

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS**

**Curso Académico 2018/2019**

**Trabajo Fin de Grado**

**Diseño y Desarrollo de Toolbox para crear videojuegos 2D**

**Autor**: Juan Daniel Silva

**Directores**: Julio Guillén García



En el caso de que se desee puede dedicarse una página para agradecimientos. El formato de agradecimientos es libre siempre que se mantenga el tipo de letra definido como base del formato. Por ello se admiten agradecimientos del estilo:

“Quiero aprovechar la ocasión para agradecer el apoyo de mis … y de mis... Asimismo no puedo olvidar en este momento a mis compañeros …”

Aunque también se admiten agradecimientos de la forma:

A mi familia

A mi tutor

A mis compañeros

Resumen

En este Proyecto se pretende desarrollar un Toolbox para un videojuego 2D. La idea principal es plantear una serie de mecánicas y posibilidades básicas para ampliar la experiencia de un juego cuyos objetivos ya están planteados, pero también ofrecer cambios en ciertas configuraciones de forma que cada usuario pueda crear diversos escenarios con fines diferentes a los planteados en el videojuego normal, para así modificar parcial o completamente dichos objetivos, y de este modo, extender las posibilidades normales.

Para ello, no solo se plantea un Desarrollo de un editor de niveles, si no que también se ofrece un videojuego básico de plataformas 2D en el cual se ha basado el Desarrollo de dicho editor genérico, y cuyos niveles han sido desarrollados posteriormente en dicho Toolbox.

Palabras Clave:Toolbox, Editor, Mecánicas, Escenarios, Niveles, Plataformas, Aplicación.

Abstract

In this project we pretend to develop a 2D videogame Toolbox. The main idea is show several mechanics and possibilities to extend the game experience where objectives are planted, but offer too the chance of make changes relative to configuration, so that each user could create different stages with different objectives from what was raised in the normal videogame, to modify partially or completely this objectives, and extend the normal possibilities.

With this, not only present a level editor development, but we also offer a basic 2D platform videogame on which this level editor is based, and whose levels had been developed with the Toolbox.

Keywords: Toolbox, Editor, Mechanics, Stages, Levels, Platforms, Application.

# Índice de contenidos

[Índice de contenidos 7](#_Toc41474584)

[Índice de ilustraciones 10](#_Toc41474585)

[Índice de tablas 11](#_Toc41474586)

[Glosario 12](#_Toc41474587)

[Capítulo 1 Introducción 13](#_Toc41474588)

[1.1 Apartado de historia 13](#_Toc41474589)

[1.2 Editores de Niveles 14](#_Toc41474590)

[1.2.1 Editores Oficiales 14](#_Toc41474591)

[1.2.2 Editores No Oficiales 18](#_Toc41474592)

[1.2.3 Editores De Desarrollo 18](#_Toc41474593)

[1.2.3.1 UbiArt Framework 18](#_Toc41474594)

[1.2.3.2 RPG MAKER 19](#_Toc41474595)

[1.2.3.3 TILED 20](#_Toc41474596)

[1.3 Apartado de mercado actual 21](#_Toc41474597)

[1.4 Apartado de estado del arte 21](#_Toc41474598)

[Capítulo 2 Objetivos 23](#_Toc41474599)

[2.1 Descripción del problema 23](#_Toc41474600)

[2.2 Objetivos 23](#_Toc41474601)

[2.3 Estudio de alternativas 24](#_Toc41474602)

[2.4 Metodología empleada 25](#_Toc41474603)

[2.5 Planificación 25](#_Toc41474604)

[Capítulo 3 Marco teórico 26](#_Toc41474605)

[3.1 Juegos de Plataformas 26](#_Toc41474606)

[3.2 Leyes de UX 27](#_Toc41474607)

[3.3 Mecánicas 27](#_Toc41474608)

[3.4 Game Design vs Level Design 27](#_Toc41474609)

[Capítulo 4 Especificación 31](#_Toc41474610)

[4.1 Unity 3D 31](#_Toc41474611)

[4.2 C# 32](#_Toc41474612)

[4.3 Visual Studio 2017 33](#_Toc41474613)

[4.4 GitHub 33](#_Toc41474614)

[4.5 Trello 34](#_Toc41474615)

[Capítulo 5 Descripción informática 36](#_Toc41474616)

[5.1 Etapas del Desarrollo 36](#_Toc41474617)

[5.1.1 Etapa 1: Desarrollo del White Box 36](#_Toc41474618)

[5.1.2 Etapa 2: Busqueda de Assets 36](#_Toc41474619)

[5.1.3 Etapa 3: Mecánicas 37](#_Toc41474620)

[5.1.4 Etapa 4: Mecanicas del Editor 37](#_Toc41474621)

[5.1.5 Etapa 5: Sistema de Guardado 38](#_Toc41474622)

[5.1.6 Etapa 6: Máquina de Estados 38](#_Toc41474623)

[5.1.7 Etapa 7: Playtesting 38](#_Toc41474624)

[5.1.8 Etapa 8: Inclusión de Assets Decorativos 38](#_Toc41474625)

[5.2 Leyes UX 39](#_Toc41474626)

[5.2.1 Ley de Fitts 39](#_Toc41474627)

[5.2.2 Ley de Hyck 39](#_Toc41474628)

[5.2.3 Ley de Jakob 39](#_Toc41474629)

[5.2.4 Ley de Región Común 40](#_Toc41474630)

[5.2.5 Ley de Proximidad 40](#_Toc41474631)

[5.3 Mecánicas y Funcionalidades Implementadas 40](#_Toc41474632)

[5.3.1 Transformaciones 40](#_Toc41474633)

[5.3.2 Traslación 41](#_Toc41474634)

[5.3.3 Rotación 41](#_Toc41474635)

[5.3.4 Escala 41](#_Toc41474636)

[5.3.5 Instanciar 41](#_Toc41474637)

[5.3.6 Serialización 41](#_Toc41474638)

[5.3.7 Máquina de Estados 42](#_Toc41474639)

[5.3.8 IA’s de Enemigos 43](#_Toc41474640)

[Side to Side 43](#_Toc41474641)

[Pathfinding 43](#_Toc41474642)

[Distance Attack 44](#_Toc41474643)

[5.3.9 Comportamiento Imán 44](#_Toc41474644)

[Capítulo 6 Validación 45](#_Toc41474645)

[6.1 Resultado final 45](#_Toc41474646)

[6.2 Experimentación 45](#_Toc41474647)

[Capítulo 7 Conclusiones 46](#_Toc41474648)

[7.1 Logros alcanzados 46](#_Toc41474649)

[7.2 Lecciones aprendidas 46](#_Toc41474650)

[7.3 Lineas futuras 46](#_Toc41474651)

[Bibliografía 47](#_Toc41474652)

[Ludografía 48](#_Toc41474653)

[Anexo I Documento de diseño 49](#_Toc41474654)

[Anexo II Plan de negocios 50](#_Toc41474655)

[Anexo III Código 52](#_Toc41474656)

[Anexo IV Galería de imágenes 53](#_Toc41474657)

# Índice de ilustraciones

[Ilustración 1 - Super Mario Maker 14](#_Toc41380864)

[Ilustración 2 - Transformice 15](#_Toc41380865)

[Ilustración 3 - Valve Hammer Editor 16](#_Toc41380866)

[Ilustración 4 - UbiArt Framework 18](#_Toc41380867)

[Ilustración 5 - RPG MAKER MV 19](#_Toc41380868)

[Ilustración 6 - GauGAN en acción 21](#_Toc41380869)

[Ilustración 7 - Representación de la historia en videojuegos 28](#_Toc41380870)

# Índice de tablas

[Tabla 1 - Tabla de Marco Teórico 1 23](#_Toc8308222)

[Tabla 2 - Tabla de Marco Teórico 2 24](#_Toc8308223)

[Tabla 3 - Tabla del Marco Teórico 3 25](#_Toc8308224)

# Glosario

|  |  |
| --- | --- |
| Abandonware | Videojuegos descatalogados, bien por su antigüedad o bien porque su empresa desarrolladora ha desaparecido y sus derechos no pertenecen a otra. Al no ser posible su comercialización, la comunidad de jugadores entiende que es legal compartirlo y descargarlo siempre que no haya ánimo de lucro. |
| Demake | Adaptación de un videojuego moderno a una plataforma antigua. |
| Game Jam | Evento en el que un grupo de personas participan para desarrollar un videojuego en un tiempo limitado, a menudo 48 o 72 horas. Dependiendo del evento se pueden aplicar otras restricciones además del tiempo, como la temática, las herramientas que se pueden utilizar, etc. |
| Home brew | Aplicaciones y juegos creados por programadores de manera no profesional para cualquier plataforma, generalmente consolas. |
| Mods | Extensión del software que modifica un videojuego para ampliar sus posibilidades, ya sea con nuevos niveles, mecánicas, enemigos… |
| Framework | Entorno de trabajo pensado para facilitar la programación de cualquier aplicación o herramienta. |
|  |  |
|  |  |

# Capítulo 1 Introducción

|  |
| --- |
| Para un correcto entendimiento del Proyecto planteado, es necesario contextualizar historicamente el estado de los Toolbox en el ámbito de los videojuegos 2D. Para ello, se procederá a una vista atrás en editores ya en desuso, así como un analisis de los editores actualmente ofrecidos tanto por compañías oficiales como por terceros, estos últimos, en la mayoría de los casos, orientados para el Desarrollo de mods de videojuegos concretos. |

## Apartado de historia

En el origen de los videojuegos, la idea de poder crear tu propio nivel para compartirlo con diferentes usuarios, o simplemente aumentar el tiempo de juego era prácticamente inexistente, ya que o no se contaba con los suficientes recursos para poder trabajar en ello, o simplemente las empresas desarrolladoras no ofrecían esta posibilidad.

Con el avance tecnológico (principalmente en computadores), poco a poco surgieron pequeños desarrolladores que se embarcaban en la ardua tarea de modificar los juegos más vendidos del momento (generalmente aquellos que contaban con un modo multijugador). Esto no solo llevo a la modificación directa del videojuego, si no también al desarrollo de aplicaciones externas que permitían a cualquier usuario desarrollar niveles de sus juegos favoritos e introducirlos de manera no oficial en ellos.

Según la web **Game Creation Tools Clasification** [1], una página colaborativa centrada en la clasificación de herramientas de creación de videojuegos, el primer editor de niveles fue para el videojuego Eamon, para Apple II. Se trataba de un juego de dominio público basado completamente en texto, a excepción de algunas aventuras que contenían gráficos. Por ello, los usuarios podían crear sus propias historias con el disco para desarrolladores, lo que expandió el juego durante años, llegando incluso a comercializarse una versión más sofisticada. Unos años más tarde, surge Adventure Construction Set (ACS), utilizado para la construcción de juegos Ultima-type, de Electronic Arts, pionero en las aventuras gráficas. Si seguimos avanzando en el tiempo, podemos descubrir como aparecen más y más editores, muy pocos oficiales y con acceso al público, y muchos por parte de terceros. Con el avance de los gráficos de ordenador, comienzan a verse juegos en 3D, como Fighter Maker, que permite jugar contra el ordenador o contra un segundo jugador, y además ofrece un modo de edición donde los usuarios pueden editar el perfil de los personajes, establecer qué botones ejecutan los movimientos y la cantidad de daño de estos. Pueden editarse hasta 800 movimientos diferentes de 20 estilos de combate distintos.

Algunas de las empresas de videojuegos que vieron este crecimiento, tomaron la decision de incluir en su software las herramientas necesarias para que los jugadores fuesen capaces de crear niveles sin necesidad de descargar aplicaciones de terceros. Esto a su vez, fomentó la creación de Editores de niveles y Toolbox, lo que agilizó el Desarrollo de los videojuegos, de forma que empresas como Ubisoft, creasen un editor para sus propios desarrolladores, evitando así la intervención directa de los programadores a la hora de crear y empaquetar cada nivel de cada videojuego, tal y como ocurrió con **Rayman Origins**.

Con esto, también han surgido videojuegos que consisten directamente en la creación de niveles de videojuegos, para posteriormente compartirlos entre la comunidad de jugadores, dando la posibilidad de votar dichos niveles y generar un ranking entre los mejores, como es el caso de **Super Mario Maker** y **Super Mario Maker 2**.

Entre los multiples editores encontrados, hemos decidido escoger los siguientes, atendiendo a la siguiente clasificación:

## Editores de Niveles

## Editores Oficiales

#### Super Mario Maker

Mario Maker es un videojuego diseñado por la compañía japonesa de videojuegos Nintendo que ofrece la posibilidad de crear escenarios de la saga Super Mario Bros, publicarlos, y permitir que otros usuarios prueben los mapas creados por los jugadores.

Para ello, el videojuego ofrece un editor donde podemos elegir la versión del videojuego que deseamos utilizar, y una barra de herramientas con 12 elementos que podemos colocar. Esta barra de herramientas ofrece la posibilidad de cambiar la combinación de elementos, la cual se puede editar, liberando así el espacio ocupado en la pantalla.



Ilustración 1 - Super Mario Maker

La parte “jugable” del nivel, se divide por medio de una rejilla, limitando así dónde se pueden colocar los diferentes elementos en el mapa. Además, para ciertos objetos, como contenedores, lo que hace es colocar el contenedor en cuestión en el escenario, y arrastrando dentro de él otro de los elementos, automáticamente se introduce en él, permitiendo que este objeto salga cuando el usuario golpee el cubo.

También, debido a la similitud entre objetos, para evitar tener un número excesivo de elementos en la barra de objetos, permite el cambio de uno por otro simplemente agitando el objeto en cuestión tras ser arrastrado al mapa.

#### Transformice

Transformice es un juego web plataformas donde controlamos a un ratón. Con un tiempo determinado por mapa, debemos conseguir una porción del queso que se encuentra en el mapa y llegar a la ratonera.

Cada mapa es diferente, y prácticamente todos los niveles han sido desarrollados por la comunidad gracias al editor que tiene el juego. A rasgos generales, este editor cuenta con un número finito de objetos (considerados como elementos con físicas utilizados para estorbar, ayudar o decorar), y un número finito de plataformas.

Estas plataformas son escalables, pero no tileables, ya que se trata de una textura cuadrada de alta resolución, por lo que la calidad no varía cuando se escala en ambos ejes, pero al hacerlo solo en uno, se puede observar cómo se distorsiona la textura.



Ilustración 2 - Transformice

En lo referente al modo de edición de mapas, como se puede apreciar en la imagen, unos radio button nos permiten seleccionar en función de las características, no solo de los elementos, si no también del mapa. Una vez seleccionada una de estas características, se nos mostrará un segundo menú con los elementos cuyas características se corresponden con las seleccionadas. Estos elementos se eligen por su nombre en alguna de las características y por imágenes en otras.

Esto, desde mi punto de vista, reduce la facilidad de editar niveles para los usuarios, ya que teniendo en cuenta que, como ya se ha destacado antes, gran parte de los niveles han sido creados por los usuarios, esto generaliza el uso de algunos elementos, lo que provoca un desuso constante de objetos no reconocibles por los usuarios.

Además, en muchos casos, la mayoría de los usuarios no conocen los nombres concretos de dichos objetos, por lo que encontrar la entidad suele ser tedioso en muchas ocasiones.

A pesar de ello, uno de sus puntos a favor es la colocación de elementos. Una vez seleccionado el objeto en cuestión, aparecerá junto al puntero del ratón, y manteniendo pulsado el botón izquierdo, se podrá colocar y escalar. Cuando soltemos el botón, podremos acceder a sus valores (posición, rotación características específicas), pudiendo editar los valores numéricos introduciendo nuestros propios valores, y permitiendo modificar algunas de las características normales de la entidad.

#### Broforce

#### Valve Hammer Editor

Aunque se trate de una herramienta para el desarrollo de niveles en 3D, se ha considerado importante hablar de ella, ya que ha tenido una gran repercusión en este campo debido a la cantidad de mods creados para los juegos de la compañía.

Valve Hammer Editor es un editor de mapas desarrollado por Valve para el motor de videojuegos **Source**. Este editor es utilizado para el desarrollo de diversos juegos de la compañía como Half-Life 2, Portal y Left 4 Dead, así como por empresas independientes para la creación de mods.

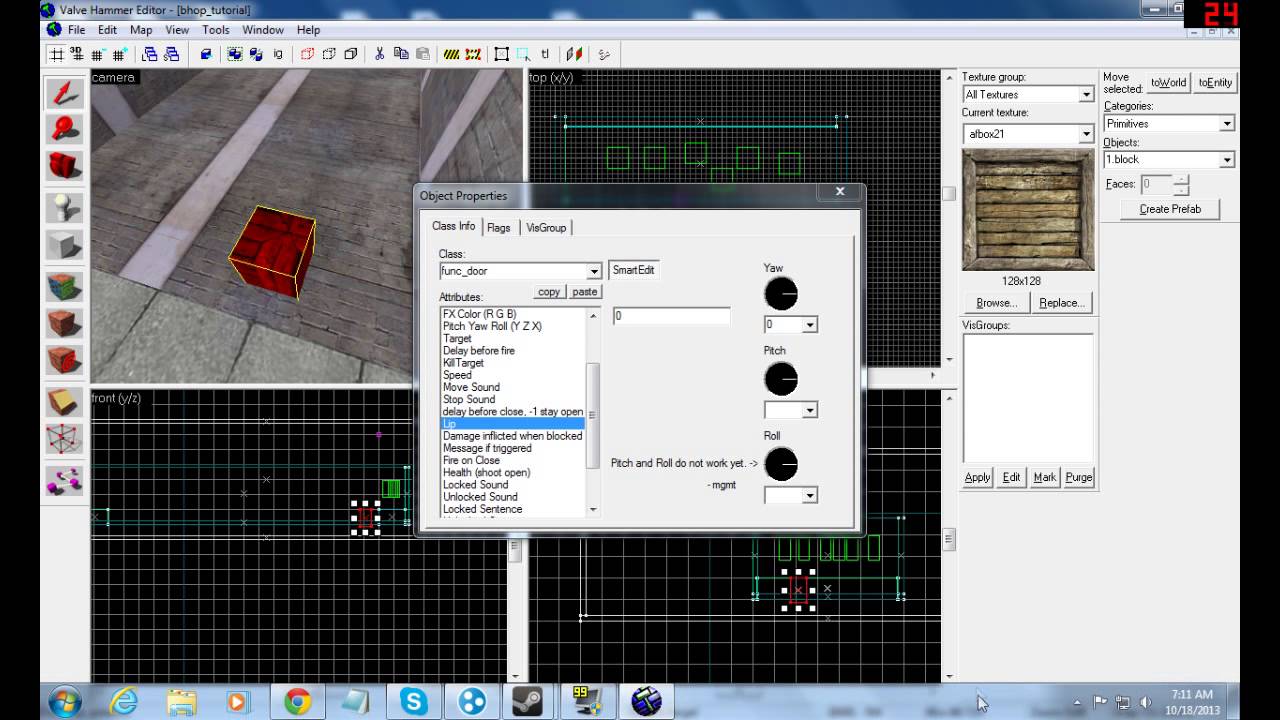


Ilustración 3 - Valve Hammer Editor

En el editor, al igual que en algunos programas de edición 3D, nos muestra 4 imágenes desde diferentes perspectivas (isométrica, planta, alzado y perfil), lo que nos permite editar los elementos desde todas las perspectivas, y a su vez comprobar los resultados en tiempo real tal y como se verán en el resultado final.

Cuenta con varios botones clave para su rápido acceso, ya que son las herramientas más utilizadas a la hora de crear diversos mapas. Entre ellas se encuentra:

* **Herramienta Selección**: Nos permite seleccionar los diversos elementos de la escena que estamos creando. El problema puede ser el tener que acceder a dicha herramienta cada vez que deseamos seleccionar un objeto, y cambiar de herramienta de nuevo para modificar el elemento.
* **Herramienta de Cámara**: Aunque podamos movernos con las letras AWSD, este movimiento solo se hace en los ejes X e Y, por lo que la herramienta nos ayuda a movernos con mayor libertad con el ratón.
* **Herramienta de Entidades**: Con ella, colocamos diferentes personajes en la escena.
* **Aplicación de Texturas**: Este botón nos muestra una ventana desde la que podemos editar diferentes texturas para nuestros objetos, las cuales se aplican utilizando las dos herramientas siguientes ya sea para aplicarlas a todo el objeto, o a ciertas áreas que el diseñador elija.
* **Herramienta de Clipping**: Con ella, podemos deformar los elementos para crear diferentes formas.

Hay que destacar que los elementos de este editor son cubos, por lo que debemos desarrollar el mapa completo a partir de ellos, aplicándoles las texturas y deformaciones que se deseen por medio de las herramientas mencionadas anteriormente.

## Editores No Oficiales

Minecraft Editor

Bomberman

## Editores De Desarrollo

## UbiArt Framework

La idea principal de crear UbiArt Framework [2] es mitigar el complejo proceso de desarrollo de un videojuego y poder realizar una gran producción con alta calidad más rápido de lo normal. Tal y como detallan es un “framework para artistas”, ya que plantea el desarrollo de un videojuego de cara a que un artista diseñe ciertos assets y simplemente los introduzca en el editor, encargándose este de poder transformarlos y ajustarlos a las necesidades del nivel.

En lo referente al desarrollo de niveles con este fremework, ofrece la posibilidad de realizar modificaciones al nivel en tiempo real, es decir, partimos de un terreno básico predeterminado con nuestro personaje, el cual ya podemos controlar, y podemos realizar las modificaciones pertinentes a transformaciones de terrenos o inclusión de assets pausando el juego, de forma que los cambios realizados tengan una mayor exactitud, y reduzca el tiempo de testing para comprobar si bloqueamos el flujo o anulamos el acceso a ciertas zonas por imposibildad de acceso debido a la lejanía de plataformas.

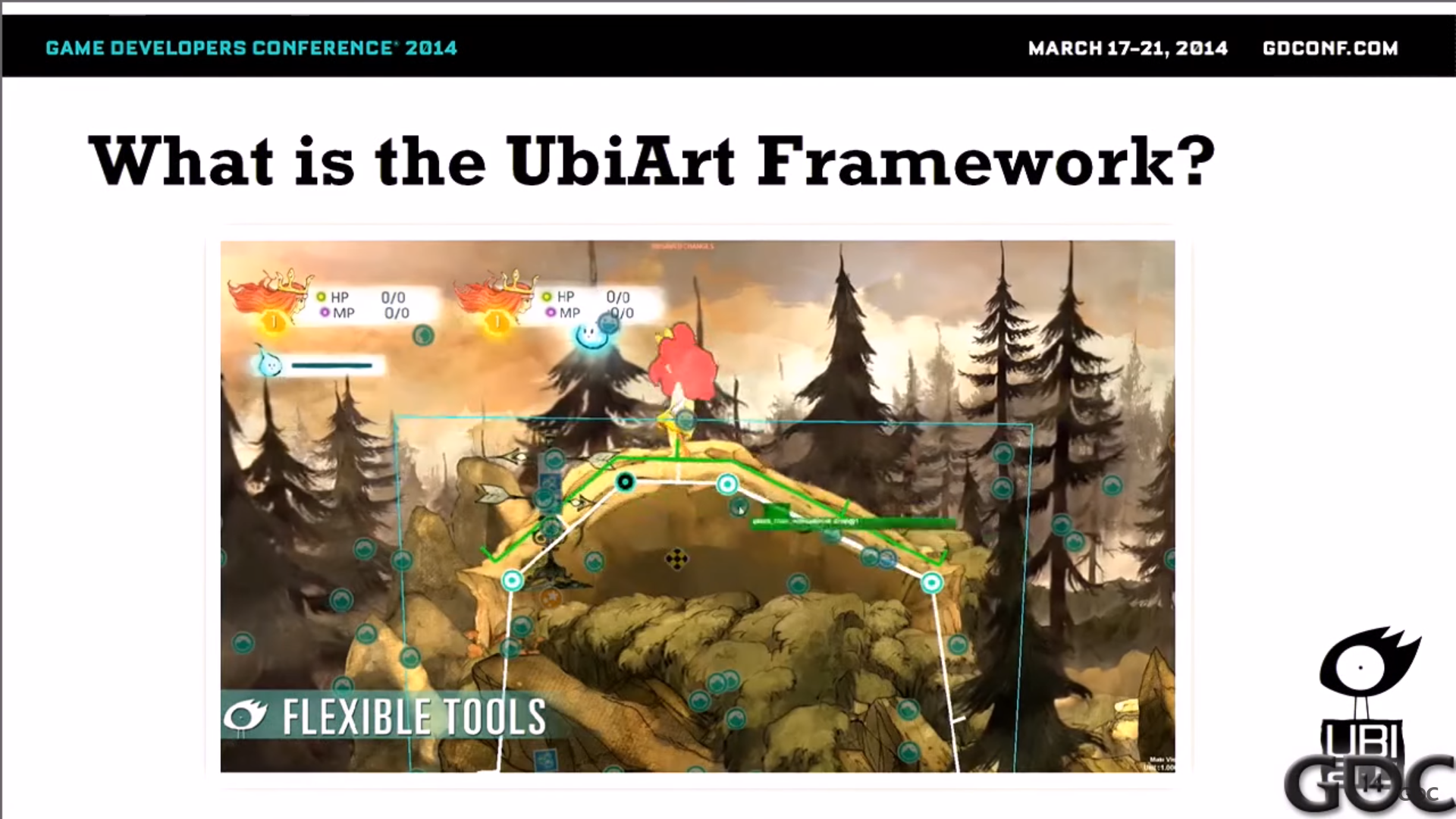


Ilustración 4 - UbiArt Framework

También es posible crear niveles sin ningún tipo de asset en la escena, es decir, los diseñadores desarrollan estas escenas utilizando “White Boxes”, con todas las mecánicas necesarias para superar el nivel, y comunicándose con los artistas, estos deciden cómo crear los assets pertinentes para los elementos de dicho nivel.

Pero a pesar de los esfuerzos de la empresa, este framework ha caido prácticamente en el olvido debido a su dificultad de aprendizaje, y la necesidad de invertir una cantidad de tiempo excesivo en enseñar al resto de los trabajadores cómo utilizarlo, aunque a pesar de ello, se llevaron a cabo videojuegos con una gran acogida, como Child of Light, Valiant Hearts: The Great War o Rayman Origins.

## RPG MAKER

Se trata de un software de creación de videojuegos, generalmente de estilo RPG tal y como su nombre indica. Fue creado en 1992 con el nombre de RPG Tsukūru Dante 98, pero con los años, diversas versiones fueron lanzadas al mercado hasta llegar a la actual, RPG Maker MV.

Este software permite a los usuarios crear mapas a partir de Tiles predeterminados, o introducir sus propios diseños. También contiene un gestor de eventos que da dinamismo al juego posibilitando la introducción de animaciones, diálogos, y transiciones. Y finalmente, en un ámbito más profesional, en las versiones de PC se ofrece un editor de Scripts que permite cambiar por completo las mecánicas establecidas por el editor y desarrollar juegos más elaborados.

Anteriormente se utilizaba RGSS (Ruby Game Scripting System), un lenguaje de programación derivado de Ruby (lenguaje interpretado y orientado a objetos). Actualmente, con la versión de RPG Maker MV, es posible crear nuestros propios plugins utilizando Javascript, lo que ha facilitado enormemente a los desarrolladores el poder realizar modificaciones a través de código debido a que es uno de los lenguajes más utilizados actualmente.

La interfaz apenas varía entre versiones, y los elementos suelen ser comunes. Por un lado, tenemos el menú con opciones generales. Como podemos observar, los objetos se separan por finas líneas verticales atendiendo a su funcionalidad, y de esta forma se crean grupos para el ojo humano. Una de estas secciones es la que nos permite alternar entre modificaciones de terreno y eventos. Si la opción de terreno está activada se iluminan un conjunto de elementos, indicando al usuario que son herramientas de modificación. También podemos acceder a las opciones de bases de datos, personajes o plugins entre otras (últimos elementos del menú).

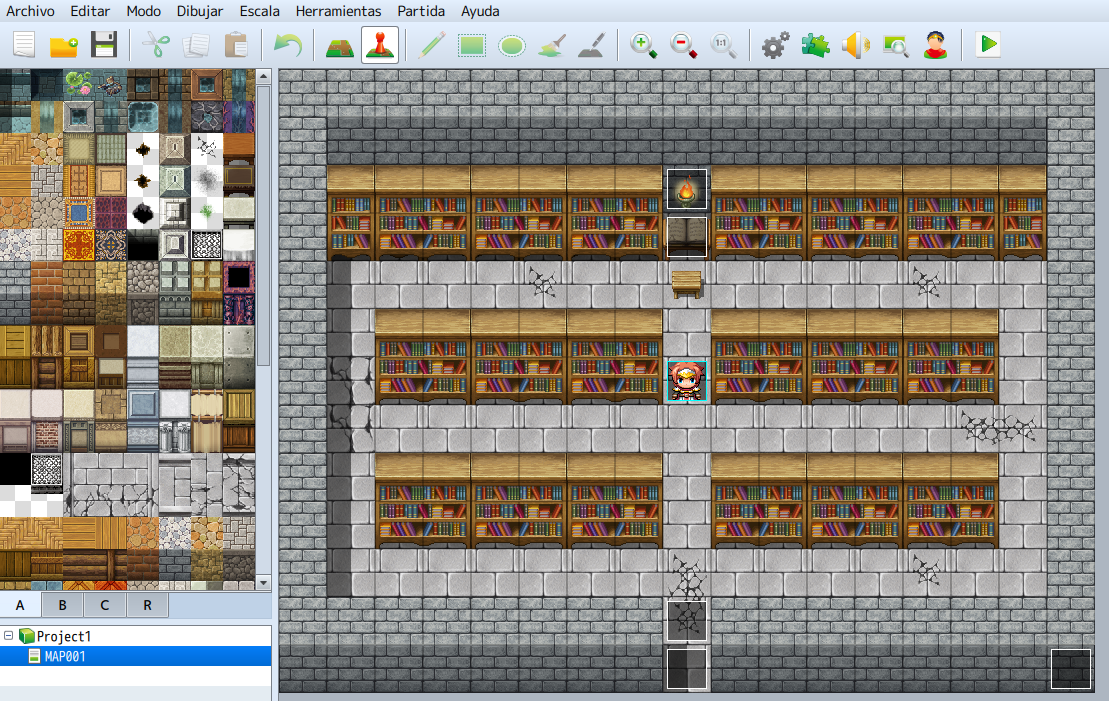


Ilustración 5 - RPG MAKER MV

En el lado izquierdo, se encuentran los tiles con los que podemos rellenar el mapa. Estos tiles se dividen en profundidad utilizando tres capas, A, B y C. A través de estas opciones, podemos crear un mapa completo. Debajo de estos tiles, se encuentran los mapas creados en nuestro proyecto, a los que podemos acceder para determinar opciones concretas del nivel (música por defecto, zona de combate, tipo de escenario…).

Finalmente, la zona que abarca el resto de la pantalla es el escenario en cuestión. Inicialmente se parte de un tablero con las dimensiones establecidas al generar el mapa, y a través del menú de tiles, el usuario lo rellena a su gusto, pudiendo incluir eventos en él.

## TILED

Editor de desarrollo no oficial.

## Apartado de mercado actual

Actualmente, encontrar videojuegos con editores de niveles incorporados es algo inusual, pues debido al rápido desarrollo de diferentes videojuegos, así como sus secuelas, los videojuegos (normalmente AAA), tienden a tener una vida media de dos años. Esto no significa que pasado este tiempo estos videojuegos se descataloguen o no sean comprados, sino que a esas alturas ya ha aparecido una segunda parte o algún juego similar en el mercado, lo que mantiene la vida, no del videojuego en sí, si no de la temática o la saga.

Aún así, algunas empresas, debido al éxito obtenido con un juego concreto, deciden mantenerlo vivo a través de la comunidad, y de este modo ofrecen editores de niveles a los usuarios y un workshop donde poder compartir los niveles con el resto de la comunidad. Un claro ejemplo es Left 4 Dead 2, lanzado al mercado en 2009, y a día de hoy, diez años después sigue teniendo una comunidad activa que desarrolla niveles y modos de juego para ser compartidos de forma gratuita con el resto del mundo.

Pero sin lugar a duda, estas oportunidades son brindadas con mayor frecuencia en videojuegos 2D con modo multijugador, ya que su desarrollo es más sencillo, tanto para empresas de videojuegos como usuarios, y al ser multijugador, no es necesaria la existencia de una gran comunidad para mantenerse, pues con pequeños grupos de usuarios que jueguen juntos es suficiente.

Por otra parte, siempre ha existido el desarrollo independiente de lo que se denominan Mods, y un claro ejemplo es Minecraft. Este videojuego se encuentra en constante desarrollo y recibe actualizaciones oficiales prácticamente cada mes, pero desde su lanzamiento al mercado como versión Alpha, ha recibido modificaciones por parte de usuarios que incluso han llegado a ser incluidas en las versiones oficiales. Ahora bien, dentro de los videojuegos 2D, debemos destacar el desarrollo de mods de la comunidad de The Binding of Isaac, el cual ha obtenido una gran popularidad en los últimos años y que ha generado una enorme comunidad que desarrolla diversos mods para añadir dificultad al juego o incluso facilitar a los jugadores la posibilidad de completar un nivel.

Además, a diferencia de Minecraft, The Binding of Isaac, al encontrarse en la tienda de Steam, ofrece mayor facilidad para la publicación y descarga de mods, ya que esta plataforma ofrece estos servicios para la comunidad. Sin embargo, aunque la comunidad de Minecraft es enorme, no existe una plataforma oficial para la descarga de Mods, aunque si grandes páginas web que contienen una gran cantidad de mods para las diferentes versiones.

## Apartado de estado del arte

Actualmente y de forma un tanto genérica, podemos ver que los editores y toolbox han ido desapareciendo, dando paso a los generadores procedimentales, que utilizan técnicas de creación de datos con algoritmos. En el caso de gráficos por computador, es habitual encontrarlo en la creación de texturas, pero si pasamos al plano de los videojuegos, podemos encontrar estas herramientas para la generación de contenido o aleatoriedad, produciendo experiencias de juego poco predecibles.

Una de estas herramientas es GauGAN, herramienta desarrollada por Nvidia y basada en el uso de redes generativas antagónicas. Se apoya en la técnica conocida como ‘normalización espacialmente adaptativa’, que genera imágenes fotorrealistas a partir de un diseño semántico del usuario, es decir, a partir de un diseño básico donde cada color representa un mismo objeto o material.



Ilustración 6 - GauGAN en acción

De esta forma, al realizar una simple viñeta de un ambiente, podemos cambiarla automáticamente por otra al cambiar los objetos asignado a los colores que se encuentran en la escena. Hay que destacar, que el mecanismo utilizado por la herramienta funciona de forma que existen dos redes neuronales, una encargada de generar imágenes, y otra de discriminar cuales son o no lo suficientemente realistas.

También decir que al ser una herramienta entrenada con un modelo de Deep learning con millones de imágenes reales, es capaz de determinar muchos detalles tales como reflejos, los cuales si no aparecen (aunque debiesen), la segunda red neuronal descartará dicha imagen hasta que la primera los muestre.

En términos de nivel, podemos encontrar cientos de juegos que utilizan la generación aleatoria para crear sus niveles, como puede ocurrir en Minecraft, donde el mundo se genera de forma automática a partir de una “semilla” y es aparentemente infinito, o No Man’s Sky, muy conocido por su algoritmo de generación de mundos, así como otra generación procedural diferente tanto a texturas como niveles, que es el audio.

No Man’s Sky utiliza un sistema procedural de generación de sonidos, de forma que muchos de sus audios ambientales o sonidos no están pregrabados, y se generan a medida que se desarrolla el juego. Estos sonidos se generan en base a unas variables programadas utilizando técnicas de sintetización.

Estas variables pueden ser la temperatura del planeta en el que nos encontremos, la velocidad de movimiento, o incluso las proporciones de la criatura que nos encontremos, de forma que la longitud de su cuello influya considerablemente en el sonido que se vaya a generar para darle una voz más característica.

# Capítulo 2 Objetivos

## Descripción del problema

Actualmente, existen diversos géneros de videojuegos, de forma que la experiencia para el usuario varía en función de las mecánicas y posibilidades que cada uno de ellos ofrece. A pesar de la gran variedad existente en la actualidad, se buscaba una forma de ampliar, tanto la experiencia como la vida útil del videojuego, y para ello se introdujo la idea de la generación de niveles por parte de los usuarios, de forma que se crease una comunidad propia que mantuviese activo el juego.

Con esto, se plantea la creación de un toolbox que permita alcanzar dicho objetivo, facilitando los assets necesarios para poder crear un nivel funcional. Para alcanzar este nivel de desarrollo, es necesario crear previamente un videojuego básico con diversas mecánicas y elementos visuales, por lo que el primer paso es crear dicha base.

Una vez planteado y desarrollado el videojuego, se puede proceder a la realización de la parte del editor empleando el desarrollo previo, y aumentando las capacidades ajustadas a la edición de un nivel similar a los creados en la parte del videojuego.

## Objetivos

* Desarrollo de Mecánicas: Se deben generar suficientes mecánicas de manera que sirvan para la creación de un juego simple.
* Búsqueda de Assets: Es necesario plantear una estética concreta para el editor, y para ello deben encontrarse o crearse assets para dicho estilo.
* Generación de niveles test: Creación de niveles simples e independientes que se utilizarán para probar las mecánicas y comprobar si el estilo elegido es el correcto.
* Maquetación UI del Toolbox: Se planteará una posible solución a la interfaz que se utilizará en el Toolbox, atendiendo a las Leyes UX mencionadas en este documento.
* Programación de Mecánicas del Editor: Deben crearse nuevas mecánicas de interacción con el editor, como transformaciones, almacenamiento de datos, etc.
* Desarrollo de nivel funcional: Una vez terminado el editor, debe crearse un nivel con él y comprobar su funcionamiento de manera que sea posible establecer si el Toolbox se ha finalizado, o por el contrario necesita alguna modificación.

## Estudio de alternativas

Dado que este estudio se podría haber realizado de otra manera, debo destacar algunos de los planteamientos iniciales para su desarrollo, así como la explicación de las decisiones tomadas.

En primer lugar, se planteó la opción de desarrollar un Toolbox partiendo de nada, es decir, creando un desarrollo de interfaz y base de código propia utilizando otros lenguajes (C++, JAVA, Python…). Esta idea fue descartada originalmente debido a que realizar esta tarea suponía realmente crear un motor de videojuegos debido a la necesidad de incluir mecánicas y restricciones desde un comienzo, así como el tratamiento de datos y funcionalidades. Esto fue descartado debido a que este desarrollo suponía una inmensa cantidad de trabajo que nos alejaba de la idea principal del estudio.

Una vez descartado esto, se planteó el uso de dos motores de videojuegos para su desarrollo: Unity y Unreal Engine.

El uso de Unreal Engine permitiría simplificar la programación utilizando el método de Blueprints, lo que podría traducirse en menor tiempo de desarrollo frente a un enfoque más elaborado en lo que respecta a la parte gráfica. Pero esto no era del todo correcto, dado que, debido a los pocos conocimientos técnicos de este software, así como de la parte de desarrollo gráfico debido a mi principal enfoque en la programación, habría supuesto un mayor esfuerzo de aprendizaje con, lo que posiblemente fuese un resultado menos elaborado.

Por tanto, la decisión quedó clara: utilizar Unity. Puesto que mis conocimientos en la Programación Orientada a Objetos son bastante amplios, la familiaridad con la que me encuentro trabajando en Unity es elevada y la facilidad que ofrece el desarrollo de cualquier programa básico con este software nos ofrece una amplia documentación tanto oficial como en lo referente a la comunidad, Unity fue el software idóneo para comenzar a trabajar en el estudio. Además, al ser multiplataforma, pueden crearse versiones para los diversos sistemas operativos actuales, ampliando así el alcance del editor para los diferentes tipos de usuarios.

## Metodología empleada

Sistema de reuniones (cada dos semanas reunión, cumplir objetivos, etc, etc). Hemos especificado al principio, como se ha desarrollado todo, etc.

## Planificación

Tiempo en desarrollo de cada cosa: Mecánicas, pruebas, estilos…

# Capítulo 3 Marco teórico

|  |
| --- |
| SE INCLUYE COSAS NECESITADAS QUE NO ME HAN ENSEÑADO EN LA CARRERA |

## Juegos de Plataformas

El videojuego elegido debía ser en 2D, ya que, a nivel de usuario, es más sencillo y entretenido la creación de niveles de este tipo que cualquiera 3D, ya que la idea de trabajar con tres ejes, múltiples puntos de vista y complejas configuraciones de mecánicas pueden hacer tan tediosa la edición de mapas hasta el punto de obviar la posibilidad de utilizar el editor.

En lo referente al tipo de videojuego, se eligió la temática de plataformas debido a las múltiples posibilidades que ofrece en cuanto a la diversidad de niveles, ya que al introducir las mecánicas (de las cuales se habla más adelante), el usuario puede ser capaz de crear un juego con objetivos completamente diferentes a los que en un principio se plantearon.

Concretamente, los objetivos de este juego principal son los siguientes. Se plantea un personaje principal con capacidad para utilizar diversas armas, lo cual amplía en gran medida la jugabilidad (ataque físico, a distancia...). Este personaje se mueve por un mundo repleto de enemigos de diversos tipos, y su objetivo es alcanzar la meta. Para ello, deberá desplazarse lateralmente por diferentes niveles, utilizando su capacidad de salto y sus habilidades para, o bien derrotar a los enemigos que encuentre por el camino, o simplemente llegar al final.

A lo largo del juego, el personaje se encontrará con diferentes puertas que le llevarán a otros niveles, de forma que el recorrido no sea únicamente lineal, sino que ese desplazamiento otorgue la sensación de profundidad.

Tabla 1 - Tabla de Marco Teórico 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Leyes de UX

Llevarme LEYES AL DESARROLLO Y DEJAR ESTO EN MEDIA/UNA PAGINA EXPLICANDO PORQUE ES IMPORTANTE TENERLAS EN CUENTA

De cara al desarrollo del Toolbox, es importante realizar un estudio previo a nivel de interfaz, dado que dicho editor no será utilizado exclusivamente por desarrolladores, sino que el objetivo es que llegue a cualquier tipo de usuario (con mayor o menor experiencia en videojuegos), que ya haya utilizado la aplicación.

Para ello, un buen comienzo sería la aplicación de las leyes de UX.

## Mecánicas

Tabla 3 - Tabla del Marco Teórico 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## Game Design vs Level Design

Introducir información de referencias bibliográficas, páginas web, etc.

Cuando hablamos de Game Design, debemos tener en cuenta que es el desarrollo del aspecto más fundamental de un juego, las reglas. En esto, debemos destacar que no hace referencia ni a la parte de programación, ni tampoco a la creación de contenido, como podría ser modelado de objetos 3D, la parte del arte, efectos de sonido, guion… En definitiva, se trata de la creación de mecánicas adaptadas a nuestro videojuego, aunque en ciertas ocasiones, en este campo puedan plantearse ideas de entornos, historias, o personajes, pero claramente, no es el Game Designer quien lo desarrolla, sino quien lo plantea.

Keith Burgun opina que el término “videojuego” ha tenido un impacto negativo en lo referente a los juegos y el Game Design. Según él, dicho término se emplea en cualquier entretenimiento digital interactivo sin importar la naturaleza del sistema, llegando incluso a ser sinónimo de “juego”, de forma que los puzles, juguetes o simuladores son todos ellos juegos. Esto radica en una perdida de ciertos sistemas concretos que se engloban en un conjunto al que no pertenecen (Como un simulador y un puzle), lo que genera problemas a la hora de crear ciertos estándares de desarrollo, ya que ¿cómo podemos generalizar dos cosas que no tienen nada en común, pero son referidos con el mismo término?

Este planteamiento que diferencia “juego” de “videojuego” es importante de cara a los Game Designers, ya que son ellos los encargados de realizar unas correctas especificaciones de su desarrollo con el fin de evitar confusiones entre otras personas del gremio, o simplemente aquellas que desean aprender acerca de este campo.

Pero, volviendo al Game Design, cuando se comienza a desarrollar un videojuego, debemos plantear preguntas de algunos puntos clave, que permitan un desarrollo concreto para el resto del equipo.

Metas Externas

Metas de Gameplay

Gameplay Emergente\*\*\*\*\*

Historia Narrativa

¿Se va a incluir una historia? ¿Esta historia va a ser lineal, temporal, atemporal…? ¿Va a existir un Lore? Son entre otras, las preguntas iniciales con respecto a este aspecto de un videojuego. Debemos partir de la definición de historia como “*Narración secuencial de eventos, reales o ficticios enfocados al entretenimiento*”. Pero esta definición en muchos casos no es acertada, como puede ser con el concepto de Lore. Este concepto hace referencia a lo que podríamos traducir como intrahistoria, es decir, eventos que ocurren sin ser narrados directamente.

Este tipo de narración es muy común en los videojuegos actuales, ya que lo consiguen por medio de diálogos con personajes secundarios, elementos visuales dentro del nivel (un castillo en ruinas puede implicar que es muy antiguo o que hubo una gran batalla), etc. Es muy útil introducir este tipo de historia, dado que, por un lado, amplia el tiempo de juego para los usuarios, motivándoles a explorar el mundo completo. También nos permite reducir el guion y ofrecer a los jugadores la posibilidad de dejar volar su imaginación al crear teorías que pueden coincidir con la idea principal o ser completamente distintas (“*en este castillo hubo una gran batalla por el poder de las tierras que lo rodean, pero tras finalizar la batalla no hubo ganador, pues el castillo quedo inhabitable*”).

En cuanto a si la historia va a ser lineal o no, será útil en el avance del videojuego en cuanto al concepto de niveles (de lo que se hablará más adelante). Una historia lineal, suele implicar que el jugador tendrá una mayor limitación en cuanto a explorar el universo del videojuego, siendo el avance de principio a fin, pasando por puntos intermedios determinados. Es prácticamente lo mismo que contar una historia, es decir, el narrador la cuenta sin desviarse del objetivo final.

En el otro extremo a las historias lineales, encontramos que el jugador no necesariamente debe pasar por puntos intermedios para llegar al final, sino que puede seguir los objetivos que quiera hasta finalizar la historia. También existe un punto intermedio en el que el jugador sí debe pasar por algunas partes de la historia para poder avanzar en ella, pero para acceder a ellas puede hacerlo de diversos modos.

Es un concepto bastante complejo, tanto entenderlo como explicarlo, pero la siguiente imagen lo resume de una forma clara y concisa.

****

Ilustración 7 - Representación de la historia en videojuegos

Género

¿Qué tipo de videojuego voy a crear? ¿Va a ser de acción, estrategia, disparos…?

Antes de lanzarse al desarrollo de cualquier videojuego, algo esencial es especificar el genero del juego, ya que, si lo que queremos es crear un simulador de vuelo, no necesitaremos una mecánica de salto. Para ello, se procede a explicar algunos de los géneros más comunes con una visión superficial de ellos, debido a la complejidad de todos ellos, así como la profundidad de estudio.

En los orígenes de los computadores, y debido a los limitados recursos gráficos, los videojuegos de aventura podrían considerarse los pioneros en este campo, ya que este género no solo hace referencia a los que todos pensamos (The Legend of Zelda, Assassins Creed, etc.) considerados Acción-Aventura, sino que también engloba aventuras conversacionales y gráficas.

Las aventuras conversacionales fueron muy populares debido a las posibilidades que ofrecía. Generalmente, el ordenador se encargaba de describir los hechos por texto en pantalla, y el jugador, por medio del teclado, especificaba la acción a realizar, desembocando en otra descripción. A medida que el mundo de los gráficos avanzaba, este género evolucionaba, siendo su siguiente paso el de incluir imágenes del entorno, eliminando así las descripciones textuales. Tal fue la popularidad, que llegaron a crearse videojuegos con imágenes de vídeo, lo que se denominó película interactiva.

Este género, con el avance de la tecnología derivó en las aventuras gráficas, que utilizaban el mismo concepto, pero en vez de utilizar descripciones textuales, incluían un personaje que se desplazaba con mecánicas Point & Click. De este modo, utilizando el ratón, los jugadores podían desplazarse por un entorno, recoger y utilizar objetos, interactuar con personajes, etc.

Tras explicar los conceptos generales del Game Design, es hora de hablar del Level Design. Si bien es cierto que es creído que el Level Design surge del Game Design, y por tanto es un derivado de este, vamos a intentar demostrar que ambos se encuentran en el mismo escalón a nivel de desarrollo, y que uno no puede existir sin el otro. Al igual que antes, definir directamente el Level Design es complicado, por lo que partiremos de conceptos y funciones de este campo para poder explicarlo claramente.

Retomando los géneros de videojuegos, podemos plantear la idea a partir del deporte. En los deportes, generalmente ya tienen establecidas tanto las normas de juego como los escenarios (fútbol, baloncesto…), pero en algún momento de la historia, alguien tuvo que definir las medidas del campo de juego para que la competición sea justa para todos los participantes.

Por otro lado, tenemos los juegos de mesa, muy cercanos en cuanto a diseño a los videojuegos. Estos juegos no precisan de una alta representación del entorno (véase el tablero de ajedrez), sino que se centran en aproximar la realidad o incluso diseñar de forma abstracta. En el ajedrez, el juego representa una guerra entre dos ejércitos, pero no para ello se crea un campo de batalla con trincheras, sino que se representa el avance de las tropas por medio de casillas.

Un ejemplo de un gran diseño de nivel es la máquina de Pinball. Todas ellas diferentes, con una mecánica básica, los botones mueven las palancas para empujar la bola de nuevo al tablero. Pero no es la mecánica lo que prima en este análisis, sino la disposición de los elementos que permiten a la mola rebotar de uno a otro, generando una aleatoriedad de desplazamiento que hace el juego poco repetitivo y dinámico.

Con este ejemplo, queda claro que, en muchas ocasiones, el Level Design no depende solo del Game Design, ya que, sin el escenario concreto de Pinball, el Game Design sería demasiado básico. Por ello, y a lo que se pretende llegar, es que ambas ramas de diseño no se encuentran, como se decía anteriormente, en diferentes escalones, sino que las dos se encuentran al mismo nivel, y deben trabajar conjuntamente para poder desarrollar, ya no solo un videojuego, sino cualquier sistema de entretenimiento.

En relación con el ToolBox desarrollado, es posible equipararlo a un motor de videojuegos, ya que ambos tienen aspectos comunes en lo referente a la creación de un videojuego.

Por un lado, tenemos la visualización del mundo. Generalmente, en un motor de videojuegos solemos tener una vista tridimensional y otra bidimensional con una proyección ortográfica, pero en el caso del ToolBox, se ofrece una vista bidimensional frontal que representa el mundo en el que se producirán los eventos.

Otro aspecto destacable es la navegación, ya que, si un motor ofrece la posibilidad de acercarse a un objeto, o rotar sobre él, en el ToolBox optamos por escalarlo, rotarlo o desplazarlo, puesto que al ser 2D, no es necesario realizar ajustes de cámara para transformar dicho objeto.

# Capítulo 4 Especificación

|  |
| --- |
| A continuación, se procede a realizar un pequeño resumen con los diversos programas utilizados para el desarrollo del trabajo en todos los ámbitos, programación, diseño, organización y gestión del proyecto. |

## Unity 3D

Unity es un motor de videojuegos multiplataformas desarrollado por Unity Technologies. Puede utilizarse en tres plataformas, Windows, OS X y Linux. Recibe la calificación de multiplataforma debido a que es posible crear una aplicación y exportarla a diversas plataformas (WebGL, Android, IOS, Windows…).

Unity surgió tras el desarrollo de un juego denominado GooBall, el cual no obtuvo beneficios. A pesar de ello, sus desarrolladores decidieron potenciar el motor empleado para la creación del juego de forma que cualquiera pudiera utilizarlo. De esta forma, los desarrolladores independientes sin capacidad para crear su propio motor de juego u obtener diversas licencias comenzaron a utilizarlo, otorgando el éxito que hoy día tiene el motor.

Ofrece a los programadores utilizar el lenguaje C#, un lenguaje orientado a objetos fácil de utilizar para aquellos con pocos conocimientos, y además lo hace por medio de scripting basado en Mono, una implementación de código abierto de .NET Framework.

Anteriormente, los lenguajes de programación podrían ser JAVASCRIPT y C#, pero tras el lanzamiento de una de sus versiones, quedó únicamente C#. Además, Unity incluía MonoDevelop como entorno de desarrollo, pero junto con la eliminación de JAVASCRIPT, se eliminó, dando paso a Visual Studio como entorno predeterminado. De esta forma, se facilita el modo Debug, conectando el programa con el motor, de forma que la depuración es mucho más sencilla, el uso de puntos de parada, la obtención de datos durante la ejecución, etc.

Además, por parte de los diseñadores, les ofrece la posibilidad de desarrollar diversos niveles de forma sencilla gracias a sus herramientas, así como a los animadores y artistas, ya que ofrece herramientas de animación para crear las mismas directamente en el software, o exportando animaciones, crear a través de una máquina de estados los diferentes cambios de animación de uno de los personajes. También ofrece múltiples elementos de Interfaz de Usuario para poder utilizarse en diversos aspectos (Menús, Opciones, Registros, Páginas Web…).

Este motor puede también trabajar de forma conjunta con diversos programas utilizados por los artistas, tales como Blender, 3DS Max o Maya entre otros, permitiendo la modificación de objetos editados en alguno de estos programas, actualizándose los cambios automáticamente en el proyecto de Unity que los contiene, y de esta forma, se evita la importación manual tras cada modificación de dichos objetos.

La elección principal de el uso de este programa en el desarrollo de este trabajo no es solo por las facilidades que ofrece de cara a un desarrollo individual, sino también porque es una de las herramientas más utilizadas en el ámbito del desarrollo de videojuegos a nivel profesional, lo que ofrece una oportunidad de plantear un proyecto, no solo como reto educativo, sino también como reto profesional.

Por otra parte, dado que Unity es una buena herramienta para el desarrollo de ciertas aplicaciones 2D, el hecho de plantear un editor de niveles con este programa ha sido la mejor opción tras descartar diversas plataformas de desarrollo.

Para el desarrollo de este proyecto, se ha utilizado la versión 2018.3.2f1, dado que se utilizó la versión más reciente del motor en el momento de su inicio, y tras los enormes avances llevados a cabo en él, cambiar de versión podía ser peligroso de cara a nuevos errores por ciertas actualizaciones del motor.

## C#

Se trata de un lenguaje de programación orientado a objetos desarrollado por Microsoft como parte de la plataforma .NET similar a JAVA. Deriva de C/C++ incluyendo mejoras derivadas de otros lenguajes.

Es el lenguaje de programación base de las últimas versiones de Unity, lo que hace que su uso sea obligatorio en el desarrollo del proyecto. Además, ofrece múltiples posibilidades a la hora de programar, como el uso de Threads que permitan la ejecución simultanea de bloques de instrucciones, o la gran cantidad de funciones predefinidas que facilitan la interacción con clases objeto, estructuras de datos, y eventos.

Dado que Unity funciona por medio de Scripts, no es necesaria la organización del código por paquetes, sino que cada clase actúa por separado. Claro está, que, en el caso de necesitar una relación entre diversas clases, es posible la creación de scripts que funcionen como una clase objeto, de forma que no sea necesaria la interacción directa con el motor para acceder a ellas.

También debemos destacar la presencia de corrutinas, funciones que nos permiten detener su ejecución, devolviendo el control a Unity. Esto es útil cuando por ejemplo tenemos que esperar un tiempo determinado para continuar la ejecución, o en algunos casos, necesitamos un tiempo de espera para realizar comprobaciones o conexiones.

C# a su vez ofrece una librería para poder trabajar

## Visual Studio 2017

Es un entorno de desarrollo integrado para Windows que soporta lenguajes como C#, C++, JAVA o PHP. Visual Studio permite crear sitios y aplicaciones web y servicios web en cualquier entorno con soporte .NET. De esta forma, se pueden desarrollar aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, móviles… Además, ofrece asistencia en tiempo real según se escribe el código, otorgando la función de autocompletar que muestra posibles códigos a escribir. También, ofrece información acerca de funciones o variables para inspeccionar las definiciones, o ver los errores cometidos a la hora de programar.

Actualmente, se instala por defecto con Unity, por lo que la configuración para el modo Debug viene establecida por defecto, lo que facilita la depuración de código activando esta opción. También contiene asistencia de autocompletado de código enlazado a las librerías de Unity, no solo con nombres de funciones, sino con una extensa API que nos informa de la función del método, los parámetros posibles, y en caso de errores por tipo de datos, ofrece la posibilidad de aplicar una solución automática, tanto de conversión de datos, como de modificación de variables.

## GitHub

Github es una plataforma de desarrollo colaborativo que permite el alojamiento de proyectos de diversos tipos. Principalmente se utiliza para almacenar programas en desarrollo de diversos lenguajes de forma pública, aunque también ofrece un plan de pago para privatizar los proyectos.

Es una buena forma para compartir proyectos y realizar un desarrollo multiusuario con pocos problemas, ya que ofrece soluciones a la hora de la publicación simultanea de archivos modificados por varios usuarios, de forma que sea posible decidir qué cambios se aplican al fichero.

Además, Github ofrece una plataforma para proyectos web que permite publicar las versiones a modo de uso online, sin necesidad de probarlo en local.

También debemos destacar que GitHub almacena cada subida a la plataforma, lo cual permite que en caso de ser necesario un retroceso a una versión anterior a la que actualmente se encuentra en la nube, podremos hacerlo con una simple restauración, sin eliminar los cambios futuros a dicho punto.

Github trabaja con ramas, es decir, cuando realizamos la primera subida, todos los datos se enlazan a una rama (generalmente la denominada máster), y si no realizamos ninguna modificación al respecto, cualquier subida permanecerá en ella. Pero es posible la creación de diversas ramas para el desarrollo, lo cual permite realizar modificaciones de fragmentos del proyecto a partir de un punto de la rama principal, de forma que cualquier modificación en cualquiera de las ramas no modifique ninguna de las otras.

Así pues, con esta herramienta, es posible realizar un trabajo paralelo sin verse afectado por los errores del desarrollo actual de otra rama, hasta que los desarrolladores crean conveniente y fusionen la rama secundaria con la principal, uniendo el proyecto de nuevo.

## Trello

Trello es una aplicación de administración de proyectos que utiliza el sistema Kanban para el registro de actividades con tarjetas. Además, permite organizar tareas, crear listas, adjuntar archivos, gestionar tiempos y usuarios dedicados a tareas…

Cuenta con una versión web y una aplicación móvil lo que facilita el acceso desde cualquier lugar a los tablones, así como la gestión de las posibilidades anteriormente mencionadas.

Ha sido utilizado en este proyecto con el fin de mantener un desarrollo organizado, un control de dificultad/tiempo en el avance de las diversas tareas, y un lugar donde poder almacenar las diferentes ideas surgidas a lo largo del desarrollo de la aplicación.

Trello se organiza en tableros. Un tablero puede contener listas con títulos, a las cuales añadimos tarjetas. Son estas tarjetas donde realmente se especifican las tareas, ya que en ellas se puede especificar un título, una descripción genérica de la tarea, y diversas opciones más, entre las que encontramos:

* **Etiquetas**: Permiten organizar tareas por la función que especifiquemos a dichas etiquetas (desarrollo UI, seriealización, diseño de assets…).
* **Miembros**: Con esta opción, es posible añadir diferentes usuarios autorizados en el tablero a las tareas, de forma que se controle quién se encarga de cada parte del desarrollo.
* **Checklist**: Es útil para especificar objetivos de la tarea, de forma que una vez completado dicho objetivo, se marque como terminado, y podamos observar el porcentaje desarrollado de la tarea.
* **Vencimiento**: Este campo nos permite establecer un deadline para la tarea, y así controlar el tiempo empleado en ella, comprobar si la planificación del tiempo ha sido correcta, demasiado extensa o por el contrario demasiado breve.

En este proyecto, la planificación se realiza en diversas listas. La primera de ellas es el denominado cajón desastre, en el cual se encuentran ideas para futuras implementaciones, errores a solucionar en un futuro, etc. Después encontramos las listas por hacer, orientadas a un objetivo concreto (desarrollo del editor, de las mecánicas, de assets…), que contienen tareas primordiales que aún no se han comenzado y que se deben realizar cuanto antes.

Tras estas listas, tenemos haciendo, una lista específica para las tareas que están en desarrollo. Es importante destacar que a ella deben destinarse las tareas en el momento en que comienza su desarrollo, tanto para controlar lo que se está haciendo, como para que, en caso de la existencia de otros usuarios, sepan que esa tarea ya no puede tocarse.

Finalmente, las listas de terminado, que se dividen por mes, de forma que también se mantenga un control, no solo sobre el tiempo de desarrollo de una tarea, sino también del grueso de trabajo finalizado de forma mensual.

# Capítulo 5 Descripción informática

## Etapas del Desarrollo

Inicialmente, la idea de este proyecto partía de dar soporte a diversos desarrolladores de crear escenarios acordes a un tipo de juego con unas mecánicas concretas, de forma que no fueran necesarios conocimientos de programación para poder llevar dichas tareas a cabo. Por ello, se plantearon diversas etapas dentro del desarrollo del proyecto para ir formando de forma visual la idea que se tenía en mente.

## Etapa 1: Desarrollo del White Box

Para un primer planteamineto del proyecto, se diseñó un pequeño nivel en el que se colocaba un Sprite básico que se desplazaba de izquierda a derecha con el teclado sobre una superficie rectangular que ocupaba toda la pantalla. Para ello, utilizando un rigidbody colocado en el Sprite representativo del player, se añade una fuerza en función de la tecla pulsada, y además se produce una rotación de dicho Sprite para determinar la dirección en la que mira.

Una vez diseñada esta mecánica básica, se procedió a crear otro nivel denominado “Editor”, que mostraba una interfaz simple con un icono de dicho personaje y que al hacer click sobre él, se instanciaba en el centro de la pantalla un prefab asignado al script correspondiente. Al comprobar que todo funcionaba correctamente, se procedió a la inclusión de un script que permitiese escalar la superficie anteriormente mencionada, y a añadir todo ello al editor.

## Etapa 2: Busqueda de Assets

Se pensó en un juego 2D de Scroll Lateral con una estética similar a Sovel Knight, es decir, un mundo medieval ficticio donde la pelea consista en el uso de armamento cuerpo a cuerpo (espadas, hachas, martillos), así como ataques a distancia, ya sea por medio de armas o hechizos (arcos, bolas de fuego, cuchillos…).

Para ello, tras una larga investigación de diversos assets gratuitos, así como otros obtenidos a través de diversos bundles, se encontró una forma rápida y muy adaptada a la idea principal del arte, de obtener los diversos personajes que aparen en este proyecto. Esto se consiguió gracias al editor de personajes **Fantasy Heros Editor desarrollado por Hippo Games**.

A través de este proyecto realizado en Unity, podemos obtener prefabs de estos personajes con el equipamiento que hayamos elegido, su armamento y sus animaciones incorporadas, de forma que su introducción en código es simple y rápida.

Una vez planteada esta estética en los personajes, se pasó a la investigación de otro tipo de assets para los escenarios (suelos, elementos decorativos, elementos interactuables…). Esto fue más complejo, debido a que, al determinar una estética a partir de unos personajes prediseñados con un estilo muy concreto, resulta prácticamente imposible descubrir otros elementos que sigan la misma guía de estilos. A pesar de ello, se obtuvieron resultados satisfactorios con pulcros assets que encajan en el escenario al ser implementados con dichos personajes, como son los suelos, los cuales son tileables, puertas o espinas.

## Etapa 3: Mecánicas

Con los assets a nuestra disposición, y un whitebox inicial con el que poder trabajar, comenzó una fase en la que se debía incluir la mecánica de movimiento en el asset del personaje elegido. Su inclusión fue relativamente sencilla, y los cambios únicamente fueron respectivos a la animación.

Dado que un movimiento horizontal era demasiado escaso, se amplió creando una mecánica de salto y salto doble, que permitiría ampliar la experiencia de juego del usuario. También, dado que debían crearse enemigos, se incluyó un sistema de vida en el personaje y una mecánica de ataque, la cual funciona con la espada que posee al pulsar la tecla definida para realizar dicho movimiento.

A continuación, y para comprobar que la vida funcionaba correctamente, se desarrolló un script básico de daño, colocando elementos estáticos en el escenario para que dañasen al jugador. Este script sirvió para dañar tanto al personaje como para los enemigos, dado que se planteó inicialmente como un collider que busca quién está entrando en él, y, por tanto, distinguir entre player y enemigo era relativamente sencillo.

Con todo esto, se desarrolló la primera IA partiendo del movimiento básico programado en el White box, pero aplicándole a dicho movimiento una automatización. Las dos IA’s restantes siguieron al desarrollo de esta primera.

## Etapa 4: Mecanicas del Editor

Una vez se han desarrollado las mecánicas principales del juego base, se procede a la elaboración del editor como tal. Una de las primeras modificaciones realizadas a lo que ya se tenía, fue la de cambiar el modo de instanciar elementos, colocándolos en la posición del ratón en vez del centro de la pantalla. A continuación, se procedió a modificar el escalado para elementos tileables, así como la rotación y traslación de elementos ya instanciados.

Aplicando algunas de las leyes UX que se mencionan en este documento, se dispusieron los diferentes elementos en pantalla para facilitar la usabilidad del editor a nivel usuario. También, se creó un script que gestionase las transformaciones en las instancias sin necesidad de elegir una transformación cada vez que se quisiese modificar de algún modo alguno de los elementos.

Con estas transformaciones ya establecidas, se pasó al control de la escena, permitiendo que los usuarios fuesen capaces de probar el nivel que estaban editando in-time, añadiendo un botón que activase las mecánicas de todos los personajes, y se desactivasen algunos controles únicos del editor, tales como el movimiento de personajes. Sin embargo, se permite a los usuarios instanciar y arrastrar elementos a la escena, ya que en muchas ocasiones puede ser necesario realizar ajustes precisos en cuanto a la disposición de plataformas, ya sea para asegurarse de hacer zonas accesibles, o simplemente no les convencen algunas de ellas allí donde se colocaron.

## Etapa 5: Sistema de Guardado

Dado que el editor ya es funcional, y se pueden crear niveles, era el momento de almacenarlos, utilizando el sistema que se detalla posteriormente en este mismo apartado. No solo guardar era necesario, ya que era posible que el usuario decidiese cargar alguno de los niveles que ya había guardado, por lo que también se implemento el sistema de carga, aprovechando a su vez que los ficheros ya se almacenan de forma local.

## Etapa 6: Máquina de Estados

Debe entenderse, que lo que aquí se denomina “máquina de estados”, hace referencia a las transiciones entre las diversas puertas que pueden colocarse en los niveles. Se denomina de este modo, porque la transición que se realiza de una puerta 1 a una puerta 2, no necesariamente debe tener retorno, es decir, el usuario puede establecer que la puerta 1 nos desplace hasta la puerta 2, pero esta última puede no hacer nada o llevar a una puerta 3.

Inicialmente, esta máquina se creó con la restricción de solo poder comunicar 3 puertas por nivel debido a que, en un primer planteamiento a nivel de interfaz, se pretendían utilizar render textures para indicar de forma visual las puertas que se estaban estableciendo. Al incluir la transición entre niveles, se deshechó esta idea, y se simplificó con un listado de nombres, lo cual reduce en cierto modo la usabilidad, pero aumenta en gran medida la capacidad del editor.

## Etapa 7: Playtesting

Con el Editor terminado, los assets establecidos, y las mecánicas programadas, llegó la hora de pulir y solucionar los errores que podían surgir. Se llevó a cabo un profundo testeo del editor, creando múltiples niveles, centrando las creaciones en puntos concretos donde pudiese fallar, y finalmente, pasando a cargar un nivel terminado fuera del editor, es decir, en la escena que controla el juego como tal.

Este playtesting hizo realizar múltiples modificaciones, tales como cambios en los datos almacenados de los diversos objetos, cambios sutiles en las transformaciones, valores por defecto de los cálculos en las IA’s, así como la adición de restricciones que podían influir negativamente a la hora de crear un nivel o jugar uno de estos.

## Etapa 8: Inclusión de Assets Decorativos

Tras finalizar la fase de playtesting y refinar mecánicas y funcionalidades, se añadieron instancias decorativas que permitiesen a los usuarios poder vestir y decorar el nivel a su gusto, como por ejemplo tres fondos de pantalla que diesen una estética al nivel (bosque, desierto y un entorno flotante), y elementos de fondo como árboles y arbustos.

Destacar que estos assets fueron buscados en esta etapa, ya que no se pensó en ellos en la fase de búsqueda de assets, y fue algo difícil encontrar todos los que habrían encajado perfectamente con los tres tipos de escenarios anteriormente mencionados.

## Leyes UX

Tal y como se habló previamente en el apartado del Marco Teórico, se aplicaron algunas de las leyes UX en el desarrollo de la UI del editor. A continuación, se numeran dichas leyes, ofreciendo una pequeña explicación de su uso en el proyecto:

## Ley de Fitts

“El tiempo necesario para llegar a un objeto es proporcional a la distancia a la que se encuentra y su tamaño”.

Esta ley demuestra que los movimientos rápidos y los elementos pequeños tienen una mayor tasa de error debido a la relación velocidad-precisión. Por ello, en nuestro Toolbox, los elementos tienen un tamaño medio, ni muy grande, ni muy pequeño, para asegurar que el puntero se encuentra en el interior de los elementos y evitar dudas, así como su distribución en la pantalla, la cual se encuentra colocada de forma que los elementos interactuables no se encuentren desperdigados por toda la escena.

## Ley de Hyck

“El tiempo que se necesita para tomar una decisión aumenta con el número de posibilidades y su complejidad”.

Al tener una gran cantidad de opciones, el usuario puede tardar más tiempo en tomar decisiones debido a que trata de contemplar todas las opciones posibles para determinar cual es la mejor de ellas.

Por ello, aunque al usuario se le ofrecen múltiples opciones a la hora de colocar elementos en el escenario, estos se agrupan por tipo (ley de la que hablaremos más tarde), de forma que se limiten las posibilidades en ciertas situaciones concretas y se facilite la decisión a los usuarios.

## Ley de Jakob

“Los usuarios invierten más tiempo en otros sitios. Esto significa que prefieren los sitios que trabajen de forma idéntica a los que ya conocen”.

Esta ley plantea que los usuarios están acostumbrados a trabajar con ciertos patrones de diseño, por lo que, al sentirse familiarizados con los elementos, no sienten reticencia a navegar en dicha plataforma.

En el Toolbox este apartado se puede observar en los elementos orientados a la edición, como las transformaciones de objetos, salvaguardado y carga de niveles, etc. Se ha tratado de utilizar iconos, no solo representativos de la acción a la que se refieren, sino también basados en ciertos estándares utilizados por múltiples aplicaciones, no solo orientadas a videojuegos, sino a ediciones de texto, imagen…

## Ley de Región Común

“Los elementos tienden a ser percibidos en grupos si se encuentran compartiendo una zona claramente delimitada”.

Al añadir bordes alrededor de uno o varios elementos, creamos de una forma sencilla la separación de otros elementos diferentes.

En el caso de nuestro editor, se han agrupado los elementos dentro de cajas atendiendo a su funcionalidad, de forma que en una de ellas se encuentran los elementos instanciables que formarán parte del nivel como personajes, plataformas u objetos, y en otra los iconos dedicados a la parte de edición como transformaciones de objetos, guardado y carga de niveles, acceso a la máquina de estados…

## Ley de Proximidad

“Los objetos cercanos tienden a ser agrupados”.

Esta ley se refiere a que, al mostrar una ristra de elementos próximos entre ellos, se da a entender que todos ellos se encuentran relacionados de alguna forma concreta, de forma que mentalmente los usuarios los diferencian del resto.

Este es el caso de los elementos descritos en la Ley de Región Común, no solo se encuentran delimitados por un área, sino que también son agrupados dentro de dicha área atendiendo a su uso, estableciendo así las plataformas, por un lado, los personajes jugables por otro, enemigos, etc.

En el caso de las transformaciones, se agrupan dentro del área respetando una distancia entre los elementos comunes de edición y los demás, ofreciendo cercanía entre los botones de transformación de elementos del juego, y separándolos a su vez de las opciones de guardar y cargar, así como estos de la máquina de estados para el flujo entre niveles.

## Mecánicas y Funcionalidades Implementadas

A continuación, se procede a la explicación de las diversas implementaciones realizadas, tanto en lo referente a funcionalidades del editor, como a las diversas mecánicas implementadas en el juego base:

## Transformaciones

Con el objetivo de otorgar una mayor flexibilidad a la edición del nivel, partiendo de la idea principal de que Unity permite modificar la posición, rotación y escala de sus objetos instanciados, se integró este comportamiento en los objetos.

Para ello, se utiliza una clase integrada en las instancias, donde primero, se determina qué tipo de transformación está permitida, ya que no todos los elementos instanciables pueden ser transformados igualmente. Esto, principalmente, se decidió para evitar diversos errores no deseados que pudiesen ser provocados por escalar o rotar algunos elementos.

Utilizando la posición del ratón al seleccionar una instancia y su posición durante su desplazamiento, se aplican las diversas transformaciones:

## Traslación

Al mantener pulsado click izquierdo sobre una instancia, se desplazará tantas unidades como se desplace el ratón, realizando las correspondientes conversiones de coordenadas de pantalla a coordenadas del mundo.

## Rotación

Teniendo en cuenta la posición del ratón, se calcula el ángulo entre el eje x que pasa por el centro de la instancia y la posición actual del ratón, actualizándose mientras este se desplace por la pantalla.

## Escala

Para escalar los elementos, hay que tener en cuenta que algunos son tileables, de forma que al alterar el valos escala de su componente transform, no se aplicará ningún tipo de modificación al Sprite, por tanto, se debe escalar directamente el Sprite para poder sentir este comportamiento del asset. Al hacer esto, los colliders que contenga dicho objeto no se ajustarán de forma automática, por lo que también se deben realizar ajustes en ellos. Por suerte, algunos de ellos contienen el tamaño exacto del Sprite, por lo que se pueden ajustar ambos a la vez. Sin embargo, otros deben realizar los mismos cálculos que en el Sprite, y ajustarse en función de su tamaño, aunque estos cálculos son tan básicos que su precio computacional es prácticamente nulo.

## Instanciar

Para instanciar los diversos objetos en el editor, se ha diseñado una interfaz que los agrupa por categorías, donde cada elemento tiene el mismo nombre que su prefab correspondiente. Por ello, al seleccionar uno de ellos, se carga el prefab con dicho nombre en la posición en la que se encuentre el ratón, y mientras este se mantenga pulsado y se desplace por la pantalla, se desplazaran juntos hasta que se libere el ratón.

Inicialmente, los objetos se instanciaban en el centro de la pantalla y entonces se tenía que pasar al comportamiento de traslación mencionado previamente, pero por comodidad a la hora de editar y así mejorar una de las incomodidades de otros editores (la gran cantidad de clicks necesarios), se optó por este uso.

## Serialización

Para el desarrollo de la serialización, se decidió utilizar una mezcla entre las posibilidades que ofrece Unity, C# y código de uso público. Principalmente se basa en JSON Helper, una clase desarrollada por *box head productions*, que permite la serialización de múltiples datos transformándolos a formato JSON.

Utilizando esta clase, es posible almacenar conjuntos de datos del mismo tipo en diversos archivos, los cuales se almacenan en un directorio elegido por el usuario, con nombres predefinidos en función del tipo de objeto que engloba.

Así pues, es necesario crear una clase objeto por cada conjunto que almacena los datos necesarios para la carga del nivel. De esta forma, al guardar los elementos del nivel, se insertan nuevos objetos en una lista del tipo de dicho objeto, y esta se escribe en un fichero en formato JSON.

Para la carga de datos, solo necesitaremos deserializar los datos, utilizando la clase mencionada al principio. Para ello, se recogen todos los ficheros almacenados en el directorio elegido por el usuario, y uno por uno, se instancian prefabs con las modificaciones pertinentes, es decir, los datos guardados.

Con todo esto, existe una clase genérica en el editor, que contiene la clase de guardado y carga, así como una serie de listas de Objetos, correspondientes a cada conjunto de prefabs, y una lista que almacena todos los gameobjects del nivel. Así, cuando se guarda un nivel, se recorre la lista de gameobjets, se inserta cada objeto en su lista correspondiente, y luego se serializan en sus ficheros y se limpian las listas.

Al cargar un nivel, se recogen todos los ficheros y se instancian sus prefabs correspondientes con la configuración almacenada, insertándolos en la lista genérica de gameobjects, y activando las clases necesarias en el editor, y desactivando las clases que corresponden al gameplay, para evitar problemas.

## Máquina de Estados

En el desarrollo del sistema de puertas, se vio necesaria la necesidad de interconectar dichas puertas, ya sea con otra del mismo nivel o de escenarios distintos. Para ello, se lleva a cabo la gestión de puertas desde la misma creación de una, hasta la asignación de su enlace.

Para ello, la forma más rápida considerada a la hora de identificar las puertas ha sido la de obligar al usuario a determinar un nombre concreto para la puerta que se está instanciando, de modo que él pueda saber cual es la que quiere enlazar. Al colocar en el nivel por primera vez una de estas puertas, se muestra una ventana con un input text y se bloquea cualquier acción excepto la de introducir un nombre.

Una vez almacenado dicho nombre, el usuario puede continuar con la edición del nivel de forma normal. Si desea conectar esa puerta con alguna otra, deberá utilizar el botón de máquina de estados situado en el grupo de transformaciones. Al hacer click, se mostrará una ventana que solicita la selección de la carpeta donde se encuentra almacenado el nivel al que se desea realizar la transición. Una vez seleccionado, una ventana con dos menús desplegables aparece en pantalla, mostrando una lista con todas las puertas del nivel en el que se encuentra y otra con las puertas del nivel seleccionado previamente. Al elegir las dos, se habilita el botón de guardado, que enlaza dichas puertas.

Internamente, lo que se está haciendo es cargar las puertas directamente de una lista local de puertas instanciadas, y dada la ruta del siguiente nivel, si este es distinto al actual, se carga desde el fichero de puertas todos los nombres almacenados, así como su id. Al elegir las dos puertas a enlazar, la puerta del nivel actual actualiza sus campos de ruta siguiente e id de puerta siguiente, para que, al guardar, estos se almacenen en los ficheros.

## IA’s de Enemigos

Se han desarrollado tres inteligencias artificiales distintas que corresponden a tres enemigos diferentes, con el fin de aumentar la jugabilidad y diversas opciones de edición. Todas ellas, parten inicialmente de un sistema de detección de suelo, donde al iniciarse, si no encuentran suelo, caen por gravedad hasta chocar con un obstáculo, donde comienza su comportamiento. Estas tres IA’s se corresponden con:

## Side to Side

Un comportamiento simple y básico, donde el enemigo se desplaza de izquierda a derecha y hace daño al jugador por contacto. Inicialmente, se planteo un movimiento determinado por una distancia máxima de recorrido de izquierda a derecha, pero esto era difícil de calcular sin la colocación de waypoints en el escenario, ya que podía caer al vacío en cualquier momento.

Por ello, utilizando el sistema de comprobación de contacto con una superficie determinada, se alteró el comportamiento para que, en caso de llegar al extremo de la superficie, cambiase su dirección. De esta forma, se puede especificar una distancia máxima de movimiento, y a su vez, no será necesario preocuparse de que dicha distancia no sobrepase el extremo de la plataforma.

## Pathfinding

Para el desarrollo de este comportamiento, se plantearon varias posibilidades, teniendo en cuenta obstáculos y alturas. Finalmente, la solución fue la implementación de un Raycast que apunta al enemigo.

Se crean dos radios alrededor del enemigo, uno de detección del jugador y otro de ataque. Cuando el usuario entra en el radio de detección a su altura o a una inferior, el enemigo comienza a desplazarse en su dirección en el eje x. Si el jugador sale de dicho radio, el enemigo vuelve a su posición original.

Cuando el usuario entra en el radio de ataque del enemigo, este se detiene y comienza a atacar al jugador, restando una cantidad de vida establecida. El usuario puede atacarle o tratar de huir, en cuyo caso será perseguido por el enemigo atendiendo a las especificaciones anteriores.

Si el enemigo cayese de la plataforma en la que se encuentra, caerá en vertical hasta encontrar una superficie, momento en el cual actualizará su posición de partida, y continuará con su comportamiento normal. Si, por el contrario, no hay superficie alguna en su caída, será destruido tras alcanzar una altura concreta.

## Distance Attack

Utilizando parte de las ideas del comportamiento anterior, podríamos decir que este enemigo tiene una percepción infinita, ya que siempre va a detectar a su enemigo.

Utilizando la fórmula de la distancia entre dos puntos , calculamos la distancia entre el jugador y este enemigo. Al llegar a una distancia mínima, el enemigo instanciará una flecha en dirección al enemigo, y esta se desplazará a una velocidad concreta hasta ese punto. Si al llegar, el jugador ya no se encuentra en esa posición, continuará su desplazamiento en la misma dirección durante un tiempo. Si, por el contrario, encuentra un obstáculo en el camino, la flecha se detendrá y desaparecerá pasado un tiempo.

## Comportamiento Imán

Las plataformas tienen un comportamiento aplicado cuando se están desplazando (no al instanciarse), en el que, si uno de sus lados entra en contacto con el lado de otra plataforma, automáticamente se ajusta y se coloca pegado a esta segunda plataforma. Este comportamiento se puede desactivar al desplazar la plataforma y mantener pulsado shift izquierdo.

Para ello, las plataformas cuentan con dos colliders en los extremos, que se reajustan al escalar o rotar la superficie, y que, al entrar en contacto con otra, se desplazan atendiendo al tamaño del Sprite de dicha plataforma, y se bloquea su movimiento durante un tiempo determinado.

Inicialmente, se planteó utilizando trigonometría básica, entendiendo cada Sprite como una recta, y utilizando su rotación para componer un ángulo con el eje x, pero después de muchos intentos y análisis de los valores correspondientes, se determinó que unity debe utilizar un pequeño offset en estas rotaciones, ya que a pesar de que los cálculos matemáticos eran correctos, a la hora de aplicar la traslación, siempre quedaba un espacio entre ambas plataformas, creciente cuanto mayor era la rotación de la plataforma.

# Capítulo 6 Validación

|  |
| --- |
|  |

## Resultado final

## Experimentación

# Capítulo 7 Conclusiones

|  |
| --- |
|  |

## Logros alcanzados

Poner los objetivos que hemos puesto y poner si se han alcanzado y si me han gustado o no.

## Lecciones aprendidas

Qué cosas han sido un acierto y que cosas un error, una pagina como mucho (blockout útil?, si hubiera sido mejor usar las físicas de unity, crear un juego completo antes, etc)

## Lineas futuras

Ideas de cara al futuro, que mejoraría del editor, como liberalizarlos, etc etc

# Bibliografía

|  |  |
| --- | --- |
| [1] | Ludoscience, «Game Creation Tools Clasification,» [En línea]. Available: http://creatools.gameclassification.com/EN/index.html. [Último acceso: 27 Abril 2019]. |
| [2] | C. McEntee, 2014. [Online]. Available: https://www.youtube.com/watch?v=WFu1utKAZ18. |
| [3] | J. Schreier, Blood, Sweat and Pixels, HarperCollins, 2017. |
| [4] | M. Barton, Honoring the Code, CRC Press, 2016.  <https://hippogames.itch.io/fantasyheroes> |

# Ludografía

**La Abadía del Crimen**, Opera Soft, Opera Soft, 1987.

**Castlevania: Lord of Shadows**, MercurySteam & Kojima Productions, Konami, 2010.

**Commandos: Behind Enemy Lines**, Pyro Studios, Eidos Interactive, 1998.

**Gods Will be Watching**, Deconstructeam, Devolver Digital, 2014.

**Praetorians**, Pyro Studios, Eidos Interactive, 2003.

**They are Billions**, Numantian Games, Numantian Games, 2017.

**Shovel Knight**

# Anexo I Documento de diseño

# Anexo II Plan de negocios

HACER UN PATREON? COMO HACER QUE LLEGUE MAS LEJOS

Dado que el proyecto se basa en un Toolbox con mecánicas sencillas para ordenador, el planteamiento de un sistema de juego por tiempo, como puede ser el basado en un número de vidas concreto que disminuye a cada nivel jugado y se restaura pasado un tiempo, no parece muy apropiado.

Sin embargo, dado que el sistema de microtransacciones parece ser uno de los modelos más extendidos y aceptados actualmente, no es una mala opción como sistema de monetización.

Para ello, se planteará la posibilidad de ofrecer personajes nuevos o modificaciones de los ya existentes (ya sea con más vitalidad, mayor daño o simplemente un diseño más atractivo) a cambio de un número concreto de monedas de juego.

Estas monedas podrán obtenerse por jugar a los niveles oficiales del juego, a cambio de un módico precio, o por el sistema de edición de niveles que se plantea a continuación.

Supongamos un usuario al que denominaremos “*diseñador*” que ha creado diversos niveles extra para la comunidad del videojuego, y que dichos niveles tienen una alta popularidad entre los jugadores. El simple hecho de la satisfacción personal para el usuario por haber creado esos niveles y que hayan tenido tanto éxito no será suficiente pasado un tiempo, pues una recompensa para el *diseñador* es algo más que merecido.

Por otro lado, entendamos que los niveles desarrollados por los jugadores son compartidos por medio de una plataforma online para el resto de los usuarios, y que existe la posibilidad de valorar dichos escenarios, pongamos del 1 al 5, para de esta forma, entrar en un ranking de *mejor valorados*.

Con todo esto, se podría establecer una recompensa equiparable al rango de puntuaciones en el que se encuentra el nivel de *diseñador*, obteniendo monedas de juego a cambio, que podría gastar en las modificaciones anteriormente planteadas sin necesidad de invertir dinero real en la aplicación.

Posteriormente, y valorando el éxito de la aplicación en el mercado, podría llegarse a plantear la posibilidad de que por cada nivel creado que se encuentre entre las 3 primeras posiciones del ranking global durante un tiempo determinado (pongamos por caso 1 mes), se pueda recompensar económicamente a los diseñadores de dicho nivel, de forma que se incentive a los usuarios a desarrollar niveles para tratar de alcanzar dichas posiciones, y de este modo, se mantenga activa la comunidad, el videojuego, y el desarrollo por parte de terceros del videojuego.

Por otra parte, en lo referente a los elementos ofertados en la tienda del juego, y en vista de cómo se ha desarrollado el Toolbox, plantearemos otra situación para nuestro modelo de negocio.

Si *diseñador* ha realizado la compra del mago de fuego, el cual posee un báculo que realiza ataques mágicos a distancia que permite quemar a los enemigos y destruir ciertos elementos del escenario, no es de extrañar que quiera introducirlo en su nuevo nivel.

El problema surge cuando un usuario que no ha comprado dicho personaje quiere jugar a ese nivel.

Puesto que no tendría sentido la venta de elementos especiales que no es necesario comprar para jugar en niveles de terceros, se plantea la siguiente posibilidad:

Para poder jugar al nivel especial de *diseñador*, cuyos niveles son tan divertidos que toda la comunidad desea probarlos, pero no cumplen con los requisitos para ello, se permitirá que todos los usuarios jueguen una sola vez al nivel hasta su finalización (por completarlo o perecer), y posteriormente podrán pagar con monedas de juego para adquirir dicho escenario de forma indefinida, o simplemente comprar los elementos necesarios.

De esta forma, los usuarios podrán:

* Testar estas nuevas modificaciones para influir en su decisión de compra.
* Jugar a niveles para los que no tienen los requisitos.
* Obtener niveles de forma permanente sin necesidad de realizar la compra de modificaciones.

Con esto, tanto el *diseñador* se asegura de que puede obtener valoraciones de sus niveles independientemente de los elementos introducidos en ellos, como el jugador puede experimentar una experiencia de juego completa sin necesidad de invertir dinero real en la aplicación, aunque con algunas limitaciones especiales.

HACER UNA ESTIMACIÓN DEL PRECIO QUE HA COSTADO MI TRABAJO

# Anexo III Código

Añadir link a github

# Anexo IV Galería de imágenes

Mostrar imágenes del juego.