29
3.1.	Ejemplo de programación visual con <i>Blueprints</i> . . . . .	32
3.2.	Jerarquía de clases básica de Unreal Engine 4. . . . .	32
3.3.	Código que muestra la instanciación de un Actor en un nivel.	33
3.4.	Ejemplo de especificadores en funciones y variables. . . . .	34
5.1.	Módulos que integran la herramienta <i>TRPG Maker</i> . . . . .	52
5.2.	Diagrama de clases del módulo editor de niveles de <i>TRPG Maker</i> . . . . .	54

5.3.	Diagrama de clases que corresponde a los objetos del escenario, en este caso bloques, personajes y objetos. . . . .	55
5.4.	Eventos para la gestión de <i>inputs</i> del ratón sobre un objeto. .	56
5.5.	VARIABLES que mantienen la lógica de un escenario. . . . .	57
5.6.	Ejemplo de trazas en Unreal Engine 4. El color rojo indica que el trazo no ha colisionado, mientras que el verde indica que ha habido colisión con un objeto con un perfil de colisión bloqueante. . . . .	58
5.7.	Diagrama de clases para el patrón <i>Command</i> implementado.	59
5.8.	Cabecera del evento <i>TickComponent</i> implementado por el componente <i>ObjectAllocatorComp</i> . . . . .	60
5.9.	Ejemplo de escenario con vista isométrica y personajes 2D de <i>TRPG Maker</i> . . . . .	61
5.10.	Máquina de estados que gestiona las batallas del editor de niveles. . . . .	62
5.11.	Ejemplo de habilidad en <i>TRPG Maker</i> es un escenario con vista isométrica. Las casillas azules representan el área de la habilidad (casillas donde puede ser usada) mientras que las casillas verdes representan el rango (casillas donde afectará la habilidad). . . . .	64
5.12.	Mapa del mundo del videojuego <i>Final Fantasy Tactics Advance</i> .	65
5.13.	Editor de conversaciones de <i>TRPG Maker</i> . Los mensajes de los diálogos corresponden a las etiquetas con imágenes, mientras que las demás corresponden a las bifurcaciones. . . . .	66
5.14.	Diagrama de clases de los eventos. . . . .	67
6.1.	Resumen de las respuestas obtenidas en relación a la pregunta “ <i>Me gusta la experiencia de uso de la aplicación y su estética</i> ”. .	70
6.2.	Resumen de las respuestas obtenidas en relación a la pregunta “ <i>Me gusta el manejo de la aplicación y sus controles (movimiento, botones, cámara, etc.)</i> ”. . . . .	71
6.3.	Resumen de las respuestas obtenidas en relación a la pregunta “ <i>Recomendaría el uso de esta aplicación con fines educativos</i> ”.	72
A.1.	Resumen de las respuestas obtenidas de la primera pregunta del cuestionario. . . . .	81
A.2.	Resumen de las respuestas obtenidas de la segunda pregunta del cuestionario. . . . .	81
A.3.	Resumen de las respuestas obtenidas de la tercera pregunta del cuestionario. . . . .	82
A.4.	Respuestas obtenidas de la quinta pregunta del cuestionario. .	84
A.5.	Resumen de las respuestas obtenidas de la séptima pregunta del cuestionario. . . . .	85

A.6. Resumen de las respuestas obtenidas de la novena pregunta del cuestionario. . . . .	86
A.7. Resumen de las respuestas obtenidas de la décima pregunta del cuestionario. . . . .	86



# Índice de tablas

4.1.	Desglose de tareas para el editor de escenarios 3D. . . . .	41
4.2.	Desglose de tareas para el editor de bases de datos. . . . .	43
4.3.	Desglose de tareas para el editor de conversaciones. . . . .	44
4.4.	Desglose de tareas para el editor de escenarios 2D. . . . .	46
4.5.	Desglose de tareas para el editor de mapas. . . . .	47
4.6.	Desglose de tareas para el gestor de colas. . . . .	49
6.1.	Comparativa entre TRPG Maker y otras herramientas en el mercado. . . . .	72



# Capítulo 1

## Introducción

**Resumen:** En este capítulo se exponen la motivación y el alcance del proyecto, además de los objetivos del mismo.

### 1.1. Motivación

La revolución de las nuevas tecnologías está teniendo un fuerte impacto tanto a nivel social como económico, favoreciendo la aparición a nuevos modelos de negocio así como la aparición de nuevas industrias, como es el caso de la industria del videojuego.

Este sector cuenta con una gran proyección de crecimiento tanto a nivel nacional como internacional, siendo así que en los últimos veinte años se ha consolidado como un contribuyente relevante a la economía mundial del entretenimiento. En comparación con otras industrias del entretenimiento más establecidas, como son las industrias del cine y la música, ha sufrido un crecimiento exponencial tanto en número de usuarios como en cifras a nivel comercial.

La industria del videojuego a día de hoy es un referente en la economía digital global ([DEV, 2016](#)), no solo por los aspectos ya comentados, sino por la cantidad de nuevos profesionales, empresas construidas y software existente para la creación de este tipo de productos. Como consecuencia, este software ha ido evolucionando progresivamente con el objetivo de suplir las demandas actuales a la hora de desarrollar videojuegos.

Esta evolución ha hecho que herramientas que en sus inicios eran simples se hayan convertido, con el paso del tiempo y tras muchas actualizaciones, en grandes motores gráficos con los que es posible crear todo tipo de videojuegos, los cuales pueden ser usados en múltiples dispositivos. Esto ha

provocado que cualquier herramienta de desarrollo actual como Unreal Engine 4<sup>1</sup> o Unity<sup>2</sup> requiera un grado de especialización muy alto, con lo que el desarrollo del videojuego se ha ido haciendo menos accesible al público general con el paso del tiempo.

A pesar de ello estas herramientas han apostado durante los últimos años por otras alternativas al desarrollo que no requieran conocimiento técnicos tan elevados. Esta otra alternativa son los lenguajes de programación visual, también conocidos como *blueprints*. Los *blueprints* son un sistema de scripting visual que emplea una interfaz basada en nodos para crear elementos del juego. Además, al igual que otros sistemas de scripting visual como Scracth ([Resnick et al. 2009](#)) también permite definición de clases u objetos en el propio motor.

Se trata de un sistema muy flexible y potente, ya que proporciona una capa de abstracción con la que no es necesario tener elevados conocimientos técnicos para poder ser usado. No obstante, este sistema de scripting visual está dirigido tanto a diseñadores como programadores, de forma que permita una mejor comunicación entre miembros de un mismo equipo de desarrollo pero de diferentes perfiles. Esta es una opción que acerca el desarrollo de videojuegos a un público más amplio, sin embargo no es una opción pensada para usuarios de a pie.

También existen otras herramientas con sistemas de scripting visual a parte de las ya mencionadas, como *Bolt*<sup>3</sup>, *PlayMaker*<sup>4</sup> o *Adventure Creator*<sup>5</sup>. Estas herramientas son *plugins* que se usan en conjunto a un motor gráfico, en este caso Unity, simplificando en gran medida el desarrollo. Sin embargo sigue siendo necesario usar un motor complejo para desarrollar, luego es necesario disponer de elevados conocimientos técnicos.

Por otro lado tenemos una serie de herramientas enfocadas hacia un público más general como *RPG Maker*, donde no son necesarios conocimientos técnicos a la hora de desarrollar. Estas herramientas utilizan un sistema *point and click* con el que solo es necesario seleccionar los objetos que se quieran añadir al juego y se sueltan en el editor, sin necesidad de programar o hacer uso de sistemas de scripting visual.

---

<sup>1</sup>[Unreal Engine 4](https://www.unrealengine.com/en-US/) - <https://www.unrealengine.com/en-US/>

<sup>2</sup>[Unity](https://unity.com/es) - <https://unity.com/es>

<sup>3</sup>[Bolt](https://assetstore.unity.com/packages/tools/visual-scripting/bolt-163802) - <https://assetstore.unity.com/packages/tools/visual-scripting/bolt-163802>

<sup>4</sup>[PlayMaker](https://assetstore.unity.com/packages/tools/visual-scripting/playmaker-368) - <https://assetstore.unity.com/packages/tools/visual-scripting/playmaker-368>

<sup>5</sup>[Adventure Creator](https://adventurecreator.org/) - <https://adventurecreator.org/>

Este tipo de herramientas han ido ganando fuerza en los últimos años y por ello hemos visto oportuna la creación de *TRPG Maker*, una aplicación de escritorio para el diseño y desarrollo de videojuegos de rol táctico desarrollada con C++ y el motor Unreal Engine 4.

## 1.2. Objetivos

El propósito principal de este proyecto es la creación de una aplicación de escritorio centrada en el diseño y desarrollo de videojuegos de rol táctico, enfocada a un público general sin conocimientos técnicos relacionados con esta disciplina. Para ello, llevaremos a cabo una serie de actividades con la finalidad de alcanzar estos objetivos:

- **Objetivo 1:** se llevará a cabo una revisión en profundidad del tema de las herramientas de creación de videojuegos pensadas para el usuario final.
- **Objetivo 2:** se escogerá un género concreto para implementar un prototipo de uso sencillo que no requiera de manuales y que pueda ser probado por cualquier usuario. Este primer prototipo será evaluado con usuarios reales, teniendo en cuenta la información cualitativa en relación a la simplicidad de uso del mismo en el resto del desarrollo del proyecto.
- **Objetivo 3:** se diseñará un sistema más complejo en función de la realimentación recibida por los usuarios, añadiendo distintas formas de programación de usuario final acordes a la herramienta y el tipo de juegos que se pretenden crear con ella.
- **Objetivo 4:** se evaluará una primera versión funcional del sistema mediante comparación con otras herramientas comerciales disponibles en el mercado.

## 1.3. Alcance del proyecto

El alcance de este proyecto va a ceñirse a la creación de una versión *alpha*. Una versión *alpha* dentro del ciclo de vida del software corresponde a un producto funcional que es evaluado por usuarios pero que aún es inestable, de forma que puede contener errores o no llegar a implementar todos los requisitos. Esta limitación viene dada por el hecho de que un proyecto como este necesita de varios perfiles técnicos (como animadores o arquitectos de software) para ser un producto real similar a los ya existentes en el mercado. Es por ello que es necesario limitar el alcance del sistema al tratarse de

un proyecto de fin de máster que va a ser implementado por una sola persona.

Es resultado de este proyecto será una versión *alpha* de la herramienta *TRPG Maker*, la cual será publicada en la plataforma [itchi.io](#) y puesta a disposición de la comunidad para su uso gratuito.

## 1.4. Estructura del documento

Con el fin de orientar al lector de cara a la lectura del documento, en este apartado se describe la estructura del documento. La presente memoria se divide en los siguiente capítulos:

- **Capítulo 1. Introducción:** en este capítulo se expone la situación actual dentro del desarrollo de videojuegos y se explica la motivación del proyecto. Además, se presentan los objetivos a lograr así como el alcance del proyecto.
- **Capítulo 2. Estado de la cuestión:** este capítulo se centra en el desarrollo para el usuario final. Además se realiza un análisis de varias herramientas con el objetivo de tener en cuenta las características comunes entre sí para el desarrollo de *TRPG Maker*.
- **Capítulo 3. Herramientas de desarrollo:** en este capítulo se detallan las tecnologías empleadas en el desarrollo del proyecto.
- **Capítulo 4. Metodología y gestión del proyecto:** en este capítulo se detalla la metodología y la planificación que se han aplicado para este proyecto.
- **Capítulo 5. Desarrollo del sistema:** en este capítulo se expone la arquitectura y el diseño del sistema, desarrollando las fases relativas al desarrollo del mismo, además de las especificaciones técnicas del mismo.
- **Capítulo 6. Pruebas y resultados:** este capítulo documenta las pruebas con usuarios llevadas a cabo así como los resultados obtenidos de los mismos.
- **Capítulo 7. Conclusiones:** en este capítulo se plasman las conclusiones obtenidas tras la realización del proyecto.

Por último, junto a esta memoria se aportarán una serie de apéndices:

- **Apéndice A: Resultados de la evaluación:** resultados en bruto de la sesión de experimentación llevada a cabo con usuarios.

- **Apéndice B: Introduction:** este anexo contiene el capítulo de introducción en inglés.
- **Apéndice C: Conclusions:** este anexo contiene las conclusiones en inglés.



# Capítulo 2

## Estado de la cuestión

**Resumen:** Este capítulo se profundiza en el desarrollo para el usuario final y se estudian varias herramientas que han sido foco de estudio en esta rama de la informática. Además, se analizarán las principales herramientas del mercado en relación al desarrollo de videojuegos.

### 2.1. End user programming

Podemos definir programar como el proceso de idear y codificar un programa, lo cual nos lleva directamente a la definición del término programa. Una definición válida de este término sería *un conjunto de especificaciones que dado un input variable puede ser ejecutado en un dispositivo con capacidad de cómputo*.

Dentro del término programar podemos destacar el de *End User Development (EUD)* o desarrollo para el usuario final. Este término fue introducido en [Nardi \(1993\)](#) para referirse al uso de hojas de cálculo en el entorno laboral como “programar para alcanzar el resultado de un programa principalmente para el uso personal, más que general”. La principal distinción es que en las situaciones de desarrollo para el usuario final el programa es un medio para lograr un fin ([Fischer et al. 2004](#)).

Esta definición también incluye a gente que es capaz de desarrollar software pero que no son profesionales, sino que simplemente escriben código como apoyo a su labor principal. Por ejemplo, un desarrollador se dedica al desarrollo para el usuario final cuando escribe un código que ayuda a diagnosticar un error. En este caso el programa como tal no aporta un fin, sino que es usado por un colectivo (en este caso desarrolladores profesionales de software) como herramienta de apoyo a la hora de desempeñar sus funciones.

En ámbitos científicos y relacionados con la ingeniería es bastante común que los usuarios desarrollen complejos sistemas haciendo uso de lenguajes de programación como C++ o Java, sin embargo este grupo de usuarios es bastante reducido ([Scaffidi et al. 2005](#)). A día de hoy prácticamente todo el mundo utiliza algún software para llevar a cabo sus labores a pesar de no tener conocimientos relacionados con el desarrollo del mismo. Un claro ejemplo son las clásicas hojas de cálculo, las cuales ofrecen muchas facilidades de personalización hasta el punto de permitir la creación de secuencias de comandos por parte del usuario final. Esto ha hecho que se hayan convertido en una herramienta relevante en el mundo empresarial, hasta tal punto que cualquier usuario es capaz representar modelos de datos complejos sin apenas conocimientos técnicos.

Dentro del desarrollo para el usuario final nos encontramos con dos ramas bien diferenciados. La primera de ellas es el *End User Programming (EUP)* o la programación final del usuario que permite al propio usuario escribir programas sin que esta sea su principal función, es decir, crean software como parte de su trabajo ([Myers et al. 2006](#)). En este caso tendríamos las hojas de calculo ya comentadas, u otras herramientas como aquellas que ayudan en el diseño de una página web, por ejemplo.

### 2.1.1. Visual programming

La programación es algo que siempre ha sido clasificado como complejo, y esta es una de las principales motivaciones que han dado pie a este tipo de sistemas. Los primeros sistemas EUP fueron sistemas de programación visual, cuyo principal objetivo era simplificar la tarea de la programación. En [Haibt \(1959\)](#) se propone un sistema para dibujar diagramas de flujo de varios niveles que convertía un código en un lenguaje cualquiera a un lenguaje interno para ser procesado. Con esta premisa se pretenden eliminar ciertos aspectos superficiales de los lenguajes de programación que pueden diferir de un lenguaje a otro, como los tipos de las variables o los operadores.

Con el paso del tiempo se han ido desarrollando nuevos sistemas de programación visual, siendo uno de los más conocidos Scratch<sup>1</sup> ([Resnick et al. 2009](#)). Scratch es un lenguaje de programación visual desarrollado por el Instituto Tecnológico de Massachusetts (MIT) y diseñado para que todo el mundo pueda iniciarse en el mundo de la programación.

Otro ejemplo de programación visual es LabVIEW<sup>2</sup> ([Johnson, 1997](#)).

---

<sup>1</sup>[Scratch](https://scratch.mit.edu/) - <https://scratch.mit.edu/>

<sup>2</sup>[LabVIEW](https://www.ni.com/es-es/shop/labview.html) - <https://www.ni.com/es-es/shop/labview.html>

Esta herramienta, a diferencia de Scratch, no está pensada para iniciar al usuario en el mundo de la programación sino que es un entorno de desarrollo pensado para diseñar y probar sistemas hardware y software. Esta herramienta también cuenta con un lenguaje de programación visual que en este caso hace uso de cajas y líneas en vez de bloques. Las cajas son componentes lógicos que tienen entradas y salidas de datos, mientras que las líneas permiten conexiones entre los componentes lógicos simulando la funcionalidad de un cable real (Figura 2.1).

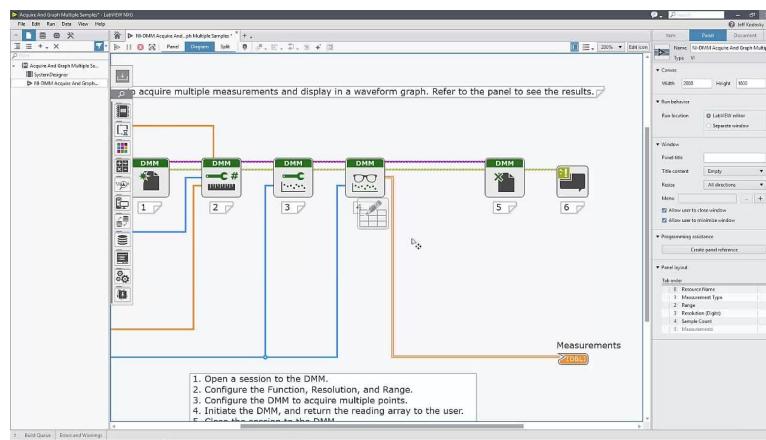


Figura 2.1: Captura de pantalla del editor de LabVIEW.

### 2.1.2. Programming by demonstration

Dentro de los sistemas EUP, otra de las ramas de investigación se centrar en permitir al usuario enseñar un nuevo comportamiento al demostrar acciones con ejemplos concretos. Esto es conocido como *Programming by Demonstration (PBD)* o programación por demostración, donde estos sistemas usan técnicas de inteligencia artificial para tratar de generalizar en base a los casos de ejemplo proporcionados por el usuario ([Cypher y Halbert 1993](#)).

En [Cypher \(1991\)](#) se propone EAGER, un sistema para entornos *HyperCard*, es decir, entornos propios de equipos Apple Macintosh que combinan una base de datos plana con una interfaz gráfica flexible que puede ser modificada por el usuario. Este entorno incluye un lenguaje de programación propio llamado *HyperTalk* para la manipulación de los datos y de la propia interfaz. EAGER toma como ejemplo los bucles de acciones que realiza el usuario sobre el propio sistema, de forma que al detectar un patrón que se repite muchas veces el propio sistema muestra en pantalla iconos y marca botones u otros elementos que puede que el usuario seleccione, basándose en los ejemplos que ha ido recibiendo previamente.

### 2.1.3. Programming with text

Las investigaciones en este ámbito también se han centrado en tratar de evitar varias dificultades dentro del desarrollo, en particular las relacionadas con la sintaxis de los propios lenguajes. Esto es lo que conocemos como *Programming with text* o programación con texto. Sin duda esta es la aproximación dentro de la EUP que más se parece a la programación clásica ya que a diferencia de las ramas ya desarrolladas, en este caso no disponemos de un lenguaje de programación visual sino que la interacción es a través de *inputs* de texto.

Dentro de esta categoría tenemos CoScripter ([Leshed et al. 2008](#)), un grabador de macros para navegadores web desarrollado por IBM. Esta herramienta es un *script* que está integrado con los navegadores web, de manera que el usuario solo ha de llenar varios campos en la propia web para realizar búsquedas sobre un tema en concreto (Figura 2.2).

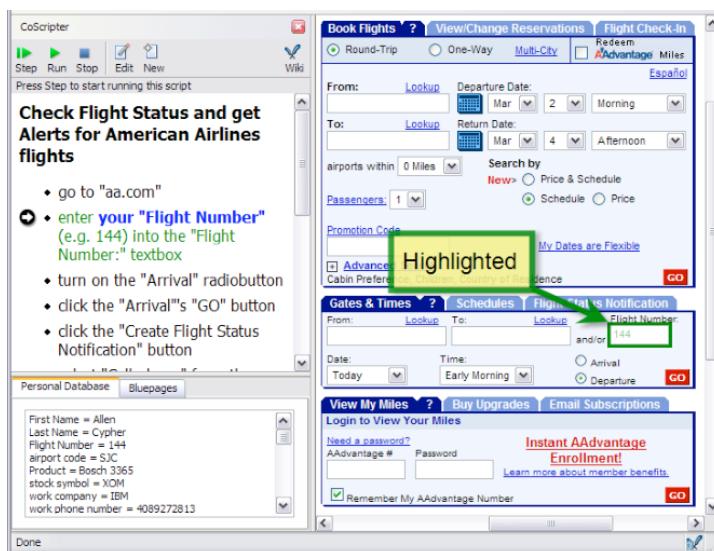


Figura 2.2: Captura de pantalla de la herramienta CoScripter.

En [Miller et al. \(1994\)](#) se proponen 3 proyectos para mejorar la enseñanza y el aprendizaje de conceptos de programación. El primer proyecto, GNOME, es un editor que permite a los alumnos crear programas libres de errores sintácticos. Para ello el sistema presenta el programa de forma jerárquica y los errores son presentados de forma incremental, es decir, tienen un contexto y no son mostrados en tiempo de compilación, facilitando así a los alumnos la comprensión del error (Figura 2.3).

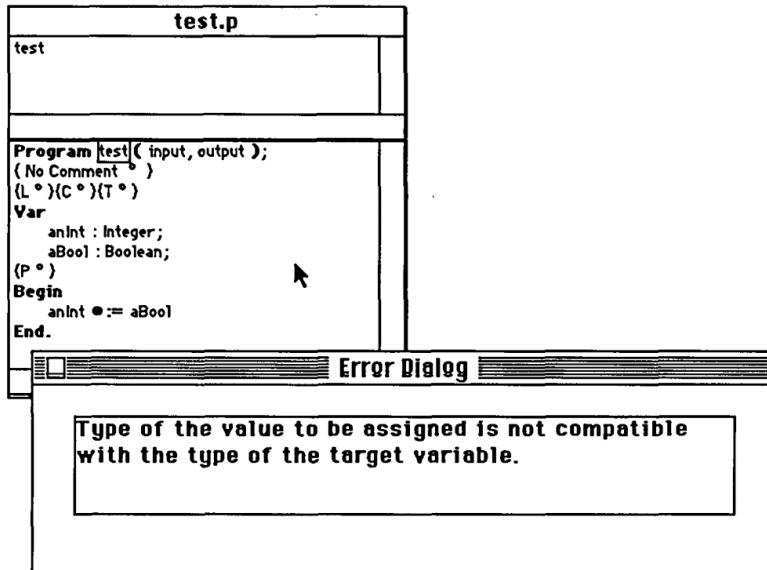


Figura 2.3: Ejemplo de un error de tipos en GNOME.

El segundo proyecto, Mac-Gnome, también es un editor centrado en la abstracción a alto nivel. Para ello incorpora numerosas interfaces gráficas y un amplio conjunto de herramientas para ver y editar programas, de forma que el trazado de errores y la visualización de los datos permite a los estudiantes aprender con más facilidad temas complejos como la recursión. El tercer y último proyecto también se trata de un editor de texto, ACSE, el cual incluye contenido multimedia y programación orientada a objetos.

#### 2.1.4. Programming by specification

La programación por especificación permite al usuario especificar un programa que luego es generado por la herramienta. En [Liu y Lieberman \(2005\)](#) se nos presenta *Metafor*, una herramienta que dada una especificación en lenguaje natural genera el correspondiente programa escrito en Python. En la figura 2.4 tenemos una captura de pantalla del editor que proporciona la herramienta. En la parte izquierda tenemos el *input* proporcionado por el usuario, mientras que en la parte derecha podemos observar el código Python generado.

The screenshot shows a window titled "74 Metafor interactive Natural Language Programming (hugo@media.mit.edu)". The left pane contains a transcript of a conversation:

```
[user] (01:28:32) Pacman is a character who loves to run through a maze and eat dots.
[agent] (01:28:32) Ok. I created a new agent Pacman that is a kind of character agent. I added the ability for Pacman to run, which can be through a maze. I added the ability for Pacman to eat. A dot is something which can be eaten.
[user] (01:28:53) Whenever Pacman eats a dot, it disappears and he wins a point.
[agent] (01:28:53) Ok. Now we are going to think about Pacman eating... When Pacman eats a dot, the dot disappears and Pacman wins a point. ...Now that we are done thinking about Pacman eating, we are going to resume thinking about __main__.

When Pacman is running through the maze, if a ghost catches him, then he loses and the game is over.
```

The right pane shows the generated Python code:

```
DEICTIC STACK: [(['Pacman', ['he', 'him', 'his', 'her', 'she', 'hers', 'who']), ('character', ['it']), ('maze', ['it']), ('dots', ['they', 'them']), ('dot', ['it']), ('Pacman', ['he', 'him', 'his', 'her', 'she', 'hers', 'who']), ('dot', ['it']), ('point', ['it'])]

DIR: ['__main__.Pacman', '__main__.dot']

CODETREE: [['__main__', 'FunctionT']]

def __main__():
    class Pacman(character):
        def run(maze):
            pass

        def eat(dot):
            dot.disappear()
            Pacman.win(point)

        def win(point):
            pass

    class dot:
        def disappear():
            pass
```

Figura 2.4: Ejemplo de funcionamiento de la herramienta Metafor.

Este tipo de herramientas presentan una importante limitación, pues no es posible determinar a priori que *output* se generará para cada *input* en particular. Además es imprescindible que el usuario debe conocer ambos lenguajes y solo acepta un determinado tipo de entradas, en este caso en lenguaje natural.

## 2.2. End user software engineering

La intención que hay detrás de la programación de usuario final es lo que distingue esta actividad de la programación como tal. Aún así, en ambos casos de trata de software que generalmente viene acompañado de una serie de métricas de calidad, reusabilidad y demás técnicas de ingeniería, lo que conocemos como ingeniería del software.

El objetivo de un programador de usuario final y de un desarrollador profesional es distinto, y por ello la ingeniería del software que se aplica en cada caso también lo es. Podemos definir la ingeniería del usuario final como la “*programación para el usuario final que implica actividades sistemáticas y disciplinadas que abordan los problemas de calidad del software*”(Ko et al. 2011). Al igual que en el resto de desarrollos, la ingeniería del software

también es un factor importante en el desarrollo final de usuario ya que cualquier producto que no haya sido desarrollado siguiendo unos procedimientos adecuados puede presentar perdida de datos o brechas de seguridad, más aún si se trata de un producto desarrollado por usuarios finales.

La principal diferencias entre ambos tipos de ingeniería del software reside en que en un ámbito profesional la atención que se presta a estas cuestiones es mucho mayor. Si un producto va destinado a ser usado por millones de usuarios es necesario que se apliquen métricas rigurosas durante el desarrollo. En contraposición tenemos la *End user software engineering* o ingeniería del software para el usuario final donde estas cuestiones también son importantes, sin embargo están relegadas a un segundo plano pues el objetivo principal es que el programa ayude a cumplir una meta. En Segal (2007) se llevó a cabo un estudio donde se comprobó que el software desarrollado por usuario finales no se valora como tal, que el proceso de desarrollo es altamente iterativo y que las pruebas del mismo no se consideran relevantes.

La ingeniería del software para el usuario final es una rama de investigación multidisciplinar que abarca desde interacción humano-computadora y educación hasta psicología. Aún así, se asemeja a la ingeniería de software profesional, y por ello podemos dividir su ciclo de vida en varios apartados:

- **Requisitos:** como se debe comportar el software en un entorno real.
- **Diseño y especificación:** como el software ha de comportarse a nivel interno para cumplir con los requerimientos.
- **Reutilización:** usar código existente para ahorrar tiempo y prevenir errores.
- **Testing y verificación:** asegurarse de que el software funciona correctamente y determinar los fallos existentes.
- **Debugging:** reparar los fallos encontrados en la fase de *testing* y verificación.

### 2.2.1. Requisitos

Los requisitos dentro de la ingeniería del software para el usuario final corresponden al cómo debe funcionar un software en un entorno real sin tener en cuenta cómo está implementado por dentro. Por ejemplo, un requisito para un programa de gestión de formularios de impuestos podría ser *Crear un modelo de impuestos 153 basado en mis datos financieros*. Este es un requisito de un resultado deseado que no explica como se llega a conseguir dicho resultado.

La principal diferencia con respecto a la ingeniería de software profesional reside en el origen de los propios requisitos. En un entorno profesional lo normal es que los usuarios, los clientes y los desarrolladores no pertenezcan al mismo grupo. En estos casos lo más común es llevar a cabo entrevistas o aplicar métodos que permitan llegar a los requisitos, sin embargo en el desarrollo para el usuario final no es así. En este caso el usuario final es a la vez cliente, usuario y desarrollador, por lo tanto la ingeniería del software en este caso difiere del ámbito profesional y el análisis de requisitos pasa simplemente a entender el contexto, las necesidades y las prioridades del propio usuario.

La toma de requisitos es mucho más fácil para el usuario final dado que los requisitos son establecidos por el propio usuario, y además los posibles cambios de los mismos no son tan drásticos como en un desarrollo profesional pues no hay que acordar nada con otra persona.

### 2.2.2. Diseño y especificación

Dentro de la ingeniería del software, el paso inmediato tras la toma de requisitos consiste en diseñar y especificar el sistema que se quiere construir. Estas especificaciones ayudan durante la implementación ya que permiten priorizar ciertos apartados del software como la calidad por encima del rendimiento, asegurando así el cumplimiento de todos los requisitos.

En la programación de usuario final el hecho de pasar de requisitos a diseño y especificación puede ser un proceso complicado, debido principalmente a que los beneficios de estos procedimientos pueden no estar claros para el usuario final. La mayoría de los beneficios de los procesos de diseño y de las especificaciones están creados a gran escala y con vistas de futuro, sin embargo el usuario final puede no tener en cuenta un uso del software tan a largo plazo. Por ejemplo, estudios relacionados con el uso de hojas de cálculos a nivel laboral muestran que los usuarios crean hojas de cálculo más y más complejas con el paso del tiempo ([Shaw 2004](#)).

En términos generales, la ingeniería del software dentro del desarrollo para el usuario final se centra en:

- **Diseñar el proceso:** los procesos de diseño de software se limitan a cómo los requisitos son traducidos en especificaciones de diseño, y luego estas en implementación. La mayoría de estos procesos son tomados de entornos de desarrollo profesional, como la capacidad de mantenimiento o el rendimiento del software, mediante la experiencia propia del usuario. Por ejemplo, [Ronen et al. \(1989\)](#) propone un proceso de diseño software centrado en asegurar que las hojas de cálculo

son fiables y seguras de actualizar, mientras que otros autores como [Rosson et al. \(2007\)](#) proponen un proceso de diseño relacionado con la programación web de usuario final basado en escenarios y mapas conceptuales con los que orientar a los desarrolladores hacia soluciones de diseño particulares.

- **Escribir especificaciones:** en la ingeniería del software profesional, la forma más directa de ver si se han satisfecho los requisitos del sistema consiste en escribir las especificaciones y usar herramientas que buscan inconsistencias con las especificaciones dadas. Para aplicar esta idea a la ingeniería del software para el usuario final tenemos varias herramientas como ViTSL ([Abraham et al. 2005](#)) o Gencel ([Abraham y Erwig, 2006](#)). ViTSL es una herramienta que introduce una especificación visual del lenguaje para hojas de cálculo que permite definir plantillas, evitando así errores relacionados con la omisión de datos o fallos de tipo mientras que Gencel es un sistema que comprueba la consistencia de las hojas de cálculo haciendo uso de casos de prueba autogenerados.

### 2.2.3. Reutilización

Entendemos reutilización como la capacidad de cambiar un código existente para adaptarlo a un nuevo contexto o programa. Dentro del desarrollo profesional de software es bastante común reutilizar código de otros proyectos, adaptar parte un programa para el desarrollo actual o simplemente usar librerías externas que nos proporcionan diversas funcionalidades mediante un API. Este tipo de prácticas son llevadas a cabo principalmente para ahorrar tiempo, evitar fallos al escribir código nuevo o para fomentar la mantenibilidad del código del desarrollo.

Es evidente que la motivación para estas prácticas también está presente en la programación para el usuario final, sin embargo en este caso la reutilización de código suele estar más enfocada en ahorrar tiempo que en conseguir un software mantenable. En algunos casos esto puede suponer un problema pues el usuario final no es un programador profesional, por lo tanto puede ocurrir que la reutilización de código sea lo que hace viable el proyecto pues para estos usuarios es más cómodo usar algo existente que funciona antes que hacerlo desde el principio ([Blackwell, 2002](#)).

A la hora de reutilizar código la forma más común de hacerlo es mediante APIs, las cuales han sido creadas específicamente para un uso profesional por lo tanto el usuario final puede tener dificultades a la hora de usarlas si no dispone de los conocimientos necesarios. Dada esta situación se han planteado varias fases de cara a la reutilización de código:

- **Encontrar código reutilizable:** existen diversas herramientas que ayudan al usuario final a encontrar código reutilizable como Code-Broker (Ye y Fischer, 2005). Esta herramienta toma un fragmento de código del programa y sugiere al usuario APIs que pueden serle útiles así como ejemplos de uso de la misma.
- **Reutilizar código:** una vez el usuario final ha encontrado un código reutilizable llega el momento de utilizarlo, tarea que puede complicarse en muchos casos si no se disponen de los conocimientos necesarios para ello. Diversos estudios han determinado que la principal problemática a la hora de usar código reutilizable o APIs por parte de usuarios finales es el no entendimiento de las abstracciones de los mismos, lo cual influye directamente en la comprensión del resultado obtenido así como en su obtención (Ko et al. 2004). Dada esta situación, existen diversas herramientas que ayudan al usuario a integrar código reutilizable en sus proyectos. En Lin y Landay (2008) se propone una herramienta multiplataforma para el desarrollo de interfaces de usuario, proporcionando una serie de patrones de diseño los cuales pueden ser parametrizados y combinados para formas interfaces más complejas 2.5.

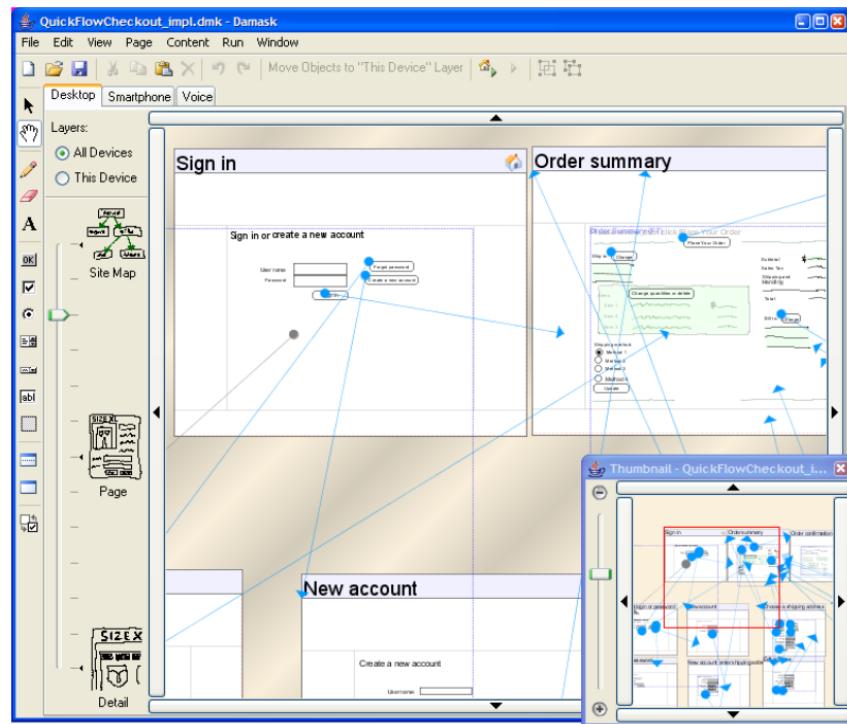


Figura 2.5: Captura de pantalla de la herramienta Damask.

### 2.2.4. *Testing* y verificación

Hay muchas formas de demostrar el correcto funcionamiento de un programa, aunque la meta siempre es la misma; conseguir que los usuarios se sientan lo más cómodos posible con el programa que usan. En la ingeniería del software profesional y en el desarrollo para el usuario final la meta a lograr en la misma, sin embargo los usuarios finales tienden a dar por hecho que sus programas funcionan correctamente cuando no es así. Por ello, la investigación en relación al *testing* y la verificación de programas desarrollados por usuarios finales tiende a abarcar la sobreconfianza de los mismos en relación a la corrección de dichos programas, principalmente en relación a las hojas de cálculo.

Los programadores por regla general suelen confiar en sus capacidades a la hora de desarrollar, pero a diferencia de los usuarios finales, éstos poseen experiencia y conocimientos en el ámbito del desarrollo. En contraposición a este hecho, tenemos que los usuarios finales están más seguros de sus capacidades con respecto a los desarrolladores profesionales, lo cual se traduce en un elevado número de errores en relación a la hojas de cálculo ([Panko, 1995](#)), hasta tal punto que entre el 5 % y el 23 % de las hojas de cálculo creadas por usuario finales poseen algún tipo de error ([Phalgun et al. 2005](#)).

Esta situación ha supuesto la creación de metodologías y herramientas para la detección de errores como WYSIWYT, una metodología para hacer test con los que medir la corrección de las hojas de cálculo ([Rothermel et al. 1998](#)). WYSIWYT o “*What You See Is What You Test*” es una metodología que realiza pruebas de caja blanca, es decir, se llevan a cabo pruebas relacionadas con detalles procedimentales las cuales por su diseño están fuertemente ligadas al código fuente, y en el caso de las hojas de cálculo, el código fuente son las celdas (Figura 2.6).

Student Grades							
	NAME	ID	HWAVG	MIDTERM	FINAL	COURSE	LETTER
1	Abbott, Mike	1,035	89	91	86	88.4	<input type="checkbox"/> B <input type="checkbox"/>
2	Farnes, Joan	7,649	92	94	92	92.6	<input type="checkbox"/> A <input type="checkbox"/>
3	Green, Matt	2,314	78	80	75	77.4	<input type="checkbox"/> C <input type="checkbox"/>
4	Smith, Scott	2,316	84	90	86	86.6	<input type="checkbox"/> B <input checked="" type="checkbox"/>
5	Thomas, Sue	9,857	89	89	89	93.45	<input type="checkbox"/> A <input type="checkbox"/>
6							
7	AVERAGE		86.4	<input checked="" type="checkbox"/>	88.8	<input checked="" type="checkbox"/>	85.6 <input checked="" type="checkbox"/> 87.69 <input type="checkbox"/>

Figura 2.6: Ejemplo de la metodología WYSIWYT en la validación de las celdas de una hoja de cálculo.

Puesto que probar todos los programas exigiría un número infinito de casos de prueba, las aproximaciones de caja blanca optan por medir cuando se ha hecho suficiente *testing* como para asegurarse de la corrección del programa. Con WYSIWYT el criterio usado para las pruebas es la *cobertura de definiciones* que, en el caso de las hojas de cálculo, consiste en comprobar la dependencia de datos. Con esta aproximación los usuarios finales pueden comprobar la consistencia de las hojas de cálculo de forma incremental a medida que desarrollan, y además pueden ser validadas por cualquier usuario sin tener en cuenta la sobreconfianza del mismo.

### 2.2.5. *Debugging*

Mientras que el *testing* y la verificación se encargan de detectar la presencia de errores en un programa, el *debugging* es el proceso de detectar y corregir esos errores. Esta tarea suele ser complicada incluso para los desarrolladores profesionales, y generalmente requiere el uso de herramientas de bajo nivel como *breakpoints* y *logs*. Un elevado número de errores puede hacer que las tareas de *debugging* sean aún más complicadas sumado a la falta de experiencia del usuario final. Además, la aparición de numerosos fallos en un desarrollo final de usuario puede dar lugar a estrategias de *debugging* “rápidas” que modifican el código hasta que parezca funcionar correctamente.

Dentro del *debugging* en la programación para el usuario final se nos plantea es el análisis de dependencias. Estas dependencias pueden ser de control cuando una serie de instrucciones solo son ejecutadas bajo ciertas condiciones, o de datos cuando la dependencia tiene en cuenta las variables del programa, por ejemplo a la hora de sumar dos variables (Xu et al. 2005). Una de las principales herramientas para el *debugging* es Ko y Myers (2004), la cual fue prototipada para el entorno de programación Alice (Dann et al. 2011). En ella los usuarios ejecutaban sus programas para ver una estructura 3D del mismo de forma que podían saber en cualquier momento la traza de ejecución (Figura 2.7).

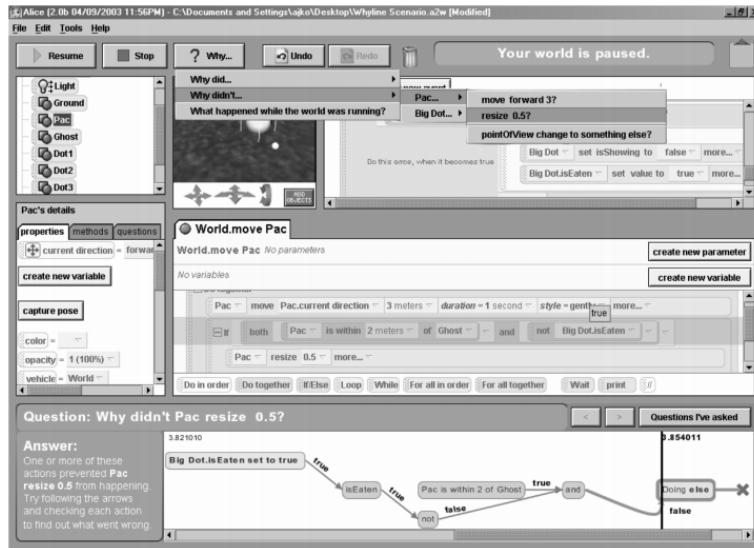


Figura 2.7: Captura de pantalla de la herramienta de *debugging* prototipada para Alice.

## 2.3. Herramientas profesionales

En esta sección se comentan las principales herramientas de desarrollo y diseño de videojuegos que son aptas para el desarrollo de usuario final.

### 2.3.1. RPG Maker

*RPG Maker*<sup>3</sup> es una herramienta para el desarrollo de videojuegos de rol (RPGs) creada por ASCII Corporation en 1988. Esta herramienta ha ido evolucionando siendo *RPG Maker MV* la versión más avanzadas de las más de treinta existentes hasta el momento.

Esta herramienta permite la creación de videojuegos de rol, y para ello pone a disposición del usuario un editor de mapas, un editor de eventos y un editor de bases de datos.

#### 2.3.1.1. Editor de mapas

El editor de mapas es el elemento clase de RPGMaker, el cual podemos ver en la figura 2.8. Este se compone de varios elementos:

---

<sup>3</sup>RPG Maker - <https://www.rpgmakerweb.com/>

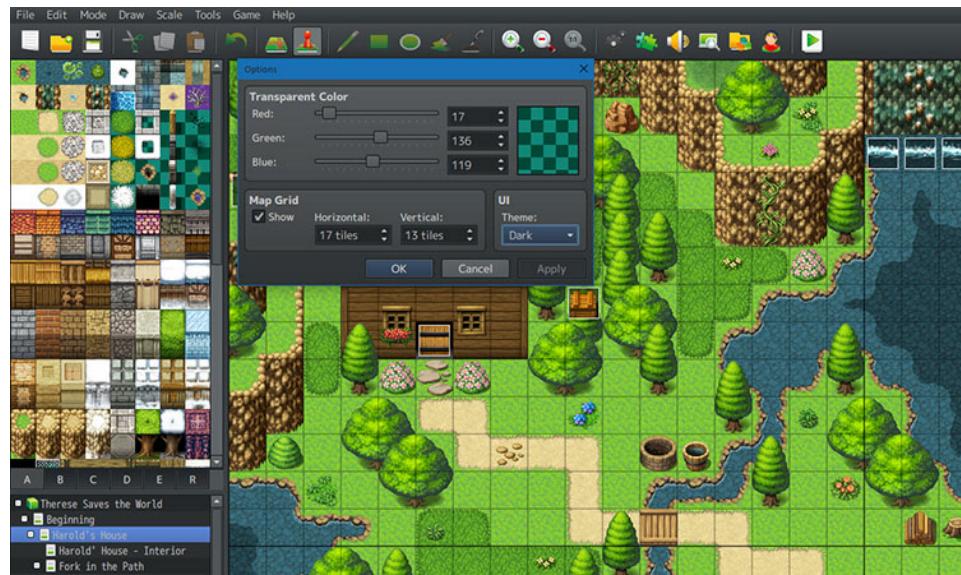


Figura 2.8: Editor de mapas de RPGMaker.

- **Paleta de texturas:** este elemento recoge todas las texturas que el usuario puede incluir en su juego.
- **Explorador de ficheros:** se ubica debajo de la paleta de texturas y muestra al usuario los distintos mapas que componen el juego.
- **Vista del mapa:** ocupa gran parte de la pantalla y permite al usuario editar el mapa actual, modificando texturas o personajes.
- **Barra de opciones:** en la parte superior de esta vista se ubica una barra con diversas opciones. Estas van desde limpiar por completo el mapa actual hasta hacer zoom o deshacer algún cambio.

### 2.3.1.2. Editor de eventos

En la figura 2.9 tenemos el editor de eventos de RPGMaker. Este incorpora un sistema de programación con *inputs* de texto con el que se puede determinar los comportamientos de los personajes frente a diversas situaciones.

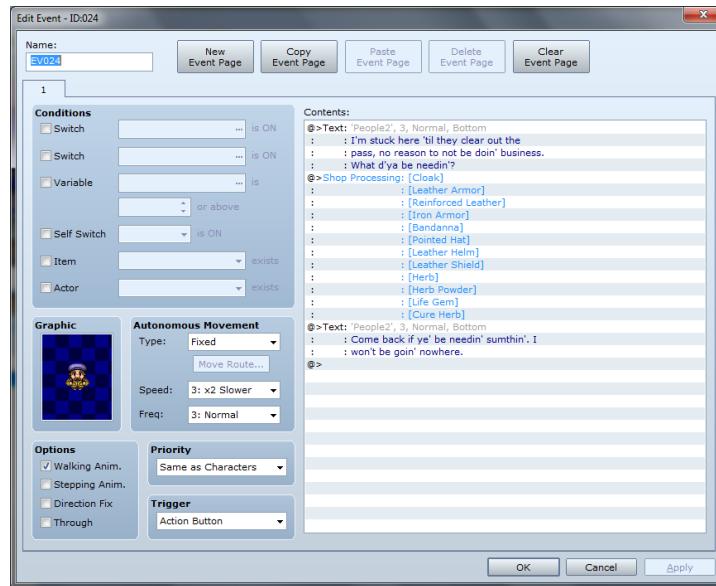


Figura 2.9: Editor de eventos de RPGMaker.

### 2.3.1.3. Editor de bases de datos

Por su parte, el editor de bases de datos permite la entrada de información mediante *inputs* de texto, con los cuales es posible modificar todo el contenido del juego.

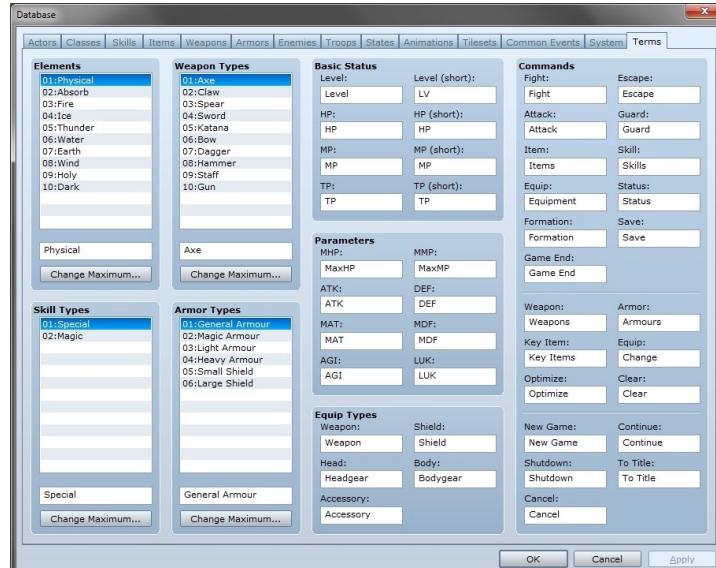


Figura 2.10: Editor de bases de datos de RPG Maker.

### 2.3.2. Construct

*Construct*<sup>4</sup> es un editor de videojuegos 2D basado en HTML5, pensado específicamente para usuarios que no son programadores permitiendo la creación de videojuegos mediante un lenguaje de programación visual.

Este lenguaje de programación visual es llamado *hojas de eventos* (Figura 2.11), y permite crear los eventos del juego y los comportamientos de los personajes.

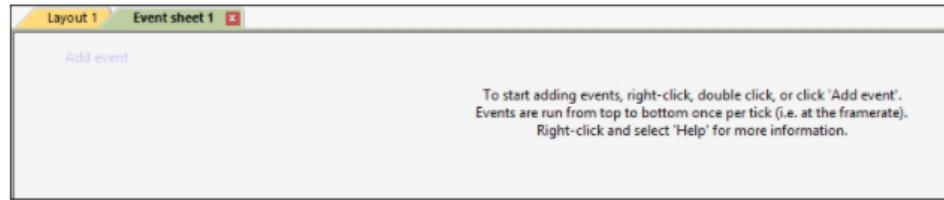


Figura 2.11: Hojas de eventos de Construct.

Cada hoja de eventos tiene una lista de eventos (Figura 2.12), los cuales contienen sentencias y *triggers* que son usados para implementar acciones y funciones. Esta herramienta también dispone de sub-eventos los cuales proveen las funcionalidades de elementos básicos de los lenguajes de programación, como sentencias OR y AND, permitiendo la creación de sistemas más complejos.

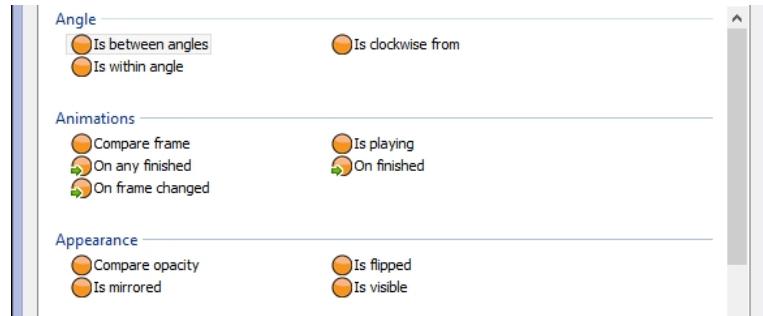


Figura 2.12: Ejemplos de eventos disponibles en Construct.

Estos eventos al ser añadidos también permiten al usuario modificarlos, especificando comprobaciones o condiciones que debes darse antes de que el objeto en cuestión sea añadido a la escena. Construct también soporta programación mediante el lenguaje JavaScript de forma opcional, permitiendo a usuarios con conocimientos avanzados crear proyectos más complejos.

<sup>4</sup>[Construct](https://www.construct.net/en) - <https://www.construct.net/en>

### 2.3.3. GameMaker: Studio

*GameMaker: Studio*<sup>5</sup> es una plataforma para el desarrollo de videojuegos para personas con pocos o nulos conocimientos de programación. Para ello incorpora un sistema de “*arrastrar y soltar*” que permite a los usuarios crear videojuegos de manera interactiva organizando iconos en la pantalla (Figura 2.13).

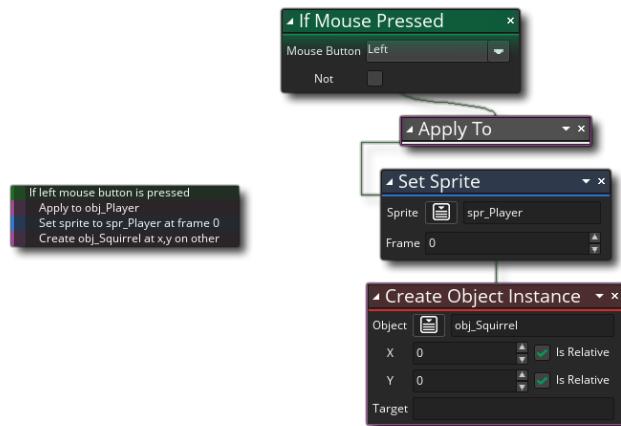


Figura 2.13: Ejemplo del sistema *point and click* de GameMaker: Studio.

Además, incorpora una serie de acciones básicas que cubren desde el movimiento o el dibujo básico de *sprites* hasta el control de estructuras dentro del propio editor (Figura 2.14).

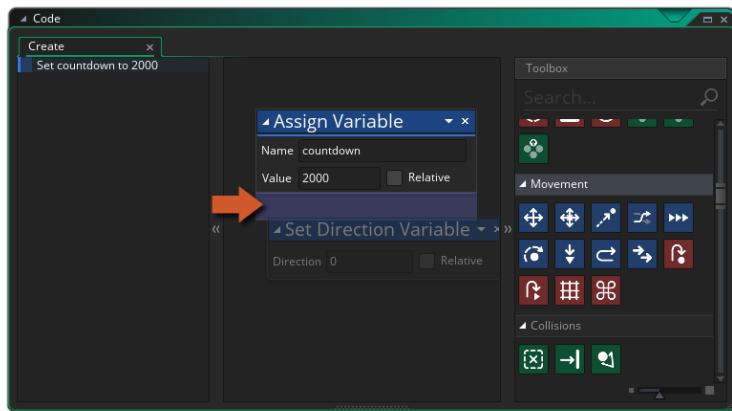


Figura 2.14: Opciones del editor de GameMaker: Studio.

<sup>5</sup>GameMaker: Studio - <https://www.yoggames.com/gamemaker>

Junto a este editor visual, *GameMaker: Studio* también incorpora un lenguaje de programación propio llamado *Game Maker Language* o GML, influenciado por *C* y *Pascal*. Este lenguaje permite a usuarios con cierto conocimientos técnicos crear videojuegos más complejos, aunque lo más interesante es la posibilidad de alternar en todo momento entre GML y la programación visual. Para ello, el propio editor permite convertir código GML en código visual y viceversa en cualquier momento (Figura 2.15).



Figura 2.15: Ejemplo de conversión entre código GML y código visual.

## 2.4. Herramientas educativas

### 2.4.1. MIT App Inventor

*MIT App Inventor*<sup>6</sup> es una herramienta en línea que originalmente fue creada por el MIT y más tarde adoptada por Google para ofrecer a sus usuarios una solución tecnológica con la que poder crear apps para dispositivos Android de forma sencilla.

Para ello, MIT App Inventor propone un proceso de creación de aplicaciones mostrado en la figura 2.16. Este proceso de creación de apps cuenta con 3 fases bien diferenciadas:

- **Diseño de pantallas:** se crean las distintas vistas que contendrá la aplicación. En ellas se sitúan diversos componentes como imágenes, botones o textos y se configuran sus propiedades.
- **Editor de bloques:** permite programar de forma visual e intuitiva el flujo de funcionamiento del programa haciendo uso de bloques.
- **Generador de app:** una vez se ha finalizado la fase de diseño e implementación se genera un instalador APK de la aplicación. Además

<sup>6</sup>MIT App Inventor - <https://appinventor.mit.edu/>

se puede obtener un código QR para su descarga directa en el dispositivo móvil o bien el APK para ser instalada, compartida o enviada a otro usuario.



Figura 2.16: Proceso de creación de una app con MIT App Inventor.

#### 2.4.1.1. Diseño de pantallas

En la figura 2.17 tenemos la interfaz gráfica que permite a los usuarios crear el diseño de su aplicación.

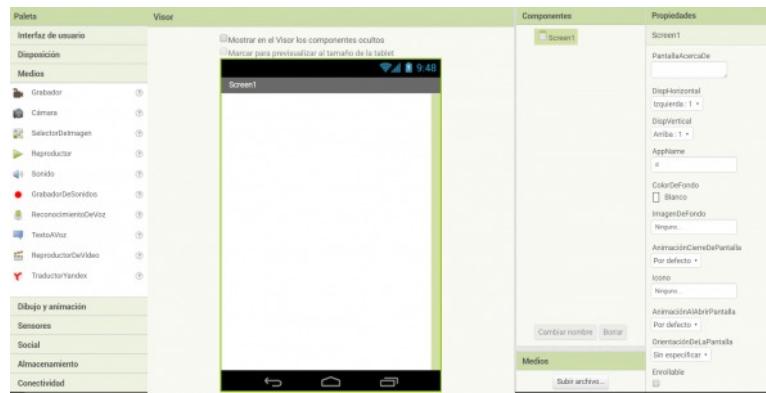


Figura 2.17: Interfaz de la opción diseño de pantallas de MIT App Inventor.

Esta interfaz se divide en 4 grandes apartados:

- **Paleta:** se encuentra a la izquierda de la pantalla y recoge todos los elementos que podemos usar a la hora de crear una aplicación como botones, componentes de vídeo o sonidos.
- **Visor:** es el elemento central de la vista de diseño de pantallas y se ubica en el centro de la misma. Esta parte de la interfaz se encarga de simular la pantalla de un dispositivo móvil, de forma que sobre ella se irán añadiendo los diferentes componentes y se dará forma al aspecto final de la aplicación.
- **Componentes:** este apartado muestra todos los componentes que han sido añadidos al Visor y estarán presentes en la aplicación.
- **Propiedades:** es el último elemento de la vista de diseño de pantallas y se sitúa a la derecha de los demás elementos de la interfaz. Este objeto muestra todas las propiedades del componente que hayamos seleccionado en el apartado Componentes. Desde aquí es posible modificar todas las propiedades de los elementos que forman parte de la interfaz gráfica de la aplicación.

#### 2.4.1.2. Editor de bloques

Esta vista es usada en la segunda fase de creación de aplicaciones de Mit App Inventor inmediatamente después de haber terminado el diseño de la aplicación. En la figura 2.18 tenemos la vista del editor de bloques que, al igual que con la vista de diseño de pantallas, se divide en dos elementos:



Figura 2.18: Interfaz de la opción editor de bloques de MIT App Inventor.

- **Bloques:** esta parte de la vista contiene los bloques que podemos usar para crear comportamientos así como una lista con todos los componentes que añadimos en la fase anterior a la aplicación.
- **Visor:** en esta parte tenemos el editor de bloques con el podemos añadir los comportamientos deseados a los componentes seleccionados.

### 2.4.2. Scratch

Scratch es una herramienta que permite crear historias interactivas, juegos y animaciones sin tener conocimientos técnicos. Para ello nos proporciona una interfaz que podemos observar en la figura 2.19 dividida en varios componentes:

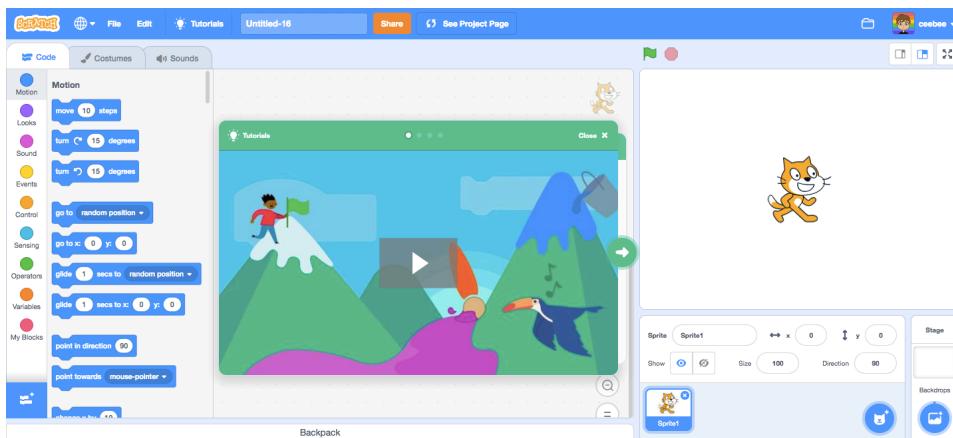


Figura 2.19: Interfaz gráfica de la herramienta Scracth.

- **Ventana de bloques, disfraces y sonidos:** este componente de la interfaz permite al usuario alternar entre bloques usados para construir el juego, disfraces o sprites para los personajes del juego y sonidos. Los bloques a su vez se dividen en varias categorías:

1. Movimiento: mueve objetos y cambia ángulos.
2. Apariencia: modifican el aspecto visual del juego, bien añadiendo elementos de conversación a los objetos del mismo, cambiando el fondo del juego o ampliando/reduciendo elementos del mismo.
3. Sonido: reproduce ficheros de audio y secuencias de sonidos programables.
4. Datos: creación de variables y listas.
5. Eventos: manejadores de eventos situados al comienzo de un grupo de instrucciones.

6. Sensores: permiten a los objetos del juego interactuar con los demás elementos del mismo.
7. Operadores: operadores matemáticos que modifican las variables del juego.
8. Otros bloques: controlan dispositivos externos.

- **Contenedor central:** este elemento posee el juego que se está creando.
- **Panel lateral derecho:** este elemento contiene un explorador de ficheros así como varias opciones relacionadas con el sonido o los fondos del propio juego.

El sistema de programación visual con bloques es el elemento clave de esta herramienta. A continuación tenemos a modo de ejemplo parte del clásico juego *pong* codificado con Scratch. La figura 2.20 corresponde con el principio del juego e inicializa la posición de la bola. Para ello, se usa un evento que conecta el bloque con la ejecución del juego y a continuación se usan tres eventos más. Los dos primeros son eventos de movimiento que se encargan de fijar la posición de la bola en el centro de la pantalla y de establecer la dirección de la misma al comienzo del juego. El tercer evento se encarga de determinar dicha dirección, y para ello se usa un bloque operador encargado de seleccionar un número aleatorio entre dos valores.



Figura 2.20: Fragmento del clásico juego *pong* codificado con Scratch.

La figura 2.21 contiene el bloque encargado de gestionar la colisión de la bola con uno de los dos jugadores. En este caso comenzamos de nuevo con un bloque de evento seguido de otros dos bloques de control, *forever* e *if*, que corresponden a las instrucciones *while* e *if* de los lenguajes de programación comunes. El bloque de control *if* contiene otro bloque operador que determina si la bola ha tocado a un jugador u otro, seguido de dos bloques de movimiento para determinar la dirección y mover la bola, además de un bloque de datos para modificar la velocidad de la misma.

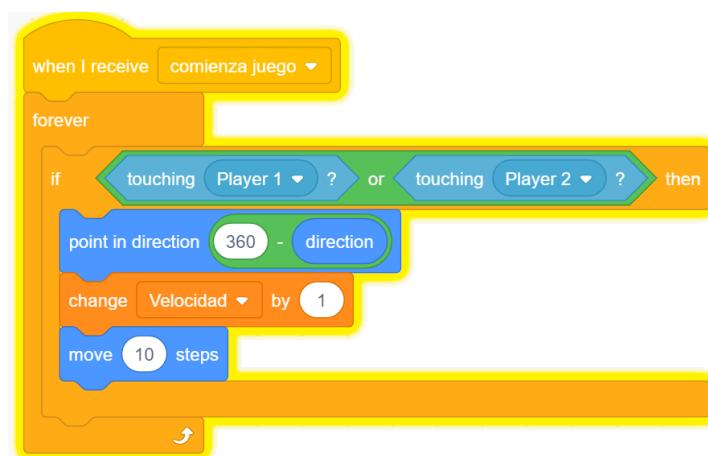


Figura 2.21: Fragmento del clásico juego *pong* codificado con Scracth.



## Capítulo 3

# Herramientas de desarrollo

**Resumen:** En este capítulo se expone las herramientas y las tecnologías utilizadas durante el desarrollo del sistema.

### 3.1. Unreal Engine

Unreal Engine es un motor de videojuegos creado por Epic Games<sup>1</sup> en 1998. Este motor fue pensado inicialmente para desarrollar *shooters* en primera persona, aunque con el paso del tiempo se ha usado con éxito en gran variedad de géneros como el RPG, o los MMOPRG (*massively multiplayer online role-playing game*).

En las primeras versiones del motor este hacía uso de un lenguaje de programación propio llamado *UnrealScript*, el cual al igual que otros lenguajes como Java permitía la programación orientada a objetos sin herencia múltiple. Este sería el lenguaje de programación establecido hasta que en la GDC (*Game Developers Conference*) de 2012 se anunció que sería substituido por C++ y se remplazaría el sistema de programación visual existente (Kismet) por el un nuevo sistema llamado *Blueprint*.

#### 3.1.1. Blueprints

Los *Blueprints* son el sistema de programación visual utilizado actualmente por el motor Unreal Engine en su versión 4. Este sistema se basa en una interfaz con nodos (Figura 3.1) que actúan como elementos de *gameplay* y al igual que en muchos otros lenguajes de programación visual es posible

---

<sup>1</sup>Epic Games - <https://www.epicgames.com/store/es-ES/>

definir objetos y clases desde el propio editor.

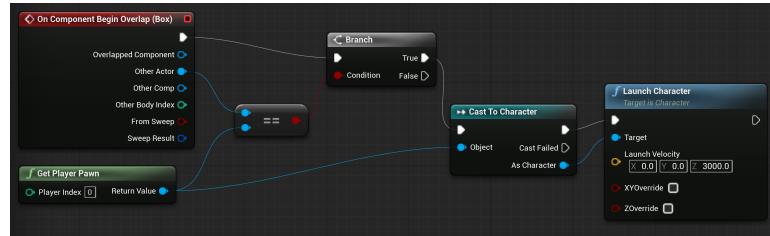


Figura 3.1: Ejemplo de programación visual con *Blueprints*.

Este sistema está planteado principalmente para ser usado por diseñadores o artistas, en definitiva, gente que es desarrollador profesional de videojuegos pero cuyo trabajo no gira en torno a la programación del software. Aún así, se trata de un sistema lo suficiente flexible y potente como para ser usado de forma íntegra por desarrolladores profesionales, ya que toda la lógica que hay por debajo de los *Blueprints* se sustenta en C++.

### 3.1.2. C++

C++ es el lenguaje de programación que usa Unreal Engine 4, en concreto la versión 11 del mismo. No obstante, esta versión de C++ tiene una capa de customización añadida por parte de Epic Games que hace que la programación con este motor sea algo diferente.

El primer punto a destacar es que la versión de C++ que usa Unreal Enigne 4 no utiliza el recolector de basura del propio lenguaje de programación. Esto es así debido a la jerarquía de clases existente dentro del motor, la cual se puede observar en la figura 3.2.

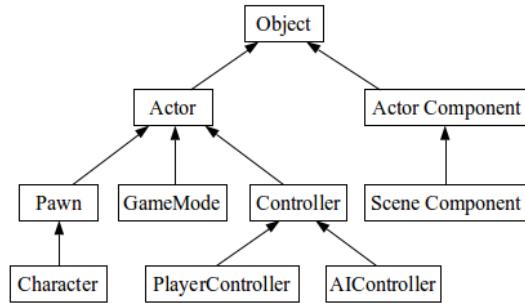


Figura 3.2: Jerarquía de clases básica de Unreal Engine 4.

En esta jerarquía el objeto más básico es *Object*, y todos los objetos existentes en un proyecto de Unreal Engine heredan de esta clase. Esta es la razón por la que existe un recolector de basura específico, pues al saber exactamente como están implementadas todas las clases el rendimiento de dicho sistema es superior. Contar con una clase base como es *Object* también nos da otras ventajas, como la capacidad de serializar cualquier tipo de objeto con una sola interfaz o grandes facilidades a la hora de replicar contenidos en red en desarrollos multijugador.

De la clase *Object* heredan otras dos, *Actor* y *ActorComponent*. Los objetos que pertenecen a *Actor* son aquellos que serán colocados en el nivel, es decir, aquellos objetos del juego con los que el jugador podrá interactuar como una tienda o NPCs. De esta clase heredan otras tres, *Pawn*, *GameMode* y *Controller*, las cuales se comentarán más adelante.

*ActorComponent* es la clase base que define la estructura de los componentes, objetos que proporcionan funcionalidades como moverse o girar de forma que un actor puede contener una colección de *ActorComponent* que le proporcionen ciertas funcionalidades.

Otro matiz a destacar en el desarrollo con Unreal Engine son los constructores y los especificadores de función. Los constructores en la versión de C++ de este motor no pueden llevar parámetros. Esto es así por el recolector de basura y la creación de los objetos, la cual es llevada a cabo mediante una llamada estática que toma la clase del objeto y una serie de datos adicionales como la localización del objeto o su rotación (Figura 3.3).

---

```

1 ARPGCharacter* Character = GetWorld()->SpawnActor<ARPG2DCharacter>
2 (FVector::ZeroVector, FRotator::ZeroRotator);
3 // llamadas a setters oportunos

```

---

Figura 3.3: Código que muestra la instanciación de un Actor en un nivel.

Por otro lado tenemos los especificadores de función, unas sentencias que permiten realizar diversas acciones sobre las clases y variables definidas en C++. Entre estas acciones tenemos la modificación de variables de un objeto, la instanciación de clases desde el editor de blueprints o la posibilidad de re-implementar ciertos métodos, entre muchas otras. En la figura 3.4 tenemos un fragmento de código con un objeto y una función.

El objeto está marcado como *UPROPERTY* y presenta dos especificadores de función:

- ***EditAnywhere***: Permite que la variable sea editada desde el editor de programación visual.
- ***BlueprintReadWrite***: Permite que la variable sea leída desde el editor de *Blueprints*.

Por otro lado, la función está marcada como *UFUNCTION* y también posee dos especificadores de función:

- ***BlueprintCallable***: Permite que la función sea llamada desde el editor de *Blueprints*.
- ***BlueprintPure***: Indica al compilador que esta función no realiza modificaciones de datos.

---

```

1 protected:
2     UPROPERTY(EditAnywhere, BlueprintReadWrite)
3         ABaseAIController * RPGController
4
5     UFUNCTION(BlueprintCallable, BlueprintPure)
6         ABaseAIController * GetCharacterAIController() {
7             return RPGController;
8         }

```

---

Figura 3.4: Ejemplo de especificadores en funciones y variables.

### 3.1.3. Clases básicas en un proyecto de Unreal Engine

En un videojuego hay muchos elementos relacionados con el *gamaplay* que hay que tener en cuenta durante el desarrollo, como por ejemplo las condiciones de victoria o los jugadores. Para gestionar este tipo de datos, Unreal Engine dispone de varias clases que comentaremos a continuación.

La primera de ellas y la más importante es la clase *GameMode*, la clase básica que sustenta cualquier proyecto. El *GameMode* es quien define las condiciones de victoria y las reglas de juego. Además se ocupa de otras tareas como *spawnnear* jugadores en el mapa o de gestionar las conexiones y desconexiones de los jugadores en un entorno multijugador.

Por otro lado tenemos las clases *Controller* que define dos tipos de objetos, los *PlayerController* y los *AIController*. La clase *Pawn* corresponde al jugador y es controlada por la clase *PlayerController* de tal forma que, en función del tipo de controlador:

- el objeto de la clase *Pawn* recibe los *inputs* del jugador (bien desde un mando u otro soporte físico) cuando es controlada por una instancia de la clase *PlayerController*.
- el objeto de la clase *Pawn* es poseída por una IA cuando es controlada por la clase *AIController*.

El juego normalmente reporta algún tipo de progreso, como el tiempo transcurrido o los jugadores restantes, en definitiva, información relativa al juego que es conocida por todos los jugadores. Aquí entra en juego la clase *GameState*, la cual se encarga de almacenar esta información que no es gestionada por el *GameMode*. Esta arquitectura está determinada teniendo en cuenta el multijugador, de forma que el *GameMode* está localizado en el servidor mientras que el *GameState* se localiza replicado en los clientes y su información es actualizada por el servidor.

## 3.2. Otras herramientas de desarrollo

### 3.2.1. Mixamo

*Mixamo*<sup>2</sup> es un servicio de animación de personajes online de Adobe. Esta herramienta permite a los usuarios crear personajes 3D eligiendo las partes del cuerpo, la ropa y textura de los mismos, los cuales pueden ser exportados para varios motores de videojuegos.

Además ofrece la posibilidad de *riggear* personajes, es decir, permite crear un sistema de controles digitales y agregárselos a un modelo 3D para que así pueda ser animado.

### 3.2.2. Aseprite

*Aseprite*<sup>3</sup> es un editor de gráficos rasterizados diseñado específicamente para la creación de *sprites* y animaciones pixel art.

El programa cuenta con herramientas de dibujado comunes a otros editores de gráficos bitmap como lápiz, goma de borrar, herramienta de relleno y diversas herramientas de selección.

---

<sup>2</sup>[Mixamo](https://www.mixamo.com/) - <https://www.mixamo.com/>

<sup>3</sup>[Aseprite](https://www.aseprite.org/) - <https://www.aseprite.org/>



## Capítulo 4

# Metodología y gestión del proyecto

**Resumen:** En este capítulo se exponen las metodologías, división de tareas y la planificación seguida para la consecución del proyecto.

Para el desarrollo de este proyecto se han fijado 2 hitos. En el primero de ellos se implementó un prototipo reducido de *TRPG Maker* el cual fue evaluado con usuarios, mientras que en el segundo hito se construyó, sobre el primer prototipo, la versión final del sistema usando el *feedback* proporcionado por los usuarios.

El primer prototipo de *TRPG Maker* se enfocó a un público concreto, en este caso niños. Para ello se destinaron 3 meses al desarrollo a este primer prototipo, desde Marzo hasta Mayo, y durante los meses de Junio y Julio se levó a cabo una fase de experimentación donde se creó un concurso en colaboración con Electronics Arts. En él se publicó la herramienta con un paquete de *assets* dirigido a un público joven de forma que los usuarios de entre 8 y 12 años pudieran participar en el concurso.

Tras haber llevado a cabo esta fase de experimentación observamos que el número de participantes era muy reducido y el perfil de los mismos muy heterogéneo, tratándose principalmente de perfiles relacionados con el desarrollo de videojuegos. En vista de esta situación vimos oportuno redirigir el proyecto hacia un público distinto, en concreto hacia un perfil de *gamedev* pues este tipo de usuarios demostraron especial interés en la herramienta durante la fase de experimentación. Para ello, se añadieron diferentes módulos y nuevas funcionalidades al proyecto con el objetivo de convertirlo en una herramienta más profesional.

Durante los meses de Junio, Julio y Agosto tuvo lugar el segundo hito donde se integró el *feedback* obtenido de la experimentación con usuarios a medida que se desarrollaban nuevas funcionalidades. Al final de este segundo hito la herramienta se publicó en forma gratuita.

Para el desarrollo del proyecto se ha optado por una metodología SCRUM, decisión motivada por la flexibilidad que ofrece durante el desarrollo y para evitar los posibles problemas con una planificación más *clásica*, como un desarrollo en cascada donde volver a un punto anterior del desarrollo es inviable.

La versión final de *TRPG Maker* se compone de 5 módulos internos que articulan el funcionamiento del sistema. Estos son:

- **Editor de niveles 3D/2D.**
- **Editor de base de datos.**
- **Editor de conversaciones.**
- **Editor de mapas.**
- **Gestor de colas.**

Para el desarrollo del proyecto se ha optado por una metodología SCRUM, decisión motivada por la flexibilidad que ofrece durante el desarrollo y para evitar los posibles problemas con una planificación más *clásica*, como un desarrollo en cascada donde volver a un punto anterior del desarrollo es inviable. Por ello a cada hito se le han asignado una serie de módulos a implementar y cada uno de ellos ha sido descompuesto en tareas atómicas.

#### 4.1. Primer hito

El primer hito ha ocupado los meses de Marzo, Abril y Mayo, y como resultado de esta fase del desarrollo se ha creado *TRPG Maker: The Good Doctors*, una versión reducida de *TRPG Maker* enfocada a un público joven. Para ello se ha usado un paquete de *assets* relacionado con el coronavirus donde los personajes jugables son los sanitarios mientras que los enemigos son el propio virus. En este hito se han desarrollado el editor de niveles 3D, el editor de base de datos y el editor de conversaciones, aunque en la prueba con usuarios que realizamos al acabar este hito solo se incluyó el editor de niveles 3D, pues es la parte más propensa a fallos y la más interesante de probar con usuarios.

#### 4.1.1. Tareas del editor de escenarios 3D

EDITOR DE ESCENARIOS 3D	
<b>T1 - ANIMAR PERSONAJES</b>	
Descripción	Animar los modelos 3D comprados en la Unity Assets Store con Mixamo. Para ello es necesario crear 8 animaciones diferentes para cada uno de los 6 modelos.
Importancia	Alta
Coste	Alto
Prioridad	Alta
Dependencias	—
<b>T2 - OBTENER DECORACIONES</b>	
Descripción	Obtener decoraciones de Sketchfab que será usadas en los escenarios. Es necesario incrustar los materiales en las mallas y reescalar todas las mallas en proporción a los modelos de los personajes.
Importancia	Alta
Coste	Alto
Prioridad	Media
Dependencias	—
<b>T3 - CREAR MALLAS PARA LOS TERRENOS</b>	
Descripción	Crear las mallas de los terrenos usando Blender. Es necesario hacerlas en proporción al tamaño de los modelos y hay que mapear los UVs de sus materiales en texturas de tamaño 1024x1024 pixeles.
Importancia	Alta
Coste	Alto
Prioridad	Alta
Dependencias	—
<b>T4 - CREAR TEXTURAS PARA LOS TERRENOS</b>	
Descripción	Usar Aseprite para crear diversas texturas partiendo de los UVs de las mallas del terreno.
Importancia	Media
Coste	Bajo
Prioridad	Baja
Dependencias	T3

<b>T5 - FUNCIONALIDAD COLOCAR TERRENO</b>	
Descripción	Implementar la funcionalidad que permite colocar terreno en el escenario. Deben poderse colocar 4 tipos de terrenos (bloque, bloque de media altura, rampa y rampa de media altura) con restricciones de colocación entre ellos: <ul style="list-style-type: none"> <li>■ Las rampas y rampas de media altura no pueden tener otro bloque encima.</li> <li>■ Las rampas y rampas de media altura no pueden tener decoraciones encima.</li> </ul>
Importancia	Alta
Coste	Medio
Prioridad	Alta
Dependencias	T3 - T4
<b>T6 - FUNCIONALIDAD COLOCAR PERSONAJES</b>	
Descripción	Implementar la funcionalidad que permite colocar personajes en el escenario.
Importancia	Alta
Coste	Bajo
Prioridad	Alta
Dependencias	T3 - T5
<b>T7 - FUNCIONALIDAD COLOCAR DECORACIONES</b>	
Descripción	Implementar la funcionalidad que permite colocar decoraciones en el escenario.
Importancia	Alta
Coste	Bajo
Prioridad	Alta
Dependencias	T2 - T5
<b>T8 - FUNCIONALIDAD ELIMINAR OBJETO DEL ESCENARIO</b>	
Descripción	Implementar la funcionalidad que permite eliminar elementos del escenario.
Importancia	Alta
Coste	Bajo
Prioridad	Baja
Dependencias	T5 - T6 - T7
<b>T9 - IMPLEMENTAR SISTEMA DE COMBATE</b>	
Descripción	Implementar la lógica necesaria para el sistema de combate de los escenarios. Deben contemplarse varias acciones entre ellas <i>Atacar</i> , <i>Mover</i> y <i>Esperar</i> .
Importancia	Alta
Coste	Alto
Prioridad	Alta
Dependencias	T1 - T3

T10 - IMPLEMENTAR SISTEMA DE MOVIMIENTOS	
Descripción	Implementar la lógica necesaria para el sistema de movimiento de los personajes por el escenario.
Importancia	Alta
Coste	Bajo
Prioridad	Alta
Dependencias	T1 - T3

T11 - IMPLEMENTAR SISTEMA DE ANIMACIONES DE PERSONAJES	
Descripción	Implementar la lógica necesaria para generar la máquina de estados con las animaciones de los personajes.
Importancia	Alta
Coste	Bajo
Prioridad	Alta
Dependencias	T1 - T3 - T9 - T10

T12 - IMPLEMENTAR MÁQUINA DE ESTADO DEL EDITOR	
Descripción	Implementar la lógica encargada de gestionar las acciones del editor así como el transcurso de las batallas.
Importancia	Alta
Coste	Alto
Prioridad	Alta
Dependencias	Todas las anteriores

Tabla 4.1: Desglose de tareas para el editor de escenarios 3D.

#### 4.1.2. Tareas del editor de bases de datos

EDITOR DE BASES DE DATOS	
<b>T1 - DESARROLLO DE COMPONENTES BÁSICOS</b>	
Descripción	Implementar componentes básicos como inputs de texto genéricos o contadores de valores parametrizables que serán usados en las vistas del editor de bases de datos.
Importancia	Alta
Coste	Alto
Prioridad	Alta
Dependencias	—
<b>T2 - DESARROLLO DE LA VISTA DE PERSONAJES</b>	
Descripción	Implementar la edición de personajes de la base de datos. Es necesario poder modificar sus valores base así como sus habilidades.
Importancia	Media
Coste	Bajo
Prioridad	Media
Dependencias	T1
<b>T3 - DESARROLLO DE LA VISTA DE HABILIDADES</b>	
Descripción	Implementar la edición de habilidades de la base de datos. Es necesario poder modificar sus valor de daño y coste así como el rango de acción y el área de efecto de la misma.
Importancia	Media
Coste	Bajo
Prioridad	Media
Dependencias	T1
<b>T4 - DESARROLLO DE LA VISTA DE OBJETOS</b>	
Descripción	Implementar la edición de objetos de la base de datos. Es necesario poder modificar el efecto que producen al usarlo así como otros datos básicos como el nombre o el daño que hacen.
Importancia	Media
Coste	Bajo
Prioridad	Media
Dependencias	T1

T5 - DESARROLLO DE LA VISTA DE CLASES	
Descripción	Implementar la edición de clases de la base de datos. Es necesario permitir al usuario modificar los parámetros básicos de cada clase así como los escalados de los mismos.
Importancia	Media
Coste	Bajo
Prioridad	Media
Dependencias	T1
T6 - CREACIÓN DE RECURSOS GRÁFICOS	
Descripción	Creación de los recursos gráficos necesarios para las vistas del editor de bases de datos.
Importancia	Media
Coste	Alto
Prioridad	Media
Dependencias	Todas las anteriores
T7 - INTEGRACIÓN EN TRPG MAKER	
Descripción	Integración de los componentes gráficos creados dentro del core del proyecto.
Importancia	Alta
Coste	Bajo
Prioridad	Alto
Dependencias	Todas las anteriores
T8 - GENERAR CONTENIDOS BÁSICOS	
Descripción	Generación de ficheros de configuración con los contenidos base que se mostrarán en cada vista.
Importancia	Alta
Coste	Bajo
Prioridad	Bajo
Dependencias	—

Tabla 4.2: Desglose de tareas para el editor de bases de datos.

#### 4.1.3. Tareas del editor de conversaciones

EDITOR DE CONVERSACIONES	
<b>T1 - DESARROLLO DE COMPONENTES BÁSICOS</b>	
Descripción	Implementar los componentes básicos que será usados a la hora de mostrar los diálogos de las conversaciones.
Importancia	Alta
Coste	Medio
Prioridad	Alta
Dependencias	—
<b>T2 - DESARROLLO ESTRUCTURA DE DATOS Y DE JERARQUÍAS DE CLASES DE LOS DIÁLOGOS</b>	
Descripción	Implementar las clases del sistema de diálogos y los propios diálogos en sí además de las estructuras de datos pertinentes para que sean escritos y leídos en memoria.
Importancia	Alta
Coste	Alta
Prioridad	Alta
Dependencias	T1
<b>T3 - GENERACIÓN DE RECURSOS GRÁFICOS</b>	
Descripción	Generar los recursos gráficos que serán usados en la vista de diálogos.
Importancia	Media
Coste	Bajo
Prioridad	Media
Dependencias	T1 - T2
<b>T4 - IMPLEMENTAR VISOR DE DIÁLOGOS</b>	
Descripción	Implementar un reproductor de diálogos.
Importancia	Alta
Coste	Alto
Prioridad	Media
Dependencias	Todas las anteriores
<b>T5 - INTEGRACIÓN EN EL CORE DE TRPG MAKER</b>	
Descripción	Añadir el módulo implementado al núcleo del sistema.
Importancia	Alta
Coste	Bajo
Prioridad	Bajo
Dependencias	Todas las anteriores

Tabla 4.3: Desglose de tareas para el editor de conversaciones.

## 4.2. Segundo hito

El segundo hito ha ocupado los meses de Junio, Julio y Agosto, y como resultado de esta fase del desarrollo se ha creado una versión más completa de *TRPG Maker*. Esta versión incluye un paquete de *assets* genéricos relacionados con los juegos de rol táctico y contiene todos los módulos ya comentados.

En este hito se han desarrollado el editor de escenarios 2D, el editor de mapas y el gestor de colas de eventos

### 4.2.1. Tareas del editor de escenarios 2D

EDITOR DE ESCENARIOS 2D	
<b>T1 - GENERACIÓN DE RECURSOS GRÁFICOS</b>	
Descripción	Generar sprites de los personajes, decoraciones y animaciones que serán usados en el editor.
Importancia	Alta
Coste	Alto
Prioridad	Media
Dependencias	—
<b>T2 - IMPLEMENTAR CLASES DEL PERSONAJE 2D</b>	
Descripción	Extender la implementación existente del editor de escenarios 3D para crear un nuevo personaje. El nuevo personaje contendrá <i>sprites</i> en vez de una malla 3D.
Importancia	Alta
Coste	Alto
Prioridad	Alta
Dependencias	—
<b>T3 - EXTENDER COMPONENTES EXISTENTES</b>	
Descripción	Modificar y extender componentes según sea necesario para usar los nuevos personajes 2D.
Importancia	Alta
Coste	Medio
Prioridad	Alta
Dependencias	T2

T4 - GENERAR NUEVAS MALLAS	
Descripción	Generar nuevas mallas para el terreno segmentadas por caras.
Importancia	Alta
Coste	Bajo
Prioridad	Alta
Dependencias	—
T5 - GENERAR TEXTURAS PARA LAS NUEVAS MALLAS	
Descripción	Crear varias texturas para cada grupo de mallas.
Importancia	Baja
Coste	Bajo
Prioridad	Baja
Dependencias	T4
T6 - IMPLEMENTAR FUNCIONALIDAD DE EDICIÓN POR CARAS	
Descripción	Implementación de una nueva funcionalidad que permita al usuario modificar cada objeto del terreno por caras.
Importancia	Alta
Coste	Medio
Prioridad	Media
Dependencias	T4 - T5
T7 - AMPLIAR ACCIONES DE LOS PERSONAJES	
Descripción	Crear nuevas acciones para los personajes que están en batalla, como habilidades especiales.
Importancia	Media
Coste	Alto
Prioridad	Baja
Dependencias	T2
T8 - REMODELAR INTERFAZ	
Descripción	Modificar la interfaz respecto a la versión anterior.
Importancia	Alta
Coste	Bajo
Prioridad	Baja
Dependencias	—

Tabla 4.4: Desglose de tareas para el editor de escenarios 2D.

#### 4.2.2. Tareas del editor de mapas

EDITOR DE MAPAS	
<b>T1 - GENERACIÓN DE RECURSOS GRÁFICOS</b>	
Descripción	Generar elementos gráficos como mapas y ciudades usados en este módulo.
Importancia	Baja
Coste	Bajo
Prioridad	Baja
Dependencias	—
<b>T2 - IMPLEMENTAR SISTEMA DE EDICIÓN DE MAPAS</b>	
Descripción	Generar el editor de mapas con ciudades y caminos, así como implementar la opción de añadir eventos.
Importancia	Alta
Coste	Alto
Prioridad	Baja
Dependencias	—
<b>T3 - IMPLEMENTAR MOVIMIENTO POR EL MAPA</b>	
Descripción	Implementar la funcionalidad necesaria para que el usuario pueda desplazarse por el mapa dentro del juego.
Importancia	Alta
Coste	Bajo
Prioridad	Baja
Dependencias	T2
<b>T4 - IMPLEMENTAR LÓGICA DE SALVADO Y CARGA DE DATOS</b>	
Descripción	Implementar el guardado y cargado de los mapas y los eventos dispuestos en él.
Importancia	Alta
Coste	Bajo
Prioridad	Baja
Dependencias	T2
<b>T5 - INTEGRACIÓN EN EL CORE DE TRPG MAKER</b>	
Descripción	Integrar el módulo en el núcleo del sistema.
Importancia	Alta
Coste	Bajo
Prioridad	Baja
Dependencias	Todas las anteriores

Tabla 4.5: Desglose de tareas para el editor de mapas.

#### 4.2.3. Tareas del gestor de colas

GESTOR DE COLAS	
<b>T1 - IMPLEMENTAR JERARQUÍA DE CLASES</b>	
Descripción	Implementar las clases necesarias para cada tipo de evento.
Importancia	Alta
Coste	Bajo
Prioridad	Alta
Dependencias	—
<b>T2 - IMPLEMENTAR EVENTOS DE MOVIMIENTO</b>	
Descripción	Generar la implementación de los eventos de movimiento.
Importancia	Alta
Coste	Medio
Prioridad	Alta
Dependencias	T1
<b>T3 - IMPLEMENTAR EVENTOS DE MODIFICACIÓN</b>	
Descripción	Generar la implementación de los eventos de modificación de inventario.
Importancia	Alta
Coste	Medio
Prioridad	Alta
Dependencias	T1
<b>T4 - IMPLEMENTAR EVENTOS DE CERROJO</b>	
Descripción	Generar la implementación de los eventos de cerrojo del mapa.
Importancia	Alta
Coste	Bajo
Prioridad	Alta
Dependencias	T1
<b>T5 - IMPLEMENTAR EVENTOS DE ANIMACIÓN</b>	
Descripción	Generar la implementación de los eventos de animación.
Importancia	Alta
Coste	Alto
Prioridad	Alta
Dependencias	T1

T6 - IMPLEMENTAR EVENTOS DE DIÁLOGO	
Descripción	Generar la implementación de los eventos de diálogo.
Importancia	Alta
Coste	Bajo
Prioridad	Baja
Dependencias	T1
T7 - IMPLEMENTAR EVENTOS DE BATALLA	
Descripción	Generar la implementación de los eventos de batalla del mapa.
Importancia	Alta
Coste	Bajo
Prioridad	Baja
Dependencias	T1

Tabla 4.6: Desglose de tareas para el gestor de colas.

### 4.3. Planificación temporal

Cada módulo del sistema ha sido desarrollado durante un mes, y para ello se ha seguido el siguiente esquema de desarrollo:

- **Conceptualización y diseño:** fase de prototipado donde se han diseñado prototipos de los módulos del sistema. Duración de 1 semana.
- **Implementación:** se implementan y prueban las implementaciones generadas en base a los diseños de la semana anterior. Duración de 2 semanas.
- **Generación de demo:** se integran las nuevas implementaciones en el núcleo de la herramienta y se comprueba su correcto funcionamiento. Duración de 1 semana.

A continuación se concretan las fechas del desarrollo y los artefactos creados:

#### 1. Creación del editor de niveles en 3D:

- Conceptualización y diseño: 02/03/2020 - 08/03/2020.
- Implementación: 09/03/2020 - 22/03/2020.
- Generación de demo: 23/03/2020 - 29/03/2020.

**2. Creación del editor de bases de datos:**

- Conceptualización y diseño: 30/03/2020 - 05/04/2020.
- Implementación: 06/04/2020 - 19/04/2020.
- Generación de demo: 20/04/2020 - 26/04/2020.

**3. Creación del editor de diálogos:**

- Conceptualización y diseño: 27/04/2020 - 03/05/2020.
- Implementación: 04/05/2020 - 17/05/2020.
- Generación de demo: 18/05/2020 - 24/05/2020.

**4. Creación del editor de niveles en 2D:**

- Conceptualización y diseño: 25/05/2020 - 31/05/2020.
- Implementación: 01/06/2020 - 14/06/2020.
- Generación de demo: 15/06/2020 - 21/06/2020.

**5. Creación del editor de mapas:**

- Conceptualización y diseño: 22/06/2020 - 28/06/2020.
- Implementación: 29/06/2020 - 12/07/2020.
- Generación de demo: 13/07/2020 - 19/07/2020.

**6. Creación del gestor de colas de eventos:**

- Conceptualización y diseño: 20/07/2020 - 26/07/2020.
- Implementación: 27/06/2020 - 09/08/2020.
- Generación de demo: 10/08/2020 - 16/08/2020.

# Capítulo 5

## Desarrollo del sistema

**Resumen:** En este capítulo se presentan los requisitos técnicos de *TRPG Maker*, se expone su arquitectura interna y se explican cada uno de los módulos que la componen en detalle.

### 5.1. Arquitectura del sistema

*TRPG Maker* es una herramienta para el desarrollo y diseño de videojuegos, enfocada en los usuarios finales sin conocimientos técnicos. Esta herramienta se centra en videojuegos del género TRPG, es decir, videojuegos de rol táctico que se caracterizan por tener una serie personajes que suben de nivel y se equipan con todo tipo de objetos, los cuales siguen una historia predeterminada que puede tomar distintos caminos según las acciones llevadas a cabo por el jugador.

Aunque este tipo de títulos tengan unas bases comunes la mayoría de ellos tienen ciertas características que los diferencian entre sí, como personajes en 3D o en 2D, sistemas de habilidades o el apartado gráfico. En *TRPG Maker* se ha buscado cubrir la mayoría de características existentes en este tipo de videojuegos, y por ello se ha optado por crear un sistema como el que se muestra en la figura 5.1.

*TRPG Maker* consta de 5 módulos independientes entre sí:

- **Editor de niveles:** este módulo es el principal de la herramienta. Con él se crean los escenarios del juego y permiten ser probados en cualquier momento. Además, permite la creación de dos tipos de escenarios:
  1. Escenarios con entornos tridimensionales junto con personajes y objetos que poseen modelos 3D.

2. Escenarios con perspectiva isométrica y personajes con *sprites* 2D.
- **Editor de bases de datos:** TRPGMaker posee una base de datos con todo el contenido disponible a la hora de crear elementos de juego, desde personajes y clases hasta habilidades y objetos que pueden ser configuradas por el usuario.
  - **Editor de mapas:** este módulo permite editar los mapas del juego, añadiendo localizaciones o eventos con los que el usuario puede interactuar.
  - **Editor de conversaciones:** este módulo permite la creación de diálogos entre los personajes del juego, los cuales pueden añadirse como eventos bien en los mapas o durante las batallas.
  - **Editor de colas de eventos:** este módulo permite al jugador administrar los sucesos que tienen lugar en el juego, como conversaciones o batallas.

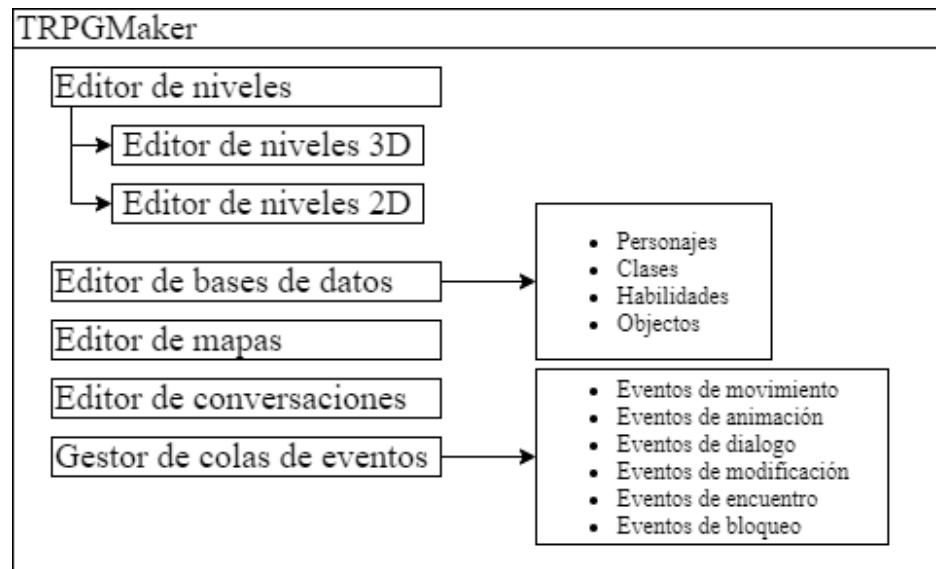


Figura 5.1: Módulos que integran la herramienta *TRPG Maker*.

### 5.1.1. Editor de niveles

El editor de niveles es el principal módulo de *TRPG Maker*, pues permite a los usuarios crear los escenarios que formarán parte de su videojuego. Este módulo, al igual que los demás sigue las directrices comentadas en el apartado 3.1.3 en relación a las clases indispensable dentro de un proyecto creado con Unreal Engine 4.

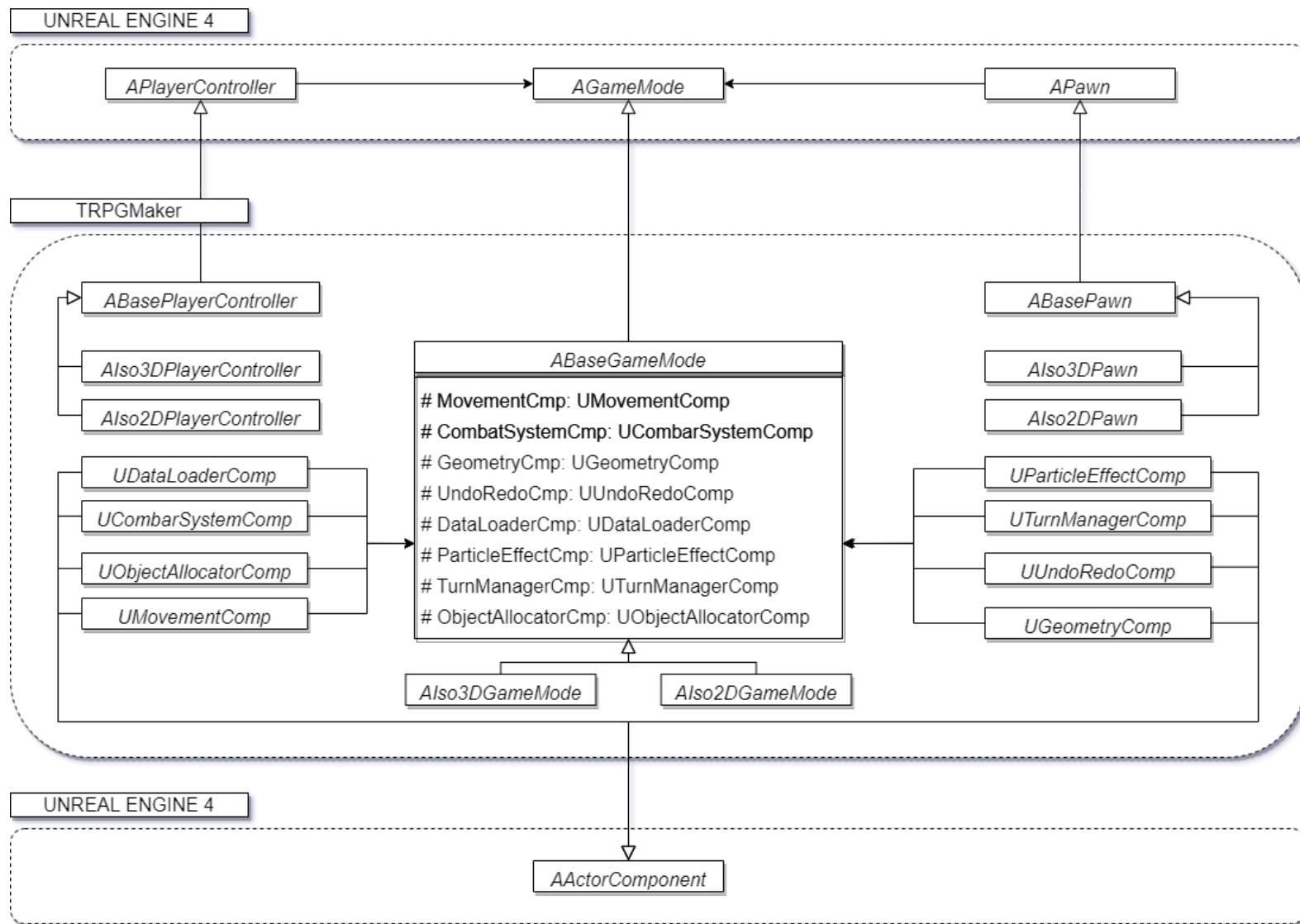
En la figura 5.2 tenemos el diagrama de clases correspondiente a parte de la lógica que implementa este módulo. Al poder crear dos tipos de escenarios, son necesarios dos *GameModes* distintos, *AIso3DGameMode* para los escenarios en 3D y *AIso2DGameMode* para los escenarios en vista isométrica con personajes y objetos en 2D.

Al tener dos *GameModes* distintos, necesitamos a su vez un *PlayerController* y un *Pawn* específicos para cada caso. En el caso de los escenarios en 3D tenemos las clases *AIso3DPlayerController* y *AIso3DPawn*, mientras que en el caso de los escenarios con vista isométrica usaremos las clases *AIso2DPlayerController* y *AIso2DPawn*. Ambos pares de clases tienen diferentes funcionalidades en cada caso, por ejemplo las rotaciones de la cámara del jugador, que son gestionadas por las clases que heredan de *ABasePawn*, poseen diferentes movimientos de cámara en función del tipo de escenario.

La clase *ABaseGameMode*, además de las instancias de *APlayerController* y *APawn*, posee una serie de objetos que se encargan de gestionar la mayor parte de la lógica de la herramienta. Se trata de clases que siguen el patrón de diseño *Singleton* y heredan de la clase *AActorComponent* de Unreal Engine 4.

En la figura 5.3 tenemos tenemos el diagrama de clases que corresponde a los objetos que pueden ser colocados en el escenario. La clase base es *ARPGEntity* que hereda de la clase *AActor* de un Unreal Engine 4. A su vez, tenemos otras 3 clases hijas:

- ***ARPGObject***: Corresponde a los objetos del propio juego que pueden encontrarse en el mapa y con los que se puede interactuar, como por ejemplo cofres o pociones, dos de los elementos típicos de los videojuegos del género TRPG.
- ***AIsoTile***: Esta clase define el terreno que puede ser colocado en el escenario. Dentro de esta categoría tenemos dos posibles tipos de objetos:
  1. ***AIso3DTile***: Estos objetos son exclusivos de los escenarios en 3D y están formados por una sola malla.

Figura 5.2: Diagrama de clases del módulo editor de niveles de *TRPG Maker*.

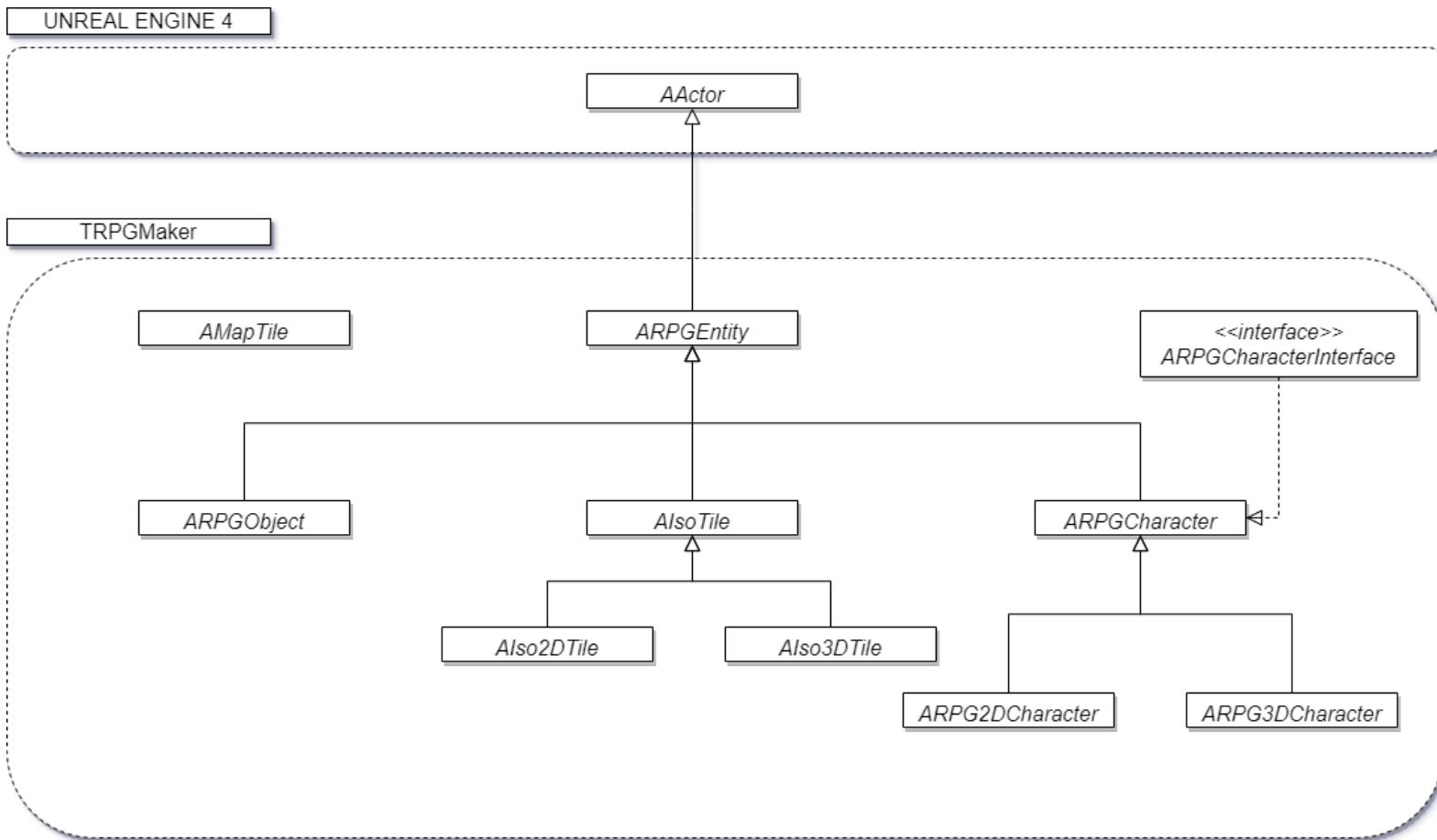


Figura 5.3: Diagrama de clases que corresponde a los objetos del escenario, en este caso bloques, personajes y objetos.

2. ***AIso2DTile***: Estos objetos están formados por varias mallas, en función del tipo de bloque que sean. Por ejemplo, los bloques y medios bloques están formados por 3 mallas (cara superior y laterales), mientras que las rampas y las medias rampas solo están formados por dos (cara superior y un lateral, en función de la orientación de la rampa).
- ***ARPGCharacter***: Esta clase define la estructura básica de los personajes. Para ello, posee una serie de atributos como salud o defensa, además de la lista de habilidades y objetos que posee el personaje. A su vez, esta clase deriva en otras dos en función del tipo de escenario:
    1. ***ARPG2DCharacter***: Para escenarios con vista isométrica. Los personajes que pertenecen a esta clase poseen un componente *UPaperFlipbook* que contiene el *sprite* 2D que representa el personaje.
    2. ***ARPG3DCharacter***: Para escenarios en 3D. En este caso el personaje posee una malla con el modelado en tres dimensiones del personaje.

La clase *ARPGCharacter* implementa la interfaz *ARPGCharacterInterface* que le provee de los métodos encargados de las animaciones del personaje. Además, posee un vector director usado para las rotaciones del personaje y un *AAIController*, una instancia de *APlayerController* que se gestiona la IA del personaje.

Tanto la clase *APRGCharacter* como todas sus hijas implementan una serie de métodos de la clase *AActor* que permiten al usuario interactuar con el entorno. Estos métodos están representados en la figura 5.4.

---

```

1 // gestiona el click del ratón sobre el personaje
2 UFUNCTION()
3     void onClicked(AActor * Target, FKey ButtonPressed);
4 // gestiona la entrada del cursor sobre el personaje
5 UFUNCTION()
6     void onBeginCursorOver(AActor* Target);
7 // gestiona la salida del cursor sobre el personaje
8 UFUNCTION()
9     void onEndCursorOver(AActor* Target);

```

---

Figura 5.4: Eventos para la gestión de *inputs* del ratón sobre un objeto.

### 5.1.1.1. Clase GeometryComp

Esta clase se encarga de almacenar toda la lógica del editor de niveles, y para ello dispone de una serie de variables mostradas en la figura 5.5.

---

```

1  private:
2
3      // bloques del escenario
4      UPROPERTY()
5          TMap<FIntVector, AIsoTile> Tiles;
6
7      // personajes del escenario
8      UPROPERTY()
9          TMap<FIntVector, ARPGCharacter> Characters;
10
11     // objetos del escenario
12     UPROPERTY()
13         TMap<FIntVector, ARPGObject> Objects;
14
15     // máximas alturas en el mapa
16     UPROPERTY()
17         TMap<F2DVector, int> MaxHeights;
18
19     // tipo de objeto por máxima altura
20     UPROPERTY()
21         TMap<FIntVector, EEntityType> EntityType;

```

---

Figura 5.5: Variables que mantienen la lógica de un escenario.

En *TRPG Maker* los escenarios son tratados como mapas en tres dimensiones, de forma que cada objeto ocupa una posición en dicho mapa. Cada posición es representada con un vector de tres enteros, lo cual nos permite identificar cada objeto en un espacio 3D al tener tres coordenadas. Por ello, tenemos 3 diccionarios que almacenan la información de cada posición del mapa:

- ***Tiles***: almacena los bloques del escenario.
- ***Characters***: almacena los personajes del escenario.
- ***Objects***: almacena los objetos del escenario.

Junto a esta información también disponemos de otras dos variables extra, *MaxHeights* y *EntityType*. *MaxHeights* nos indica, dadas unas coordenadas *X* e *Y*, la altura a la que se encuentra el último objeto en dichas coordenadas. Por ejemplo, si tenemos una columna de tres bloques en la coordenada (*X*=3, *Y*=6) y sobre el último bloque hay un personaje, *MaxHeights* almacenará el valor 4 para los pares de valores *X*=3 e *Y*=6. A su vez, *EntityType* nos almacenará el tipo de objeto que es cada elemento identado en *MaxHeights*, en el caso anterior sería un *ECharacter*.

Esta información a pesar de parecer simple, nos permite hacer cálculos más rápido durante la generación de caminos óptimos al mover personajes, haciendo accesos directos sobre estos diccionarios.

#### 5.1.1.2. Clase MovementComp

Esta clase se encarga de gestionar el movimiento de los personajes por el mapa. Para ello, utiliza el algoritmo de *Dijkstra* ([Dijkstra et al. 1959](#)) junto con el objeto *GeometryComp* para determinar por donde ha de pasar el personaje para llegar a su destino.

El algoritmo de *Dijkstra* utiliza un mecanismo del motor Unreal Engine 4, las trazas (figura 5.6), para el descubrimiento de vecinos. Este mecanismo consiste en trazar un rayo desde una posición a otra para ver que objetos hay en medio, de forma que si hay un objeto cuyos parámetros de colisión están configurados para bloquear el rayo el motor nos devuelve información sobre la colisión, entre ella el objeto con el que ha colisionado y su ubicación.

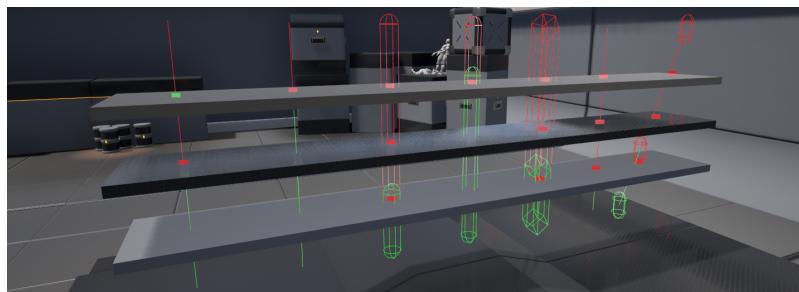


Figura 5.6: Ejemplo de trazas en Unreal Engine 4. El color rojo indica que el trazo no ha colisionado, mientras que el verde indica que ha habido colisión con un objeto con un perfil de colisión bloqueante.

Este mecanismo es costoso, y por ello solo se usa al principio del algoritmo para ver si el personaje puede moverse, de forma que en el resto de casos se usan las variables comentadas anteriormente para obtener los objetos del

escenario de forma más eficiente.

Una vez se ha ejecutado el algoritmo obtenemos un *array* con las posiciones en forma de vectores de tres elementos de tipo entero y pasamos a generar el camino. Para ello, usamos un componente de Unreal Engine 4 llamado *USplineComponent*. Los objetos de este tipo determinan una secuencia de posiciones por los que ha de pasar el personaje, la cual se obtiene interpolando los elementos del *array* obtenido del algoritmo de *Dijkstra*.

### 5.1.1.3. Clase UndoRedoComp

Esta clase implementa las funcionalidades de *hacer* y *deshacer* propias de este tipo de herramientas. Para ello hacemos uso del patrón de diseño *Command*. Este patrón de diseño permite ejecutar una operación de un objeto sin saber que hace esta operación. Para ello se encapsula el objeto en cuestión y se implementan una serie de métodos que permiten hacer y deshacer operaciones.

La implementación del patrón *Command* se muestra en la figura 5.7. Tenemos la clase *UBaseCommand* que define los métodos *ExecuteCommand* y *UnExecuteCommand* que serán implementados por las clases hijas.

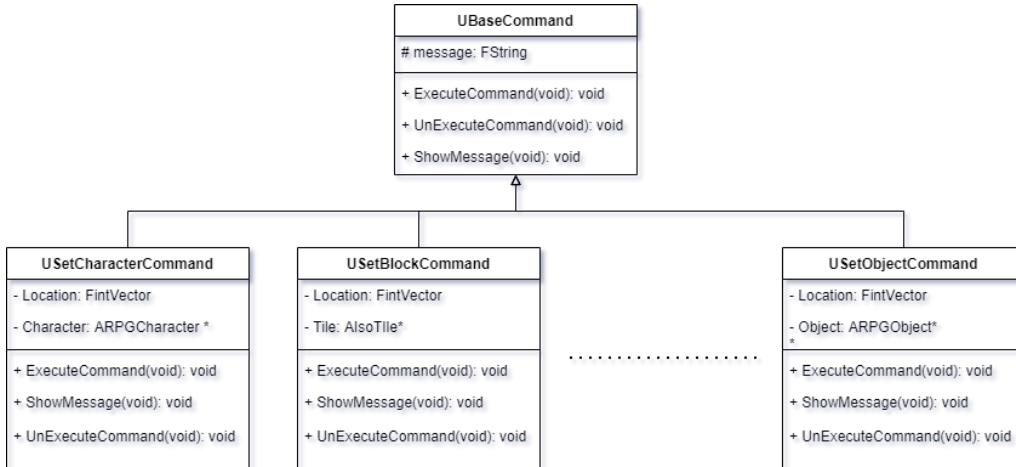


Figura 5.7: Diagrama de clases para el patrón *Command* implementado.

Este componente tiene dos pilas que almacenan objetos del tipo *UBaseCommand*. En una se almacena un objeto cada vez que sea necesario, por ejemplo al colocar un personaje o un objeto en el mapa. Cuando el usuario lleve a cabo una acción de *deshacer* se sacará un elemento de esta pila, se ejecutará su implementación y se introducirá en la otra pila, por si el usuario quiere volver a deshacer esta acción.

#### 5.1.1.4. Clase DataLoaderComp

Este componente se encarga de gestionar la carga y descarga de datos en *TRPG Maker*. En concreto, se llevan a cabo una serie de procedimientos en momentos específicos de la ejecución de la herramienta:

- **Al inicio:** Se cargan todos los datos pertinentes de forma síncrona a excepción de la información relevante para el editor de la base de datos, la cual es cargada de forma asíncrona al ser más pesada.
- **Al final:** Al cerrar la aplicación por algunas de las vías disponibles, bien directamente desde la propia aplicación o por acciones externas, como la combinación de teclas *alt + F4*. En esta situación se salva la información relevante a la base de datos por si hubiese habido algún cambio pendiente y se salva la conversación, el mapa o el escenario en caso de estar editándose en el momento del cierre de la herramienta.
- **Bajo demanda:** Estos procedimientos son requeridos de forma puntual por el usuario, por ejemplo cuando quiere guardar el progreso de un escenario o los cambios realizados en la base de datos.

Toda la información de la herramienta se ha almacenado usando ficheros JSON ya que el motor nos proporciona mecanismos de serialización para este tipo de ficheros, simplificando en gran medida la carga y descarga de datos.

#### 5.1.1.5. Clase ObjectAllocatorComp

Este componente se encargad de colocar los objetos en el mapa. Para ello, implementa un método de la clase *AActorComponent* con el siguiente encabezado:

---

```

1 public:
2   // Called every frame
3   virtual void TickComponent(float DeltaTime, ELevelTick TickType,
4     FActorComponentTickFunction* ThisTickFunction) override;
```

---

Figura 5.8: Cabecera del evento *TickComponent* implementado por el componente *ObjectAllocatorComp*.

Este método es llamado cada tick de reloj, aproximadamente cada 0.016 segundos, por lo tanto se ejecuta constantemente. En dicho método se calcula la posición en el mapa del cursor, y se usa un trazado en dicha posición para determinar sobre qué objeto está el cursor.

### 5.1.2. Acciones y gestión del editor de niveles

El editor de niveles, además de poseer toda la lógica comentada, también dispone de una interfaz gráfica que permite al usuario interactuar con la aplicación. Para *TRPG Maker* hemos optado por un sistema *point and click* de forma que el usuario selecciona las acciones a realizar en la interfaz gráfica e interacciona con el escenario haciendo *click*. Estas acciones son:

- **Colocar bloque:** coloca un bloque en el escenario.
- **Colocar personaje:** coloca un personaje en el escenario.
- **Colocar objeto:** coloca un objeto en el escenario.
- **Borrar:** elimina un elemento del escenario.
- **Limpiar escenario:** elimina todos los elementos del escenario.
- **Añadir diálogo:** permite asignar un evento de conversación al inicio de la batalla.

En función del tipo de escenario que estemos creando podremos llevar a cabo una serie de acciones específicas. En el caso de los escenarios tridimensionales es posible aplicar rotaciones a los objetos del mapa, mientras que en los escenarios con vista isométrica es posible editar los bloques por caras (figura 5.9). Ambos editores también permiten guardar el estado del escenario y cargar un estado anterior del mismo desde un fichero.

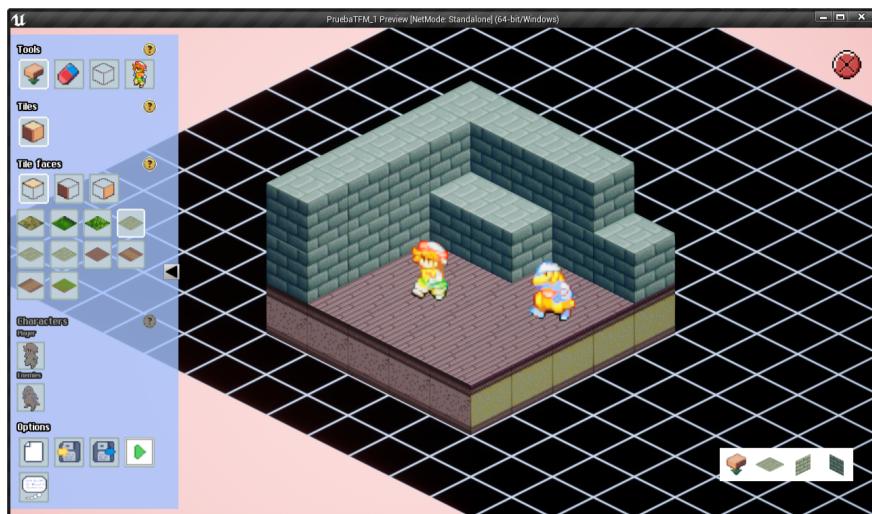


Figura 5.9: Ejemplo de escenario con vista isométrica y personajes 2D de *TRPG Maker*.

El editor de niveles también permite al usuario probar los niveles que ha creado, y para ello hacemos uso de una máquina de estados como la mostrada en la figura 5.10.

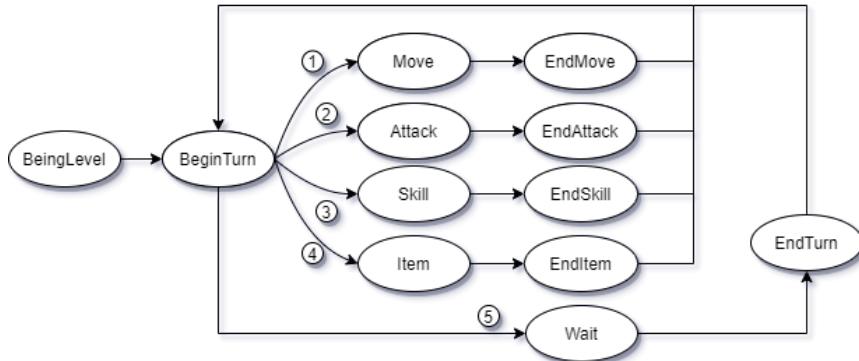


Figura 5.10: Máquina de estados que gestiona las batallas del editor de niveles.

Los videojuegos del género TRPG permiten a cada personaje realizar una serie de acciones en un turno, que por regla general son:

- **Mover** la unidad: mueve la unidad por el mapa.
- **Atacar**: realiza un ataque a otro personaje en el mapa.
- **Habilidad**: usa una habilidad normalmente asociada al propio personaje o a su clase/trabajo.
- **Objeto**: usa un objeto.
- **Esperar**: pasa el turno.

Cada acción ha sido plasmada en la máquina de estados con un estado cuyas transiciones hemos numerado del 1 al 5. El punto de inicio es el estado `BeginLevel`, donde se inicializa la batalla. Esta inicialización consiste en llenar la cola de prioridad de personajes, seleccionar la unidad activa y cargar los elementos de interfaz gráfica oportunos para el transcurso del combate.

Los personajes tienen una serie de variables de control que permiten determinar qué acciones han realizado en combate. Para ello, disponemos de dos variables, una que indica si el jugador ya se ha movido y otra que indica si ya ha realizado una acción, es decir, si ha atacado, ha usado una habilidad o un objeto.

Las transiciones 1-5 desde el estado *BeginTurn* son activadas mediante una serie de botones de la interfaz gráfica y dan comienzo a las acciones del jugador. Cualquiera de estas transiciones desencadena una serie de eventos en el *GameMode*. Por ejemplo, en el caso de transicionar al estado *Move* el *GameMode* usará el objeto *MovementCmp* para calcular los posibles caminos del personaje y luego lo moverá por el mapa. Además, cualquiera de estas acciones hará que las variables de control del personajes se desactiven, por lo tanto tras haber realizado la acción de Mover el personaje no podrá volver a realizarla.

Una vez el personaje se haya movido y haya realizado alguna acción, solo se podrá transicionar al estado *Wait*. Este estado se encarga de modificar la interfaz gráfica y de restaurar las variables de control del personaje a su estado original de cara a futuros turnos. Este estado transiciona directamente al nodo *EndTurn*, el cual se encarga de modificar la cola de personajes y de seleccionar la siguiente unidad activa, tareas llevadas a cabo por el objeto *TurnManagerCmp*.

### 5.1.3. Editor de bases de datos

Este módulo permite al usuario modificar los elementos que pueden ser usados en los escenarios. Para ello se ha optado por añadir un sistema de programación con texto (sección 2.1.3) al igual que el de otras herramientas como *RPG Maker*. En concreto, permite modificar:

- **Personajes:** es posible modificar sus atributos base y los crecimientos de los mismos, es decir, los incrementos de los atributos al subir de nivel. Además, es posible seleccionar la clase del personaje y modificar la cantidad y tipo de objetos que posee.
- **Objetos:** son elementos del juego que pueden aparecer en los niveles y pueden ser usados por los personajes tanto dentro como fuera de la batalla. Los objetos poseen una serie de propiedades como la cantidad y el tipo de objetivos a los que afecta, el tipo de efecto que provoca o el coste del mismo, las cuales pueden ser editadas en este módulo.
- **Clases:** las clases dotan a los personajes de una serie de estadísticas base y un conjunto de habilidades, de forma que es posible modificarlas en este módulo.

- **Habilidades:** corresponden a los ataques especiales usados por los personajes. Las habilidades tienen asociado un coste al ser usadas, además de otras propiedades como el la cantidad y tipo de objetivos a los que afecta al igual que los objetos o el efecto de partículas que se lanza como retroalimentación visual para el usuario. Normalmente este tipo de ataques en los videojuegos del género TRPGMaker llevan asociada un área y un rango de actuación. Este área corresponde a un conjunto de casillas del mapa sobre las que se puede lanzar la habilidad, mientras que el rango corresponde al conjunto de casillas afectadas por la habilidad tras ser lanzada (figura 5.11).



Figura 5.11: Ejemplo de habilidad en *TRPG Maker* es un escenario con vista isométrica. Las casillas azules representan el área de la habilidad (casillas donde puede ser usada) mientras que las casillas verdes representan el rango (casillas donde afectará la habilidad).

#### 5.1.4. Editor de mapas

Los mapas son el elemento principal de los videojuegos del género TRPG donde tiene lugar la mayor parte de las interacciones. Por regla general, disponen de una serie de localizaciones conectadas entre sí con las cuales el usuario puede interactuar (figura 5.12). Estas localizaciones varían de un título a otro, aunque suelen ser escenarios de batalla, tabernas donde comerciar, ciudades u otro tipo de evento.



Figura 5.12: Mapa del mundo del videojuego *Final Fantasy Tactics Advance*.

Para adaptar estas características en *TRPG Maker* se han implementado los mapas como un grafo, de forma que los nodos son combates o diálogos mientras que las aristas del grafo son los caminos que unen las localizaciones entre sí. Además, utilizamos el algoritmo de *Dijkstra* para el cálculo de caminos óptimos para mover al jugador una localización a otra.

### 5.1.5. Editor de conversaciones

En *TRPG Maker* es posible crear conversaciones que pueden seguir varios caminos en función de las respuestas del usuario. Además, es posible añadir ciertos eventos que modifican elementos de los personajes, como sus objetos. Además las conversaciones pueden tener bifurcaciones, es decir, elecciones por parte del usuario que determinan como prosigue la propia conversación. En la figura 5.13 tenemos un ejemplo de conversación entre dos personajes con un bifurcación, de forma que al elegir la opción *No, soy de los bueno* se seguirá por el diálogo con identificador 1.1 mientras que en caso de elegir la opción *Vaya, me has pillado!* la conversación continuará con el diálogo con identificador 2.1.

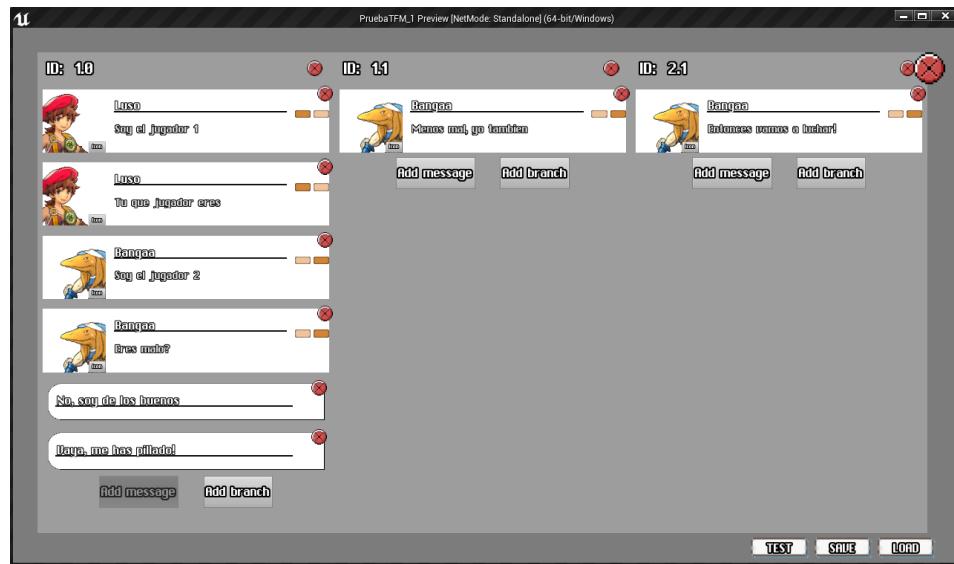


Figura 5.13: Editor de conversaciones de *TRPG Maker*. Los mensajes de los diálogos corresponden a las etiquetas con imágenes, mientras que las demás corresponden a las bifurcaciones.

Las conversaciones son tratadas como un árbol y constan de una secuencia de mensajes, denominada diálogo, entre los personajes del juego. Los diálogos pueden terminar si no se introducen más mensajes o si se introduce una bifurcación, las cuales dan lugar a tantos nuevos diálogos como posibles bifurcaciones haya.

#### 5.1.6. Gestor de colas de eventos

El gestor de colas de evento permite al usuario definir los eventos que tendrán lugar dentro del juego. Para ello, disponemos de una serie de eventos:

- **Evento de movimiento:** desplaza un personaje de un lugar a otro del mapa.
- **Evento de animación:** ejecuta una animación de un personaje.
- **Evento de diálogo:** muestra una conversación.
- **Evento de modificación:** modifica, añade o elimina un objeto del inventario de un personaje.
- **Evento de encuentro:** genera una batalla en el mapa del juego.
- **Evento de bloqueo:** bloque o desbloquea ciertos nodos del mapa para restringir la movilidad del jugador.

Y de dos tipos de colas de eventos:

- **Cola de eventos de batalla:** cada escenario posee una, y puede contener eventos de movimiento, animación, diálogo y modificación.
- **Cola de eventos de mapa:** solo hay una cola de este tipo y almacena los eventos del mapa. Estos eventos pueden ser de diálogo, de modificación, de encuentro o de bloqueo

En la figura 5.14 tenemos el diagrama de clases de los eventos disponibles en el gestor de eventos. Para su implementación se ha usado el patrón de diseño *Command*.

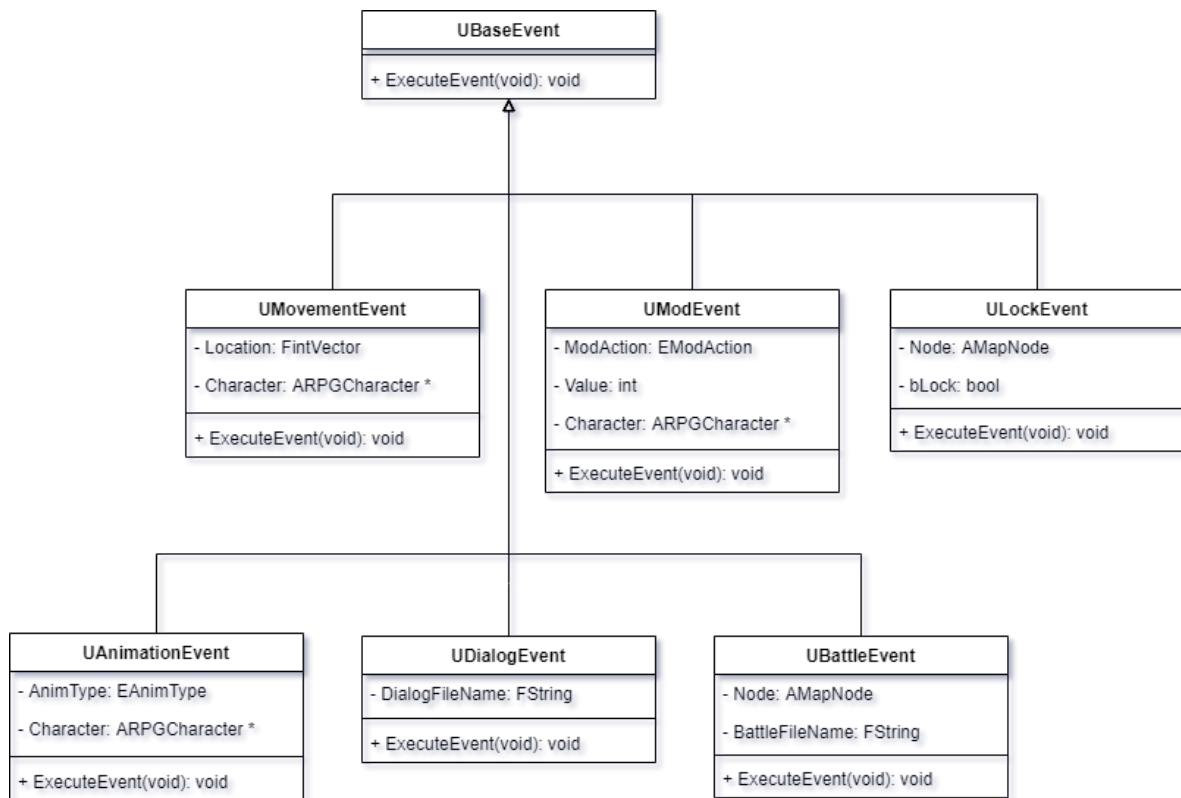


Figura 5.14: Diagrama de clases de los eventos.



# Capítulo 6

## Pruebas y resultados

**Resumen:** Este capítulo expone las pruebas llevadas a cabo para evaluar el desempeño del sistema, tanto los resultados de la encuesta y la realimentación obtenida por parte de usuarios reales, como comparativas con herramientas similares.

### 6.1. Pruebas con usuarios reales

Para probar la utilidad del prototipo y empezar a detectar errores llevamos a cabo una sesión de experimentación en una fase temprana del desarrollo tras la finalización del primer hito, en la que únicamente se probó el editor de escenarios. Como se buscaba llegar a un público general, lanzamos un concurso en colaboración con Electronic Arts donde el objetivo era crear un nivel con la herramienta haciendo uso de un paquete de personajes y texturas ya integrado en la propia herramienta. Para este concurso se creó una versión específica y limitada de la herramienta llamada *TRPG Maker: The Good Doctors* con un paquete de *assets* especial con sanitarios y virus enfocada a concienciar sobre la actual pandemia de COVID-19 y el virus SARS-CoV-2 entre un público muy joven, niños de entre 8 y 12 años.

Para este experimento nos decantamos por pruebas de usabilidad a distancia, de tal forma que no es necesario que los usuario y los jueces evaluadores tengan que estar en el mismo lugar para que sean llevadas a cabo. En este experimento nos centramos en medir la eficacia, la eficiencia y la satisfacción al utilizar este prototipo de la herramienta. Para ello, recolectamos una serie de datos, como el tiempo de uso de la herramienta o el número de bloques/personajes utilizados en el editor, de forma automática.

Esta información se complementó con un cuestionario de evaluación con:

- 5 preguntas relacionadas con la usabilidad de la herramienta, haciendo especial énfasis en la facilidad de uso y la facilidad de aprendizaje en comparación con otras herramientas similares.
- 2 preguntas relacionadas con el uso de la herramienta a nivel comercial y en el ámbito educativo.
- Otras preguntas de texto libre en relación a los controles de la cámara, la disposición del menú y posibles funcionalidades que podrían añadirse en el futuro a nuestra herramienta.

En el experimento participaron 10 personas, aunque el perfil predominante (7 de 10) resultó ser de *gamedev*. Esta situación hizo que el rumbo del proyecto cambiase drásticamente debido a la alta participación de usuarios con este tipo de perfil (y dificultad para encontrar usuarios menores de edad), de forma que a mitad del desarrollo pasamos de hacer una herramienta enfocada en un público infantil a una herramienta de carácter profesional enfocada hacia usuarios interesados en el desarrollo de videojuegos independiente.

Debido a estos cambios, al finalizar el segundo hito se llevó a cabo una evaluación del sistema final comparándolo con otras herramientas de desarrollo de videojuegos disponibles en el mercado actual, cuyos resultados son expuestos en el siguiente apartado.

A pesar de no haber alcanzado el público objetivo con esta sesión de experimentación merece la pena destacar varios puntos interesantes tras haber procesado los resultados obtenidos, aunque sea para hacer una valoración cualitativa de los mismos:

- Solo el 10% de los usuarios no está de acuerdo con la estética de la aplicación como se puede comprobar en la figura 6.1.

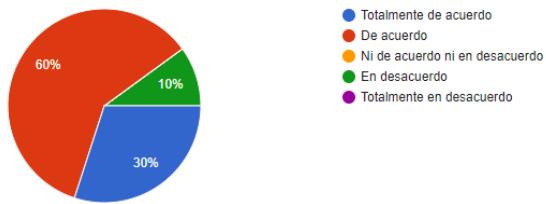


Figura 6.1: Resumen de las respuestas obtenidas en relación a la pregunta “*Me gusta la experiencia de uso de la aplicación y su estética*”.

- El 30 % de los usuarios ha tenido problemas con los controles de la herramienta (figura 6.2). Esto hace necesario una revisión de esta funcionalidad, la cual debe ser modificada para ofrecer una mejor experiencia de usuario.

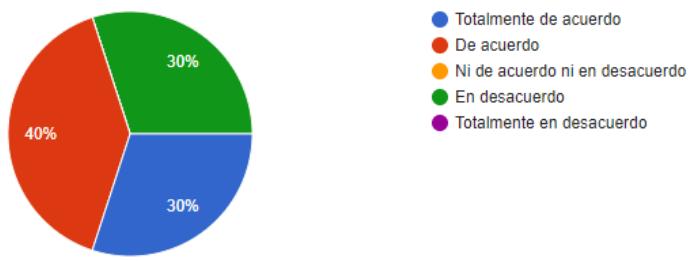


Figura 6.2: Resumen de las respuestas obtenidas en relación a la pregunta “*Me gusta el manejo de la aplicación y sus controles (movimiento, botones, cámara, etc.).*”.

- En respuesta a la cuestión “*Indica cómo crees que podría mejorarse la aplicación (experiencia de uso y estética, manejo y controles, gestión de errores, etc)*” varios usuarios relatan que han tenidos problemas relacionados con el rendimiento del sistema, por lo tanto es necesario utilizar herramientas de *debugging* o algún *profiler* para detectar este tipo de errores.
- En respuesta a la cuestión “*Menciona esas otras aplicaciones si las hay, e indica qué nuevas características te gustaría que se añadiesen a esta aplicación que has probado*” muchos usuarios mencionan la interfaz del juego *Super Mario Maker 2*, por lo tanto sería interesante introducir algunas de las funcionalidades de este título en *TRPG Maker* e incluso tomar como referencia ciertos elementos de la interfaz su gráfica.
- En respuesta a la cuestión “*Recomendaría el uso de esta aplicación con fines educativos*” el 90 % de los usuarios avala el uso de la herramienta en el entorno educativo (figura 6.3). Este es sin duda el resultado más increíble de la experimentación con usuario, y abre nuevas vías de cara al trabajo futuro de la herramienta.

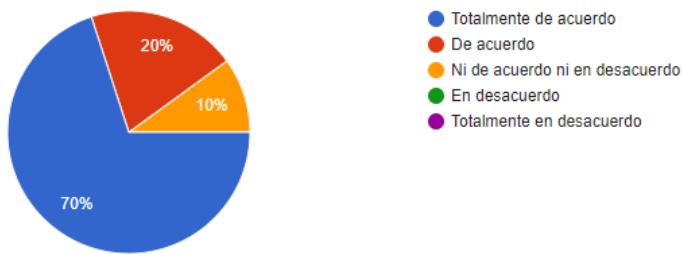


Figura 6.3: Resumen de las respuestas obtenidas en relación a la pregunta “*Recomendaría el uso de esta aplicación con fines educativos*”.

## 6.2. Comparativa con otras herramientas

Debido a los resultados obtenidos durante la experimentación con usuarios fue necesario enfocar el proyecto hacia otro público objetivo distinto. Para evaluar el resultado final del desarrollo hemos comparado las funcionalidades implementadas en *TRPG Maker* con las de otras herramientas existentes en el mercado las cuales pueden ser usadas por el usuario final para desarrollar videojuegos. En la tabla 6.1 tenemos la comparativa.

COMPARATIVA DE TRPG MAKER CON OTRAS HERRAMIENTAS						
Sistemas de programación para el usuario final	App Inventor	Scratch	RPG Maker	GameMaker: Studio	Construct	TRPG Maker
Con <i>Inputs</i> de texto			X			X
Con bloques	X	X				
Con editor <i>drag and drop</i>			X	X	X	X
Edición de personajes			X	X	X	X
Edición de niveles			X			X
Programación con código			X	X	X	
Gestión de proyectos	X	X	X	X	X	X
Ejecución integrada de juegos	X	X	X	X	X	X

Tabla 6.1: Comparativa entre TRPG Maker y otras herramientas en el mercado.

*TRPG Maker* permite al usuario modificar todo el contenido de su juego y crear conversaciones mediante un sistema de *inputs* de texto, y además incorpora un sistema de edición de niveles *drag and drop* al igual que otras herramientas como *RPG Maker*, *GameMaker: Studio* o *Construct*. También permite la edición de personajes y niveles, aunque no soporta la ampliación de funcionalidades vía código.

### 6.3. Artículo de investigación

Como parte de este proyecto se ha realizado un artículo científico que ha sido aceptado en la sexta edición del Congreso de la Sociedad Española para las Ciencias del Videojuego (COSECiVi 2020)<sup>1</sup>. El artículo lleva el siguiente título: *Development of a User-Friendly Application for Creating Tactical Role-Playing Games* y será presentado de forma remota entre los días 7 y 8 de octubre de 2020.

---

<sup>1</sup><https://gaia.fdi.ucm.es/sites/cosecivi20/>



# Capítulo 7

## Conclusiones

**Resumen:** En este capítulo se resumen las conclusiones del proyecto. Además, se establecen varias propuestas de trabajo futuro extraídas de la realimentación de los usuarios.

En este proyecto se ha ahondado en el desarrollo para el usuario final y en las implicaciones de este tipo de herramientas en la actualidad del desarrollo de videojuegos. Además, con la implementación de *TRPG Maker*, una herramienta para la creación de videojuegos de rol táctico hemos alcanzado con notable éxito los principales objetivos de este proyecto, que describimos a continuación:

- **Objetivo 1:** se ha llevado a cabo un estudio en profundidad del estado del arte sobre el desarrollo para el usuario final y se han estudiado varias herramientas, tanto profesionales como relacionadas con el ámbito de la educación, para el desarrollo de videojuegos pensado para el usuario final. Por un lado, el estudio relacionado con las herramientas educativas de tipo infantil ha sido aplicado durante el primer hito para el desarrollo del prototipo *TRPG Maker: The Good Doctors* mientras que el estudio de las herramientas de carácter más profesional se ha tenido en cuenta especialmente durante el segundo hito, de cara al desarrollo de una versión más completa de la herramienta enfocada hacia desarrolladores independientes.
- **Objetivo 2:** el proyecto se enfocó inicialmente hacia un público concreto, definido como niños de entre 8 y 12 años, con el objetivo de estudiar las posibilidades de un sistema de estas características en el ámbito educativo. Además se ha llevado a cabo una sesión de experimentación completamente abierta, con usuarios reales, haciendo especial hincapié en la información cualitativa obtenida del mismo en relación a simplicidad de uso del sistema.

- **Objetivo 3:** se ha llevado a cabo una mejora del prototipo original haciendo uso de la realimentación de la experimentación anterior. Además se han implementado diversas formas de programación para el usuario final (programación con entradas de texto y sistemas de programación *point and click*) acordes a la herramienta y en función de las necesidades de la misma a la hora de desarrollar videojuegos, tomando como referencia las herramientas profesionales estudiadas con anterioridad.
- **Objetivo 4:** para finalizar el desarrollo del proyecto se ha llevado una comparación de características tomando como base otras herramientas comerciales existentes en el mercado, a modo de validación de la primera versión funcional de *TRPG Maker*.

La realización de esta serie de actividades ha finalizado con la publicación de la herramienta en itch.io<sup>1</sup> de forma que cualquier usuario pueda acceder a ella de forma gratuita.

Como parte de las conclusiones hemos podido observar que la herramienta todavía necesita bastante trabajo para poder ser un producto completo, aún habiendo integrado el *feedback* proporcionado por los usuarios. La mayoría de comentarios destacaban fallos en los controles, fallos al guardar los niveles y en algunos casos se han llegado a reportar cierres de la aplicación. Otro punto a mejorar es la cantidad de contenido disponible, tanto en lo que se refiere a personajes y objetos del juego como mecánicas nuevas que pueden ser usadas en el desarrollo de los niveles.

También hay que matizar que parte de los objetivos de este proyecto contemplaban la implementación de un editor de programación visual con el que los usuarios pudieran gestionar las colas de eventos de la aplicación, funcionalidad que no ha llegado a ser implementada debido a la falta de tiempo y al elevado coste de la propia implementación.

Aún así, *TRPG Maker* ha tenido gran aceptación entre los usuarios y la mayoría de ellos han visto en ella una herramienta útil en el ámbito educativo. Como posible trabajo futuro nos planteamos seguir mejorando la herramienta con un nuevo enfoque centrado en el ámbito educativo como herramienta complementaria en clases de diseño y desarrollo de videojuegos.

---

<sup>1</sup><https://narratech.itch.io/trpg-maker>

# Bibliografía

- ABRAHAM, R. y ERWIG, M. Autotest: A tool for automatic test case generation in spreadsheets. En *Visual Languages and Human-Centric Computing (VL/HCC'06)*, páginas 43–50. IEEE, 2006.
- ABRAHAM, R., ERWIG, M., KOLLMANSBERGER, S. y SEIFERT, E. Visual specifications of correct spreadsheets. En *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, páginas 189–196. IEEE, 2005.
- ASGHAR, M. Z., SANA, I., NASIR, K., IQBAL, H., KUNDI, F. M. y ISMAIL, S. Quizzes: Quiz application development using android-based mit app inventor platform. *International Journal of Advanced Computer Science and Applications*, vol. 7(5), páginas 43–54, 2016.
- BELLON, S., KOSCHKE, R., ANTONIOL, G., KRINKE, J. y MERLO, E. Comparison and evaluation of clone detection tools. *IEEE Transactions on software engineering*, vol. 33(9), páginas 577–591, 2007.
- BLACKWELL, A. F. First steps in programming: A rationale for attention investment models. En *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, páginas 2–10. IEEE, 2002.
- BOYD, R. *Implementing Reinforcement Learning in Unreal Engine 4 with Blueprint*. Tesis Doctoral, University Honors College, Middle Tennessee State University, 2017.
- CYPHER, A. Eager: programming repetitive tasks by example. En *CHI '91*. 1991.
- CYPHER, A. y HALBERT, D. C. *Watch what I do: programming by demonstration*. MIT press, 1993.
- DANN, W. P., COOPER, S. y PAUSCH, R. *Learning to Program with Alice (w/CD ROM)*. Prentice Hall Press, 2011.
- DEV. Libro blanco del desarrollo español de videojuegos 2016. *Desarrollo Español de Videojuegos*, 2016.

- DIJKSTRA, E. W. ET AL. A note on two problems in connexion with graphs. *Numerische mathematik*, vol. 1(1), páginas 269–271, 1959.
- FISCHER, G., GIACCARDI, E., YE, Y., SUTCLIFFE, A. G. y MEHANDJIEV, N. Meta-design: a manifesto for end-user development. *Communications of the ACM*, vol. 47(9), páginas 33–37, 2004.
- HAIBT, L. M. A program to draw multilevel flow charts. En *Papers presented at the the March 3-5, 1959, western joint computer conference*, páginas 131–137. 1959.
- JOHNSON, G. W. *LabVIEW graphical programming*. Tata McGraw-Hill Education, 1997.
- KO, A. J., ABRAHAM, R., BECKWITH, L., BLACKWELL, A., BURNETT, M., ERWIG, M., SCAFFIDI, C., LAWRENCE, J., LIEBERMAN, H., MYERS, B. ET AL. The state of the art in end-user software engineering. *ACM Computing Surveys (CSUR)*, vol. 43(3), páginas 1–44, 2011.
- KO, A. J. y MYERS, B. A. Designing the whyline: a debugging interface for asking questions about program behavior. En *Proceedings of the SIGCHI conference on Human factors in computing systems*, páginas 151–158. 2004.
- KO, A. J., MYERS, B. A. y AUNG, H. H. Six learning barriers in end-user programming systems. En *2004 IEEE Symposium on Visual Languages-Human Centric Computing*, páginas 199–206. IEEE, 2004.
- LESHED, G., HABER, E. M., MATTHEWS, T. y LAU, T. Coscripter: automating & sharing how-to knowledge in the enterprise. En *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, páginas 1719–1728. 2008.
- LIN, J. y LANDAY, J. A. Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. En *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, páginas 1313–1322. 2008.
- LITTLE, G. y MILLER, R. C. Translating keyword commands into executable code. En *Proceedings of the 19th annual ACM symposium on User interface software and technology*, páginas 135–144. 2006.
- LIU, H. y LIEBERMAN, H. Metafor: Visualizing stories as code. En *Proceedings of the 10th international conference on Intelligent user interfaces*, páginas 305–307. 2005.

- MILLER, P., PANE, J., METER, G. y VORTHMANN, S. Evolution of novice programming environments: The structure editors of carnegie mellon university. *Interactive Learning Environments*, vol. 4(2), páginas 140–158, 1994.
- MYERS, B. A., KO, A. J. y BURNETT, M. M. Invited research overview: end-user programming. En *CHI'06 extended abstracts on Human factors in computing systems*, páginas 75–80. 2006.
- NARDI, B. A. *A small matter of programming: perspectives on end user computing*. MIT press, 1993.
- PANKO, R. Finding spreadsheet errors: Most spreadsheet models have design flaws that may lead to long-term miscalculation. *Information Week*, vol. 529, página 100, 1995.
- PERDIKURI, K. Studentséxperiences from the use of mit app inventor in classroom. En *Proceedings of the 18th Panhellenic conference on informatics*, páginas 1–6. 2014.
- PHALGUNE, A., KISSINGER, C., BURNETT, M., COOK, C., BECKWITH, L. y RUTHRUFF, J. R. Garbage in, garbage out? an empirical look at oracle mistakes by end-user programmers. En *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, páginas 45–52. IEEE, 2005.
- RESNICK, M., MALONEY, J., MONROY-HERNÁNDEZ, A., RUSK, N., EASTMOND, E., BRENNAN, K., MILLNER, A., ROSENBAUM, E., SILVER, J., SILVERMAN, B. ET AL. Scratch: programming for all. *Communications of the ACM*, vol. 52(11), páginas 60–67, 2009.
- RONEN, B., PALLEY, M. A. y LUCAS JR, H. C. Spreadsheet analysis and design. *Communications of the ACM*, vol. 32(1), páginas 84–93, 1989.
- ROSSON, M. B., SINHA, H., BHATTACHARYA, M. y ZHAO, D. Design planning in end-user web development. En *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, páginas 189–196. IEEE, 2007.
- ROTHERMEL, G., LI, L., DUPUIS, C. y BURNETT, M. What you see is what you test: A methodology for testing form-based visual programs. En *Proceedings of the 20th international conference on Software engineering*, páginas 198–207. IEEE, 1998.
- SCAFFIDI, C., SHAW, M. y MYERS, B. Estimating the numbers of end users and end user programmers. En *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, páginas 207–214. IEEE, 2005.

SEGAL, J. End-User Software Engineering and Professional End-User Developers. En *End-User Software Engineering* (editado por M. H. Burnett, G. Engels, B. A. Myers y G. Rothermel), número 07081 en Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Dagstuhl, Germany, 2007. ISSN 1862-4405.

SHAW, M. Avoiding costly errors in your spreadsheets. *Contractors Management Report*, vol. 11, páginas 2–4, 2004.

XU, B., QIAN, J., ZHANG, X., WU, Z. y CHEN, L. A brief survey of program slicing. *ACM SIGSOFT Software Engineering Notes*, vol. 30(2), páginas 1–36, 2005.

YE, Y. y FISCHER, G. Reuse-conducive development environments. *Automated Software Engineering*, vol. 12(2), páginas 199–235, 2005.

## Apéndice A

# Resultados de la evaluación

**Cuestión 1: Me gusta la experiencia de uso de la aplicación y su estética.**

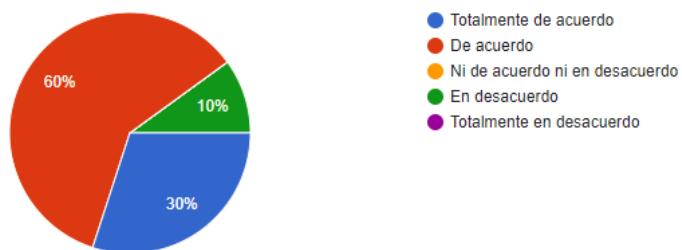


Figura A.1: Resumen de las respuestas obtenidas de la primera pregunta del cuestionario.

**Cuestión 2: Me gusta el manejo de la aplicación y sus controles (movimiento, botones, cámara, etc.)**

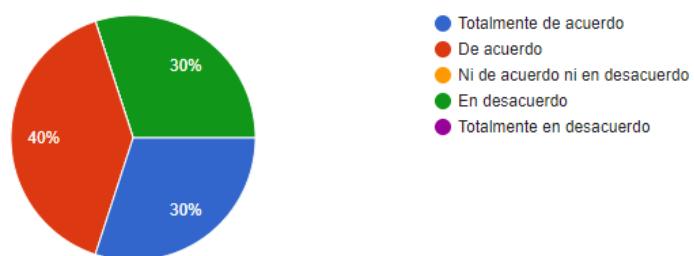


Figura A.2: Resumen de las respuestas obtenidas de la segunda pregunta del cuestionario.

**Cuestión 3: Prácticamente no he cometido errores ni he tenido que dedicar tiempo a corregirlos (eliminando bloques, personajes o decoraciones, cambiándolos de sitio, etc.)**

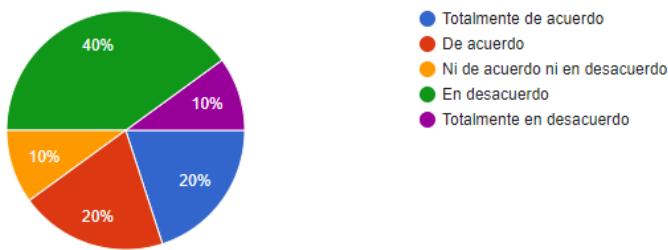


Figura A.3: Resumen de las respuestas obtenidas de la tercera pregunta del cuestionario.

**Cuestión 4: Indica cómo crees que podría mejorarse la aplicación (experiencia de uso y estética, manejo y controles, gestión de errores, etc.)**

1. Cuando un personaje IA está entre otros dos compañeros, un obstáculo y el borde de la pantalla... a veces se queda colgado y no es capaz de moverse :-( Nunca hemos podido jugar una partida entera porque se cuelga. Para girar hace falta llevar el ratón hasta una esquina de la pantalla, pero sería mejor con un botón abajo para girar o algo así... es cansado y poco intuitivo llevar el ratón hasta la esquina. No se puede crear un nuevo videojuego sin borrar el anterior (además al cargar un juego creado, no machaca el título y descripción que tengo puesto ahora el juego).
2. Marcando en la imagen del cursor que objeto tienes seleccionado (bloque, medico, virus, etc). Una pequeña imagen ayudaría :).
3. Me gustaría que hubiese armas como un martillo para vencer al Coronavirus y pegar castañazos y tal.
4. A la hora de eliminar estaría bien poder hacer una eliminación múltiple de objetos(si existe esa posibilidad no he visto como se hacía).
5. Se podría implementar una opción de cámara rápida.<sup>en</sup> los combates para hacer la experiencia más llevadera a jugadores más impacientes.
6. El control de la cámara es muy confuso, la muevo sin querer cuando quiero editar un bloque cercano. Creo que sería mejor hacer botones explícitos (por ejemplo, flechas en los laterales de la pantalla que haya que pulsar) para rotar la cámara, porque no es algo que se haga

demasiado a menudo. También añadiría la posibilidad de rotar en el eje vertical, es decir, poder ver el mapa desde arriba o desde más abajo. Me parece muy necesaria la capacidad de mover la cámara por el escenario, quizás pulsando con el botón central del ratón como hacen muchas aplicaciones. Al querer posicionar un cubo en el único hueco central que me quedaba por llenar, he tenido dificultades por varias razones: en primer lugar, no podía ver la parte sobre la que se puede colocar un cubo (la zona negra), y en segundo lugar, no se puede interactuar con las paredes de los demás cubos (por ejemplo, en Minecraft sí se puede hacer esto). No podía colocar el cubo en ese área central y he tenido que borrar todo el suelo que había delante para poder acceder a ese hueco libre. Rotar la cámara hace que desaparezca el menú de selección de objetos. Por culpa de esto, no puedo escoger un objeto que está en el lateral del menú, porque me acerco mucho al borde de la pantalla y se cierra el menú. Además, no está claro que se pueda hacer scroll en el menú de objetos.

7. El mismo botón del ratón no debería utilizarse para rotar y para eliminar. Rotar es una acción constructiva, mientras que eliminar es una acción destructiva. Desde mi punto de vista, debería alternarse entre dos modos: construir y borrar; el primero permitiría construir y rotar con los dos botones del ratón, mientras que el segundo solo permitiría borrar. Además, el cursor debería cambiar cuando el modo de borrar esté activo, para intentar prevenir accidentes. Por otro lado, la rotación de la cámara estaba completamente rota en mi caso, y la mayor parte del tiempo lo único que se conseguía al acercar el ratón a los bordes era que pivotara en sentidos alternos. Habría sido más práctico utilizar el botón central del ratón para rotar, o bien una combinación de teclas como Ctrl + clic y arrastrar. El nombre y la descripción del proyecto deberían solicitarse al guardar, y no antes. Además, cuando se sale de la aplicación, un mensaje indica que puedes guardar el progreso, pero no está claro cómo puede hacerse una vez se ha abandonado el modo de edición. Y, por último, se agradecería que las texturas de los bloques no evidenciaran tanto la repetición (la nieve, por ejemplo, parece más bien un suelo de baldosas).
8. Que los menús no desaparezcan al mover el ratón. Que el giro de cámara no sea acercando al borde. Que se pueda jugar y luego volver a editar sin pasar por todo el proceso de poner título, descripción, etc.
9. Molaría que se pudiese manejar con mando de la Xbox.
10. Mayor facilidad para cambiar el tipo de bloques o personajes.

**Cuestión 5: Me parece que hay suficiente información disponible como para aprender a utilizar la aplicación sin ayuda.**

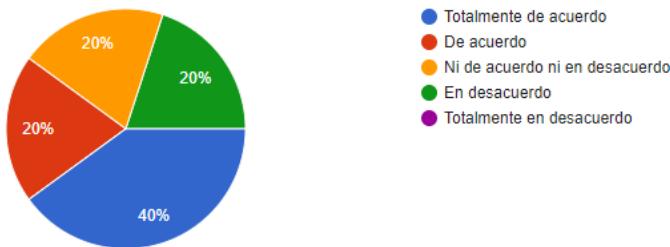


Figura A.4: Respuestas obtenidas de la quinta pregunta del cuestionario.

**Cuestión 6: Indica cómo crees que podría mejorarse la documentación de la aplicación y facilitar su aprendizaje.**

1. Quizá algo para dejar más claro cómo se rota la cámara... y también lo de la rueda del ratón para acercar la pantalla.
2. Quizá un pequeño tutorial muy guiado (opcional) para hacer un nivel super pequeño, con un grid de 5x5, por ejemplo.
3. Bueno, Tomás no sabe leer... así que aprendió viendo a sus hermanos, en realidad :-).
4. Estaría interesante añadir algún tipo de vídeo con un pequeño tutorial rápido pensado para los mas peques.
5. Explicando cómo funcionan las distintas opciones a la hora de jugar, como "esperar".
6. Un tutorial para hacer un mapa sencillo y para saber cómo funciona el combate sería adecuado: no he entendido bien por qué zonas puede subir un personaje, por qué a veces no puede atacar, por qué a veces sólo se puede mover el enemigo y qué es el atributo con el rayo.
7. Falta un tutorial que ejemplifique de manera clara todas las funciones de la aplicación. Con un recorrido por la interfaz basado en mensajes de ayuda sería suficiente.
8. Explicar mejor los parámetros de los personajes.
9. Estaría bien que te pusiesen como un tutorial interactivo que te vaya diciendo cómo hay que hacerlo.
10. Como rotar la camara.

**Cuestión 7: Habías usado anteriormente alguna otra aplicación para crear videojuegos de forma sencilla.**

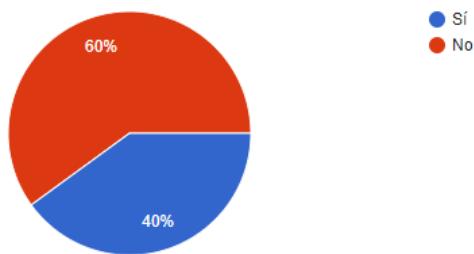


Figura A.5: Resumen de las respuestas obtenidas de la séptima pregunta del cuestionario.

**Cuestión 8: Menciona esas otras aplicaciones si las hay, e indica qué nuevas características te gustaría que se añadiesen a esta aplicación que has probado.**

1. --
2. Sólamente he probado pero muy un poco el Mario Maker, y Unreal es muy complicado. A lo mejor meter algún enfermo podría haber estado bien.
3. Mario Maker 2 (aunque es más un creador de niveles).
4. Ninguno.
5. Mayor variedad de bloques, que algunos de ellos fuesen interactivos(trampas, fuentes de salud, etc).
6. Scratch. Permitiría la opción de crear mis propios personajes con sus respectivas características.
7. Construct, Scratch, Super Mario Maker.
8. Lo que más echo en falta es una paleta de herramientas de construcción desde la que se puedan seleccionar distintos tipos de bloque sin abrir ningún menú, tal y como ocurre en Super Mario Maker.
9. Super Mario Maker. Yo quisiera que hubiese más personajes, no siempre los mismos. Que haya otras escenas, no los mismos cubos todo el rato.
10. Es la primera vez.

**Cuestión 9: Recomendaría el uso de esta aplicación para la creación de videojuegos comerciales.**

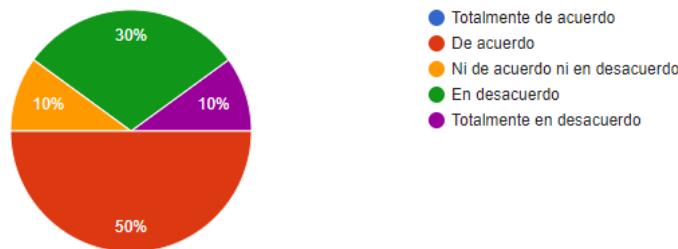


Figura A.6: Resumen de las respuestas obtenidas de la novena pregunta del cuestionario.

**Cuestión 10: Recomendaría el uso de esta aplicación con fines educativos.**

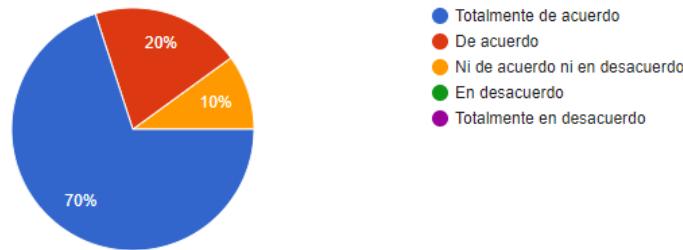


Figura A.7: Resumen de las respuestas obtenidas de la décima pregunta del cuestionario.

**Cuestión 11: En este último apartado puedes hacer cualquier comentario, sugerencia o aclaración que quieras. Tu opinión nos ayudará a seguir mejorando.**

1. La noticia triste del Coronavirus se hace más alegre gracias a este juego. Me habrá encantado que me hubiesen pedido en el colegio que cada alumno hubiese hecho un videojuego con esta aplicación.
2. Me parece muy buena aplicación, muy contenida y autoexplicativa. Creo que podría usarse perfectamente con otros modelos y en otros contextos con fines educativos, ya que permitiría a muchos profesionales aprovechar las ventajas de la enseñanza con videojuegos. Por desgracia cuando estaba jugando, he tenido un crash con fatal error. Pero antes de eso, he podido hacer una pequeña estrategia de ir 3 médicos a por 1 virus, y ha estado divertido.

3. Los muñecos y los virus son graciosos.
4. Añadir al tema de los bloques interactivos, posibilidad de poder hacer una grabación de la partida que se juegue y poder compartirla por redes sociales.
5. Podrías implementar una mayor variedad de bloques para hacer los escenarios más coloridos y diversos.
6. La aplicación es adecuada, pero creo que le falta pulir algunos controles para que alguien sin experiencia pueda utilizarla fácilmente.
7. El proyecto es muy interesante y tiene un gran potencial. ¡Enhorabuena!. Eso sí, intentad tener más en cuenta en el futuro la experiencia de usuario, y meditad mejor algunas cuestiones menores, como las que os he comentado, que podrían mejorar mucho el resultado final.
8. Los niveles no se guardan bien (no guarda las rotaciones y algunos bloques de suelo desaparecen) y esto es frustrante. Para su uso en educación hay que evitar a toda costa las fuentes de frustración.
9. Ha molado crear el juego, está guay.
10. Me ha gustado bastante.



## **Apéndice B**

# **Introduction**

### **B.1. Motivation**

The revolution of new technologies is having a strong impact both socially and economically, favoring the emergence of new business models as well as the emergence of new industries, as is the case of the video game industry.

This sector has a great growth projection both at a national and international level, being that in the last twenty years it has been consolidated as a relevant contributor to the world economy of entertainment. In comparison with other more established entertainment industries, such as the film and music industries, it has suffered an exponential growth both in number of users and in numbers at a commercial level.

The video game industry today is a reference in the global digital economy (DEV, 2016), not only for the aspects already mentioned, but by the number of new professionals, companies built and existing software for the creation of such products. As a consequence, this software has been evolving progressively with the aim of supply the current demands when developing video games.

This evolution has made that tools that in their beginnings were simple have become, over time and after many updates, in great graphic engines with which it is possible to create all type of videogames, which can be used in multiple devices. This has caused that any current development tool such as Unreal Engine 4 or Unity requires a very high degree of specialization, making videogame development less accessible to the general public over time.

In spite of this, these tools have bet during the last years for other alternatives to the development that do not require high technical knowledge. This other alternatives are the visual programming languages, also known as *blueprints*. *Blueprints* are a visual scripting system that uses a node-based interface to create game elements. In addition, like other visual scripting systems such as Scracth (Resnick et al. 2009) also allows the definition of classes or objects in the engine itself.

It is a very flexible and powerful system, since it provides a layer of abstraction with which it is not necessary to have high technical knowledge to be used. However, this visual scripting system is aimed at both designers and programmers, so that it allows better communication between members of the same development team but from different profiles. This is an option that brings the development of video games to a wider audience, however it is not an option designed for ordinary users.

There are also other tools with visual scripting systems apart from those already mentioned, such as Bolt, PlayMaker or Adventure Creator, which are used in conjunction with a graphic engine, in this case Unity, greatly simplifying the development. However, it is still necessary to use a complex engine to develop, then it is necessary to have high technical knowledge

On the other hand we have a series of tools focused on a more general public like RPGMaker, where technical knowledge is not necessary when developing. These tools use a point and click system with which it is only necessary to select the objects you want to add to the game and are released in the editor, without the need to program or make use of visual scripting systems.

These types of tools have been gaining strength in recent years and therefore we have seen the creation of TRPGMaker, a desktop application for the design and development of tactical role-playing games developed with C++ and Unreal Engine 4.

## B.2. Objectives

The main purpose of this project is the creation of a desktop application focused on the design and development of tactical role-playing games, aimed at a general public without technical knowledge about this discipline. To do this, we will carry out a series of activities in order to achieve these objectives:

- **Objective 1:** there will be an in-depth review of the issue of video game creation tools for the end-user.
- **Objective 2:** a specific genre will be chosen to implement a user-friendly prototype that does not require manuals and can be tested by any user. This first prototype will be evaluated with real users, taking into account the qualitative information regarding the simplicity of use in the rest of the project.
- **Objective 3:** a more complex system will be designed according to the feedback received by the users, adding different forms of end-user programming according to the tool and the type of games that are intended to be created with it.
- **Objective 4:** a first functional version of the system will be evaluated by comparison with other commercial tools available on the market.

### B.3. Project scope

The scope of this project will be limited to the creation of an alpha version. An alpha version within the software life cycle corresponds to a functional product that is evaluated by users but that is still unstable, so that it may contain errors or fail to implement all requirements. This limitation is given by the fact that a project like this needs several technical profiles (animators, professional developers or software architects) to be a real product similar to those already on the market. That is why it is necessary to limit the scope of the system as it is an end-of-master project to be implemented by a single person.

The result of this project will be an alpha version of the TRPG Maker tool, which will be published on the [itch.io](#) platform and made available to the community for free use.

### B.4. Document organization

In order to guide the reader through the reading of the document, this section describes the structure of the document. This report is divided into the following chapters:

- **Chapter 1. Introduction:** in this chapter the current situation within video game development is exposed and the motivation of the project is explained. In addition, it presents the objectives to be achieved and the scope of the project.

- **Chapter 2. State of the art:** this chapter focuses on end user development. In addition, an analysis of several tools is made in order to take into account the common characteristics for the development of *TRPG Maker*.
- **Chapter 3. Development tools:** this chapter details the technologies used in the development of the project.
- **Chapter 4. Methodology and project management:** this chapter details the methodology and planning applied for this project.
- **Chapter 5. System development:** in this chapter the architecture and the design of the system are exposed, explaining the relative phases to the development and the technical specifications of the system.
- **Chapter 6. Tests and results:** this chapter documents the user tests carried out as well as the results obtained from them.
- **Chapter 7. Conclusions:** in this chapter the conclusions obtained after the realization of the project are exposed.

Finally, along with this report, a series of appendices will be provided:

- **Appendice A. Evaluation results:** raw results of the experimentation session carried out with users.
- **Appendice B. Introduction:** contains the introduction chapter in English.
- **Appendice C. Conclusions:** contains the conclusions chapter in English.

## Apéndice C

# Conclusions

In this project we have deepened in the development for the final user and in the implications of this type of tools in the present day of the development of video games. Furthermore, with the implementation of TRPG Maker, a tool for the creation of tactical role-playing games, we have achieved with remarkable success the main objectives of this project, which we describe below

- **Activity 1:** an in-depth study of the state of the art on development for the end user has been carried out and several tools have been studied, both professional and related to the field of education, for the development of video games designed for the end user. On the one hand, the study related to children's educational tools has been applied during the first milestone for the development of the prototype *TRPG Maker: The Good Doctors*, while the study of the more professional tools has been taken into account especially during the second milestone, with a view to the development of a more complete version of the tool focused on independent developers.
- **Activity 2:** the project was initially focused on a concrete audience, defined as children between 8 and 12 years of age, with the aim of studying the possibilities of a system of these characteristics in the field of education. In addition, a completely open experimental session was held with real users, with special emphasis on the qualitative information obtained from it in relation to the simplicity of use of the system.
- **Activity 3:** the original prototype has been improved by using feedback from previous experiments, and various forms of programming have been implemented for the end user (programming with text entries and point and click programming systems) in accordance with the tool and depending on its needs when developing video games, using the professional tools studied previously as a reference.

- **Activity 4:** to complete the development of the project, a feature comparison was carried out based on other commercial tools on the market, as a validation of the first functional version of TRPG Maker.

The implementation of this series of activities has ended with the publication of the tool on [itch.io](#)<sup>1</sup> so that any user can access it for free.

As part of the conclusions we have been able to observe that the tool still needs a lot of work to become a complete product, even though it has integrated the feedback provided by the users. The majority of comments highlighted control failures, level saving failures and in some cases even application closures. Another point for improvement is the amount of content available, both in terms of characters and objects in the game and new mechanics that can be used in the development of levels.

It should also be noted that part of the objectives of this project contemplated the implementation of a visual programming editor with which users could manage the application's event queues, a functionality that has not been implemented due to the lack of time and the high cost of the implementation itself.

Even so, TRPG Maker has been widely accepted by users and most of them have seen it as a useful tool in the educational field. As possible future work, we plan to continue improving the tool with a new approach focused on the educational field as a complementary tool in classes of design and development of video games.

---

<sup>1</sup>[TRPG Maker](https://narratech.itch.io/trpg-maker) - <https://narratech.itch.io/trpg-maker>