

23825156:ProDungeon-AlejandroPeñaHigon.docx

por Alejandro Peña Higon

Fecha de entrega: 07-jul-2020 01:30a.m. (UTC+0200)

Identificador de la entrega: 1354310761

Nombre del archivo: -6e93-453c-acf4-b7f7346af469_ProDungeon-AlejandroPeñaHigon.docx (3.16M)

Total de palabras: 13433

Total de caracteres: 71509



2

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

5

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Alejandro Peña Higón

Tutor: Alicia Villanueva García

2019-2020

1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.



2

Resumen

El TFG consiste en el desarrollo de un plugin para Unity encargado de la generación procedural de una o varias mazmorras basada en el comportamiento del jugador mientras recorre las mismas. El reto se plantea en el contexto del interés del alumno en el desarrollo de videojuegos. Tras explorar las distintas alternativas disponibles y descartarlas, se plantea el desarrollo de este motor de generación de escenarios profundizando así en la temática.

El plugin tiene como objetivo permitir personalizar la dificultad para cada jugador, creando así una experiencia adaptada que pueda proporcionar más satisfacción en el jugador y una sensación de mayor protagonismo durante el transcurso del juego. El proyecto incluye un editor de salas que serán almacenadas para la posterior generación y un sistema de eventos para recuperar estadísticas del jugador con las que modificar la propia sala.

Para el desarrollo se usará Unity y el lenguaje de programación C#.

Palabras clave: Unity; Generación procedural; Estadísticas; C#; Rider; Plugin.

Abstract

This TFG consists in the development of a plugin for Unity in charge of the procedural generation of one or more dungeons based ²⁷ on the player behavior while traversing the dungeons. The motivation for this work emerges in the context of the interest of the student in game development. After exploring the different alternatives available for dungeons generation and discard them, we set out the development of this dungeon generator engine which has allowed the student to deepen into the thematic.

The plugin's objective is to customize the game difficulty for each player, creating an adapted experience capable of providing more satisfaction in the user and a higher feeling of being in the limelight during the game. The project includes an editor to create customizable rooms that are stored for the dungeon generation and an event system capable of gathering the player's statistics to modify the procedural creation of the levels.

The technologies used to develop it have been Unity and the programming language C#.

Keywords Unity; Procedural Generation; Statistics; C#; Rider; Plugin.

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

1

Resum

El TFG consisteix en el desenvolupament d'un plugin complet per a Unity encarregat de la generació procedural d'una o varies mazmorres basada en el comportament del jugador mentre recorre les mateixes. El repte es planteja en el context de l'interès de l'alumne en el desenvolupament de videojocs. Després d'explorar les diferents alternatives disponibles i descartar-les, es planteja el desenvolupament de aquest motor de generació de escenaris profunditzant així en la temàtica.

El plugin té com objectiu permetre la personalització de la dificultat per a cada jugador, creant així una experiència adaptada que puga proporcionar més satisfacció en el jugador y una sensació de major protagonisme durant el transcurs del joc. El projecte inclou un editor de sales que seran emmagatzemades per a la posterior generació i un sistema d'esdeveniments per a recuperar estadístiques del jugador amb les quals modificar les sales.

Per al desenvolupament s'utilitzaran Unity y el llenguatge de programació C#.

Paraules clau: Unity; Generació procedural; Estadístiques; C#; Rider; Plugin.



Agradecimientos

A mi familia y novia por apoyarme en los momentos de mayor estrés, mi estancia Erasmus y la confianza depositada en mí.

A mi tutora Alicia Villanueva García por la paciencia y por perdonar mi caótica manera de realizar el trabajo.

A mis compañeros Víctor Rivera Vázquez y Daniel Torres Inglés por ayudarme en la iniciación del desarrollo de videojuegos creando Dreamcatchers y resolviendo mis dudas sobre Unity.

Agradecimiento especial a mis abuelos, uno de ellos fallecido recientemente, por haberme dado las directrices para trabajar sin descanso confiando en que los resultados llegarán.



1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.



Índice de contenido

10	1.	Introducción	10
	1.1	Motivación	11
	1.2	Objetivos	14
	1.3	Metodología	15
	1.4	Estructura	15
	1.5	Convenciones	16
	2.	Estado del Arte	17
	3.	Plan de trabajo	19
	3.1	Presupuesto	20
	4.	Diseño de la solución	21
	4.1	Arquitectura del sistema	21
	4.2	Diseño Detallado	22
	4.2.1	Prefabs	23
	4.2.2	Scenes	24
	4.2.3	Scriptable Objects	24
	4.2.4	Scripts	25
	4.3	Tecnologías Utilizadas	29
	4.3.1	Git	29
	4.3.2	Unity	31
	4.3.3	Rider	32
	4.3.4	Trello	36
	5.	Desarrollo de la aplicación	37
	6.	Conclusiones	44
2	6.1	Relación del trabajo desarrollado con los estudios cursados	45
	7.	Trabajo futuro	47
	8.	Referencias	48
	9.	Glosario	50



Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

Índice de Figuras

Figura 1 Dreamcatchers

7

Figura 2 Diagrama de clases de los sistemas del proyecto

Figura 3 Directorio proyecto

Figura 4 Carpeta Assets

Figura 5 InputText prefab

Figura 6 Carpeta ScriptableObjects

Figura 7 Carpeta Scripts

9

Figura 8 Diagrama de clases del Editor de Salas

7

Figura 9 Diagrama de clases del Generador Procedural

7

Figura 10 Diagrama de clases del Sistema de Eventos

Figura 11 Últimos commits del proyecto

Figura 12 Motores más utilizados 2018

Figura 13 Debug 'Attach to Unity Editor'

Figura 14 Ayudas Refactorización

Figura 15 Ayuda Visual de Variables

Figura 16 Read-Only Properties

Figura 17 Configuración Rider

Figura 18 Trello

Figura 19 Calendario del proyecto

Figura 20 Sprints Trello

Figura 21 Decisiones técnicas sprint 1

Figura 22 Tablero Kanban Sprint 1

Figura 23 Tablero Kanban Sprint 2

Figura 24 RoomSet

Figura 25 Tablero Kanban Sprint 3

Figura 26 Height map del planeta tierra

Figura 27 Tablero Kanban

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

1

1. Introducción

ProDungeon es un Plugin de Unity para la generación procedural de mazmorras que busca una inmersión total del jugador¹⁸ que utiliza sus propias estadísticas para personalizar la dificultad. El nombre viene de la unión de las tres primeras letras de la palabra “Procedural” y la palabra “Dungeon” la cual se traduce del inglés como mazmorra.

Para recuperar estas estadísticas se hará uso de los Eventos, los cuales almacenan información y son capaces de ser llamados desde cualquier parte del código. De esta manera cada vez que el jugador, mediante la interacción con el juego, realice alguna acción de la cual se desee guardar un registro, se llamará al evento para que actualice su valor. Usos prácticos de estos eventos podrían ser cuando el jugador acabe con un enemigo, avance a una nueva sala, pierda una vida o supere un nivel, entre otros.

El plugin está pensado para que los desarrolladores lo utilicen para sus propios juegos y lo personalicen creando sus propios tipos de eventos, pudiendo incluso crear su propio generador de mazmorras reciclando funcionalidades del provisto por ProDungeon, editando sus propias salas, creando nuevos tipos de casillas para formar estas salas, etc.

ProDungeon fusiona el concepto de Dynamic Difficulty Adjustment y la aleatoriedad para no solo personalizar la dificultad mientras se juega, sino también generar una sensación en el jugador de novedad, al ser muy difícil encontrarse con dos mazmorras idénticas.



1.1 Motivación

Hace dos años inicié un proyecto con otros 5 compañeros, 3 estudiantes del Grado de Ingeniería Informática y 2 estudiantes del grado de Diseño y Tecnologías Creativas, el cual consistía en el desarrollo de un juego rítmico para dispositivos móviles, creando así un equipo llamado LemonByte. Fue descartado debido a que subestimamos la carga artística del mismo y se produjo lo que comúnmente se conoce como un cuello de botella en la realización de las tareas, es decir, que las artistas se vieron saturados de trabajo mientras que los programadores nos empezábamos a quedar sin nuevas características que implementar.

A pesar de la cancelación, nuestro apetito por aprender a realizar estos proyectos aumentó con creces e ideamos una nueva forma de seguir por este camino sin depender de otras personas. Los programadores dirigimos nuestra atención hacia un nuevo proyecto el cual no requería de un apartado artístico, mientras que las artistas se enfocaron más en sus carreras ya que todavía todos éramos estudiantes.

El nuevo proyecto, llamado Dreamcatchers, asentó los precedentes de este trabajo final de grado, ya que este último surgió de la necesidad de mejorar la generación de las mazmorras del juego al ser una tarea muy lenta y costosa. Aunque Dreamcatchers inicialmente se planteó como TFG de emprendimiento, únicamente ¹³ un compañero lo usó para su TFG extrayendo una parte como pieza independiente: <<Dreamcatchers: Videojuego HORDE & ROGUELIKE desarrollado en Unity para dispositivos móviles>> Ribera Vázquez, Víctor (2018) [1]. Podemos observar una captura del juego en la figura 1. El videojuego consistía en una fusión entre dos estilos de juego distintos los cuales no se habían combinado todavía de manera profesional, uno era el estilo “Horde” [2] y el otro estilo era el Rogue-lite [3]:

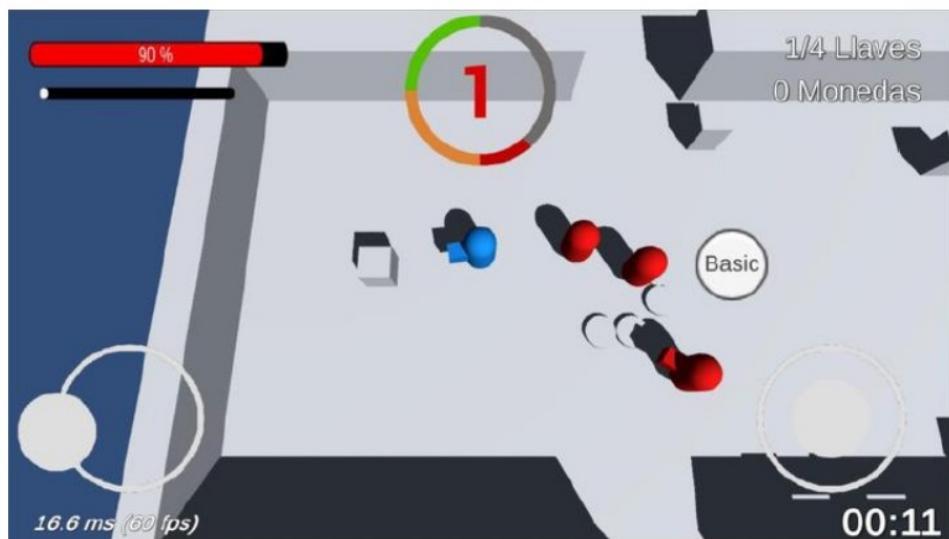


Figura 1. Dreamcatchers

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

El estilo de juego Horde consiste en una sucesión de “Waves” u oleadas en las que aparecen una determinada cantidad de enemigos, dependiendo del número de iteraciones ya superadas por el jugador, que deben ser eliminados en su totalidad para avanzar a la siguiente “wave” u oleada. El objetivo del juego es aguantar la mayor cantidad de oleadas posibles, soportando el aumento de dificultad progresivo ya sea mediante el aumento de enemigos, nuevos y más complejos tipos de enemigos y/o alguna limitación implementada por los desarrolladores. Las herramientas para superar estas oleadas varían desde mejoras para las armas o mecanismos para acabar con las entidades enemigas a la adquisición de Power-Ups o el aumento del espacio transitable para una mejor maniobrabilidad. Ejemplos de este estilo muy conocidos en la industria son: El modo Zombies de Call Of Duty [4] (el cual se ha repetido en distintas entregas de la saga siendo un éxito en todas ellas), Plants Vs. Zombies: Garden Warfare 2 [5], Modo Firefight de Halo Reach [6] y Warhammer: End Times – Vermintide [7].

Por otro lado, el estilo de juego Rogue-lite se caracteriza por ser un derivado del estilo Rogue-like utilizando así ciertas características de éste como la generación procedural o la denominada “Permadeath” (muerte irreversible), pero se diferencia de este estilo por presentar unos gráficos más complejos (los gráficos de los Rogue-likes solían ser en ASCII), por una menor importancia de la estrategia para superar los niveles, y porque el movimiento del jugador no tiene por qué ser mediante comandos de texto y/o basado en turnos. El estilo de juego busca que el jugador (teniendo en cuenta las características anteriormente descritas) realice una “Run” o iteración atravesando las mazmorras hasta que o bien alcance el final de esta o lo maten; pase una u otra cosa el jugador volverá al inicio del juego sin conservar nada, pero si el jugador consigue cumplir ciertos objetivos conseguirá desbloquear nuevos objetos que podrán aparecer de manera aleatoria en las siguientes iteraciones. De este modo se insta al jugador a que realice “Runs” de manera ilimitada hasta desbloquear todo el contenido del juego y/o batir récords de velocidad, los cuales son aclamados entre la comunidad Online. Ejemplos de este estilo son The Binding Of Isaac [8] (probablemente el juego de mayor éxito de este estilo de juego), Nuclear Throne [9] y Dead Cells [10].

Suena contradictorio que escogiésemos el estilo de juego Rogue-lite en lugar del estilo Rogue-like para la realización de este juego ya que previamente habíamos tenido problemas con el apartado artístico, pero los gráficos de Dreamcatchers ya son más complejos que los de un Rogue-like convencional debido a que son figuras tridimensionales (por lo cual, más complejos que el ASCII). Además, nos convenció más este género ya que la plataforma en la que planeábamos lanzar el juego era la de la telefonía móvil, en la que, tras una breve investigación, nos dimos cuenta de que el éxito de los juegos se basaba en la rejugabilidad y el desestrés (juegos que no buscan la total concentración del jugador sino más bien el placer de distraer la atención unos minutos).

Tras la finalización del proyecto formalmente, me decidí a ahondar en la industria del videojuego y dirigí mi evolución académica a los Países Bajos, en concreto a Ámsterdam, donde realicé mi estancia Erasmus estudiando un semestre temático sobre “Game Technology” (Tecnología de Videojuegos) [11]. Sin embargo, en lugar de aprender más sobre Unity [12] el motor gráfico sobre el que realmente me interesaba y que he utilizado para desarrollar este trabajo, nos enseñaron a utilizar otro llamado Unreal Engine 4 [13], ya que mis compañeros neerlandeses ya habían sido instruidos en esta tecnología en anteriores semestres. Durante el transcurso de esta experiencia internacional, me topé con una asignatura en el plan de estudios llamada “Automated Design” (Diseño Automatizado) en la cual nos instruyeron en las diferentes técnicas para esquematizar, diseñar e implementar este tipo de sistemas.



Aquí es dónde inicia el proceso que desemboca en lo que hoy día conocemos como ProDungeon, ya que surgió la idea de arreglar el problema de la generación de mapas de Dreamcatchers utilizando los conocimientos recién adquiridos. El primer “prototipo” (Si es que así pudiéramos llamarlo), se trataba de un sistema de eventos que era capaz de recolectar estadísticas del jugador y modificar mínimamente la generación de terreno ya existente en Dreamcatchers. La generación se mejoró haciéndola más “barata” (dentro de la industria del videojuego “barata” hace referencia a una característica, algoritmo, implementación, etc. la cual consume pocos recursos computacionales) pero seguía siendo ineficiente.

Con vistas a utilizar esta idea a modo de trabajo de final de grado se realizó una investigación para saber si existían potenciales competidores. Debido a esta investigación me di cuenta de que la idea de hacer un plugin de la generación procedural basada en el comportamiento del jugador de manera genérica no había sido realizada anteriormente por ninguna compañía y no había nadie que sugiriera que lo estaba intentando de manera personal en internet. Esto significaba que no tenía nadie en quien apoyarme, pero tenía el incentivo de ser pionero en este proyecto. La idea de exportarlo como un Plugin para Unity y no únicamente dejarlo aplicado sobre Dreamcatchers surgió de una conversación con el docente Ramón Pascual Mollá Vayá.

Si bien es cierto que algunos juegos regulan la dificultad durante el transcurso del juego, no encontré nadie que realizase plenamente la idea de este TFG, a pesar de encontrar comentarios en foros de usuarios interesados en la materia (a modo de matización, no buscaban exactamente la idea que refleja este trabajo, pero sí algunas características que aúna el mismo). Con lo cual encontré un conjunto de ideas requeridas y las fusioné pensando que sería el primero en realizarlo. Hace unos meses, el día 16 de Mayo, descubrí que no era el único que había pensado en ello como se refleja en el siguiente tuit:

<https://twitter.com/mjcguada/status/1261649764786909184?s=20> .

Tanto ProDungeon como este segundo trabajo emanan de la misma idea y se solapan en el tiempo de desarrollo, remarcando la idea de que existe un público de desarrolladores online que requieren de esta funcionalidad para implementar en sus propios proyectos.



Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

1

1.2 Objetivos

Los objetivos planteados inicialmente del trabajo pueden resumirse como sigue:

-Proveer de una herramienta útil a los desarrolladores: Se busca ofrecer una manera rápida y eficiente de generar las mazmorras tanto a nivel principiante como a nivel más profesional.

-Incitar a los desarrolladores a explotar el plugin: El programa tiene que ser modular permitiendo a los usuarios experimentar con él y permitirles personalizarlo para su propio proyecto. Por ejemplo, debe existir la posibilidad de registrar los propios eventos que posteriormente influirán en la generación. Con todo ello, se puede recibir un feedback muy valioso para la mejora del producto e incluso la adición de nuevas funcionalidades requeridas.

-Mejorar mis conocimientos del motor Unity: poniendo la vista en el futuro laboral, esta experiencia será crucial para el aprendizaje de Unity, para una sustancial mejora del porfolio y de mis nociones sobre Unity.

-Mejorar la generación de niveles de Dreamcatchers: ya que fue la motivación inicial del proyecto, tiene sentido aplicar esta nueva herramienta sobre el proyecto anteriormente mencionado. Con dicha integración se pretende que el jugador no pierda el interés en el juego y mejorar la experiencia del usuario.

-Lucro económico: Existe la posibilidad de beneficio económico al distribuirse este trabajo a través de la tienda de Unity. La idea principal es distribuir una primera versión gratuita con la que recibir opiniones y críticas de los usuarios (los cuales no tendrían que pagar nunca la aplicación, al ya haberla adquirido) y tras varias mejoras en el plugin establecer la cuota de adquisición de éste a 5€ (precio asequible tanto para principiantes como para empresas pequeñas). Al ser un precio tan reducido, los usuarios podrían incluso comprarlo por simple curiosidad para probarlo, al no suponer una gran pérdida económica.

-Disponer de una herramienta para futuros proyectos: planeo emplear este producto para futuros videojuegos creados por mí mismo.

El impacto esperado de este trabajo es en primer lugar que sirva como trabajo de aprendizaje de Unity y C#, tecnologías clave para el desarrollo de videojuegos, el cual será el objetivo de mi futuro laboral. También servirá como herramienta para futuros trabajos realizados en la plataforma, así como una posible vía de lucro para financiar estos trabajos. Por último, permitirá mejorar un proyecto previamente realizado, llamado Dreamcatchers.



1.3 Metodología

El primer paso para realizar este proyecto fue, como bien nos enseñaron en la rama de Ingeniería del Software y la asignatura de Gestión de Proyectos, realizar un **estudio del mercado** para analizar nuestras virtudes y debilidades frente a los productos potencialmente competitivos en caso de que los haya. Este ejercicio también permite identificar funcionalidades que mejoren el proyecto o la necesidad de refinar alguna de las ya existentes y así inclinar la balanza a favor nuestro. Gracias al estudio de mercado partiremos de una mejor posición, utilizando la experiencia de nuestros competidores a nuestro favor. Este paso será mostrado en esta memoria en el apartado: 2 Estado del Arte.

A continuación, con la idea ya contrastada con el contexto donde se entabla este trabajo, pasaremos a **idear un plan de trabajo** utilizando, de nuevo, conocimientos previamente aprendidos entre el tercer y el cuarto año de la carrera. En concreto emplearemos metodologías ágiles, así como el uso de un tablero Kanban, la estimación de esfuerzo de las tareas, el uso de sprints o la creación de un backlog de tareas ordenadas por prioridad.

El **desarrollo de la solución** se dividirá en 3 etapas distintas: la generación de mazmorras procedural, el editor de salas y el sistema de eventos para recuperar la información del jugador. A éstas le sucederá una etapa extra la cual consistirá en integrar los tres distintos sistemas que componen este proyecto.

Por último, en caso de que el tiempo lo permita, **se subirá la aplicación a la tienda de Unity** donde esperaremos recibir el feedback de los usuarios, para así terminar de pulir posibles funcionalidades que, a pesar de parecer lógicas y cómodas desde nuestro punto de vista, puede que no sea así para los desarrolladores que pretendan integrarlo en sus proyectos.

1.4 Estructura

La memoria se divide en 7 capítulos que a continuación introduciremos brevemente:

En la introducción se expone el origen de la idea de este trabajo y sus objetivos tanto a nivel profesional como personal, los pasos que se siguieron durante la realización del trabajo, la estructura que tomará esta memoria y las convenciones útiles para el lector para facilitar la comprensión de la misma.

En el segundo capítulo encontrarán la situación actual del contexto dónde se sitúa el proyecto mediante citas y referencias a artículos científicos encontrados durante la primera fase del trabajo, y sus diferencias con este trabajo. No encontraremos un apartado de crítica ya que, como bien se explica en el mismo fragmento de la memoria, el plugin desarrollado no encuentra una competencia directa (según la investigación realizada) y se incluyen las distintas críticas directas a los artículos en el apartado general.

En el tercer capítulo encontramos el plan de trabajo previo al desarrollo del producto, el cual incluye una estimación inicial de esfuerzo sobre cada una de las partes del proyecto, para así posteriormente reflexionar sobre estas comparándolas con los resultados reales obtenidos.



1
Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

La cuarta parte de la memoria será el inicio de la parte más técnica del trabajo, describiendo la arquitectura y el diseño detallado del plugin a través del uso de diagramas UML aprendidos tanto en la asignatura de Ingeniería del Software como en la rama a la que ésta da nombre. También se describirán las tecnologías empleadas durante el desarrollo.

En quinto lugar, cerrando la parte técnica, se procederá a la descripción del desarrollo de la aplicación, explicando la transición de la propuesta al resultado generado, las dificultades encontradas en el trámite y las decisiones técnicas que se tomaron con sus correspondientes justificaciones.

22
Como penúltima sección se encuentran las conclusiones del proyecto donde se reflexionará sobre el cumplimiento de los objetivos y planes iniciales, el valor de la experiencia desde el prisma didáctico, el grado de similitud entre el esfuerzo estimado y el real; y se relacionarán los conceptos y valores empleados con el plan de estudios del grado.

Por último, podremos leer cuáles son las proyecciones de futuro del producto y otros trabajos relacionados con el mismo.

1.5 Convenciones

Para una mejor comprensión del documento, en este apartado se explicarán las convenciones a tomar en cuenta durante la lectura de este:

-Existe un Glosario al final del documento con la definición de diversos términos que podrían ser ajenos al lector si éste no se encuentra actualizado con la industria del desarrollo de videojuegos.

-Encontraremos también junto al Glosario un apartado llamado Referencias el cual contiene todas las que se encuentran a lo largo de la memoria. Podemos detectarlas ya que aparecen indicadas con corchetes “[]” y un número indicando el índice de estas en la lista.

-Las referencias a artículos científicos se realizarán con el estilo Vancouver [14] ya que al parecer es el más extendido e incluso aparece en la gran mayoría de artículos estudiados durante la realización de este trabajo.

-Las citas textuales y los nombres de artículos que se encuentren a lo largo del documento se remarcarán en cursiva y se envolverán con los signos de desigualdad “<<” y “>>”.

-En determinados apartados del documento basados en fases o apartados se remarcará en negrita estos para clarificar y resumir de manera más visual estos.



2. Estado del Arte

Como ya se ha relatado en el apartado 1.1 Motivación, durante la investigación previa al proyecto no encontré ningún artículo que desarrollase la idea directa que ocupa este trabajo. Sin embargo, me gustaría mencionar ciertos artículos que fueron utilizados a modo de inspiración y/o guía en las distintas partes que aúna este proyecto. La mayoría de ellos se encontraron utilizando Google Scholar.

En primer lugar, siendo uno de los artículos que más me impulsó a escoger este tema como mi TFG, ya que me hizo ver que ⁸ era la única persona que necesitaba una herramienta como la que hemos creado, se encuentra <<*Procedural generation of angry birds levels that adapt to the player's skills using genetic algorithm*>>[15] escrito por Misaki Kaidan, Chun Yin Chu, Tomohiro Harada y Ruck Thawonmas. En él podemos ver el trabajo realizado para, usando un algoritmo genético, regular la experiencia del famoso juego llamado Angry Birds. No solo este trabajo está aplicado sobre un juego en concreto (siendo inutilizable para otros), sino que además el software desarrollado clasifica al jugador en 4 estilos distintos de juego y regula la dificultad en base a ello, para que el jugador no se aburra, se frustre o pierda el interés en el juego (una de las motivaciones principales de este proyecto). Como podemos ver, este proyecto es demasiado específico ya que no se puede exportar a otros juegos y únicamente ofrece 4 posibilidades de dificultad procedural, mientras que ProDungeon se puede utilizar en cualquier proyecto de Unity basado en mazmorras y genera una cantidad infinita de posibilidades de dificultad, ya que cada acción del jugador que se deseé contabilizar por parte de los desarrolladores influirá en la generación del terreno.

En el artículo <<*Mobile adaptive procedural content generation*>> [16] se expone y desarrolla la idea que previamente hemos enunciado sobre que los juegos de la plataforma móvil son distintos a los juegos tradicionales de consolas y/o computadoras, ya que el modo de uso de éstas no es el mismo. Las aplicaciones telefónicas buscan aprovechar ventanas de tiempo pequeñas al funcionar sobre dispositivos que las personas llevan encima durante su día a día, pudiendo ser utilizadas en (prácticamente) cualquier lugar y momento. Por ello, los Adaptive games tienen mucho éxito en esta plataforma, son más personales al adecuarse a las necesidades individuales del usuario y ser muy rejugables. Dreamcatchers nació como un proyecto destinado a la plataforma móvil por este mismo motivo. En el artículo se aborda la idea de los Adaptive games y cómo la generación procedural ayuda a esta adaptación del software a los jugadores, y que existen maneras de que los desarrolladores controlen dicha generación, ejemplificando con la aplicación de ésta a un juego llamado 7's Wild Ride:

³
<<*Control over the content generator can be exerted by selecting which level segments, obstacles and obstacle locations to combine. In this section, we describe the main methods used to support this on 7's Wild Ride adaptive case study.*>>

Otro artículo relacionado con este Trabajo de Final de Grado es <<*Controlled Procedural Terrain Generation Using Software Agents*>> [17] el cual trata ampliamente la generación procedural en juegos utilizando agentes software. En concreto exploran la manera de generar *height maps* basado en las restricciones del desarrollador. A diferencia de nuestro proyecto, aquí es el desarrollador antes de iniciar el programa el que decide qué puede y qué no puede suceder, obteniendo un resultado “esperado” (el resultado es aleatorio, pero cumple las normas impuestas).

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

14

Podemos observar que en la siguiente tesis <<*Controlling randomness: Using Procedural Generation to influence player uncertainty in video games*>> [18] se trata justo el caso opuesto, es decir, cómo influenciar o modificar el comportamiento del jugador a través de la generación procedural. Este documento fue muy interesante y me sugirió varias ideas de cómo encaminar el trabajo resaltando varios aspectos importantes relacionados con la generación procedural como el potencial que tiene para modificar u opacar otras partes importantes del producto como la narrativa o las mecánicas principales.

Respecto al TFG mencionado en el penúltimo párrafo del apartado 1.1 Motivación, debemos matizar que aunque ambos trabajos hereden del mismo concepto, la gran diferencia entre ambos es que ProDungeon es un plugin genérico que ayuda a aplicar esta idea de ajuste dinámico de la dificultad basado en el comportamiento del jugador sobre juegos desarrollados en Unity, mientras que el otro trabajo se trata de un juego 2D con una inteligencia artificial que genera los mapas.

Por último, me gustaría mencionar al proyecto que probablemente sea el más parecido a ProDungeon, llamado <<*Polymorph: Dynamic Difficulty Adjustment Through Level Generation*>> [19]. Aunque no se define muy claramente en el artículo, podemos entender del siguiente extracto, que se trata de un programa para construir un juego (no es genérico) de plataformas 2D con un nivel de dificultad apropiado para el jugador:

6

<<we present Polymorph, which employs techniques from level generation and machine learning to understand game component difficulty and player skill, dynamically constructing a 2D platformer game with continually appropriate challenge>>

Este proyecto se basa en ir añadiendo nuevas partes al nivel horizontalmente mientras el jugador va recorriendo el mismo. Nuestro software, a diferencia, genera mazmorras (destinadas a tipos distintos de juegos) que pueden ser renderizadas tanto en 3D con en 2D según el desarrollador requiera y las salas se añaden tanto horizontalmente como verticalmente, siendo éste un valor alterable por el comportamiento del jugador. También necesitan comparar cómo juega el jugador con un estándar de dificultad que recae plenamente en hacer jugar a varios jugadores los distintos segmentos y recibir un feedback manual, lo cual es muy costoso y manual. En ProDungeon la DDA (Dynamic Difficulty Adjustment) se hace de manera totalmente automática ya que son los propios eventos los que, al recibir las llamadas mientras el jugador intenta superar los niveles, autorregulan la dificultad del nivel. La dificultad en ProDungeon es totalmente personalizada ya que se modifica dependiendo de tantos factores como el desarrollador deseé analizar de su juego.

Con todo esto podemos concluir que no hemos encontrado ningún proyecto que se ajuste a las características principales de ProDungeon, pero sí que existen muchos estudios relacionados con la generación procedural y por lo tanto un gran interés por la industria del videojuego en esta materia.



3. Plan de trabajo

En esta sección describiremos el plan para abordar el desarrollo del proyecto y estimaremos la cantidad de esfuerzo que supone cada paso del mismo. Se debe tener en cuenta al leer este apartado que el único recurso humano disponible para la realización del trabajo es el propio alumno. La manera de indicar el posible esfuerzo que requerirán cada una de las partes de este plan se hará dando una cantidad de puntos determinada a cada una de ellas. La equivalencia de los puntos es un tema muy controvertido dentro del mundo de las metodologías ágiles, como bien nos enseñaron en la asignatura de Proyecto Software en el tercer año de la carrera y como experimenté durante mi estancia Erasmus en los Países Bajos, ya que no se debe reducir a una relación simple de puntos-horas debido a que existen factores muy complejos en esta relación que no pueden ser contabilizados, como posibles imprevistos, estado del ánimo de los trabajadores, complejidades técnicas descubiertas a posteriori, etc. Con esto en mente procedemos a describir las distintas partes y la estimación de cada una:

La primera tarea es el **diseño de la arquitectura** ideal del **producto** para su posterior subdivisión y así crear un hilo conductor que seguir mientras se ejecutan los siguientes pasos. Esta tarea es crucial al principio del proyecto ya que ayuda a aligerar el resto, dando un punto de partida y un esqueleto sobre el que ir construyendo. El esfuerzo requerido para realizar este paso se verá influenciado por los conocimientos de diseño del desarrollador, por el factor de inicio del proyecto (es decir, la posibilidad de estancamiento debido a ser el primer paso) y por la escasez de trabajos a los que referirse en caso de duda. La puntuación inicial de esta tarea es de 1 puntos.

La siguiente tarea es la **creación del backlog** basado en la arquitectura previa, la cual podría ser modificada en el proceso (ya que como hemos dicho se trata de una estimación). Se identifican las distintas partes del producto creando así posibles áreas de trabajo y se desglosan las mismas para crear las etiquetas o tareas que se situarán en la columna inicial del tablero Kanban. Junto con esto se determina la duración deseada de los sprints, así como se prepararán los 2 o 3 primeros (según la duración temporal de estos). El esfuerzo necesario estimado de este apartado será de 3 puntos.

Una vez definido en Kanban puede empezar el **desarrollo** basado en sprints, preparando los siguientes sprints cada vez que se acabe uno de ellos. De esta forma podemos utilizar la experiencia adquirida para refinar la cantidad de trabajo en cada uno de ellos. Este apartado será el más costoso ya que absorberá la mayor cantidad de recursos, tanto de esfuerzo como temporales, ocupando así un periodo temporal bastante extenso. El gran factor que afectará a este apartado será la curva de aprendizaje de las tecnologías y del nuevo lenguaje de programación, así como la resolución de errores durante el desarrollo. El esfuerzo calculado a priori de este paso es de 24 puntos.

Una vez desarrolladas las distintas partes del producto, se realizará la **integración** de las mismas para que funcionen como un conjunto. Al desarrollar las partes como unidades distintas (teniendo en mente que deben unirse posteriormente) es más fácil centrarse en pulirlas individualmente y genera una mayor satisfacción al completarlas, favoreciendo el estado mental del desarrollador. Esta decisión se tomó siguiendo la experiencia del estudiante en proyectos previos. El inconveniente de esta metodología es que se genera un trabajo posterior de conexión de las partes. El factor de riesgo en este apartado es la habilidad del desarrollador de programar

1
Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

las distintas piezas del producto de tal manera que queden preparadas para su ensamblaje. El esfuerzo necesario para realizar este paso es de 7 puntos.

A continuación se procederá al **testing**, donde se verificará que todo funciona según lo deseado. Debido a que los recursos del estudiante son limitados, no existirá una parte de quality assurance en la que otro individuo revise el correcto funcionamiento de la aplicación. Se notificarán y repararán los bugs no detectados mediante el testing a través del feedback de los usuarios en su/s primera/s versión/es gratuita/s. A pesar del deseo del alumno por dedicar una gran cantidad de esfuerzo en este apartado para evitar la mayoría de los comportamientos no deseados, debido a la envergadura del proyecto se realizarán la mayor cantidad de pruebas posibles hasta la fecha de entrega de este trabajo. Por ello, la puntuación inicial de este paso será de 10 pudiendo ser superada voluntariamente a posteriori.

Por último, **se subirá el producto final a la tienda de la plataforma Unity**. Este apartado únicamente tendrá 1 punto de esfuerzo ya que no debería suponer ningún problema con todo el trabajo realizado previamente.

3.1 Presupuesto

En cuanto a los recursos humanos, este proyecto cuenta únicamente con el alumno como desarrollador. Las ²¹ horas iniciales estimadas para el desarrollo de este son 400, pudiendo ser distribuidas entre los meses de febrero, marzo, abril y mayo, dejando el mes de junio para la redacción de las partes finales de la memoria y limpieza del conjunto. El software empleado para la realización de éste quedará citado en el apartado Tecnologías Utilizadas siendo gratuito en su totalidad.



4. Diseño de la solución

Dividiremos este apartado en dos partes, una para la explicación simple de los distintos sistemas que componen el plugin, así como la relación entre los mismos y una segunda parte donde profundizaremos dentro de cada uno de los sistemas para explicar el diseño escogido para éstos.

4.1 Arquitectura del sistema

En el diagrama UML de la siguiente figura podremos observar los distintos sistemas que intervienen en el producto:

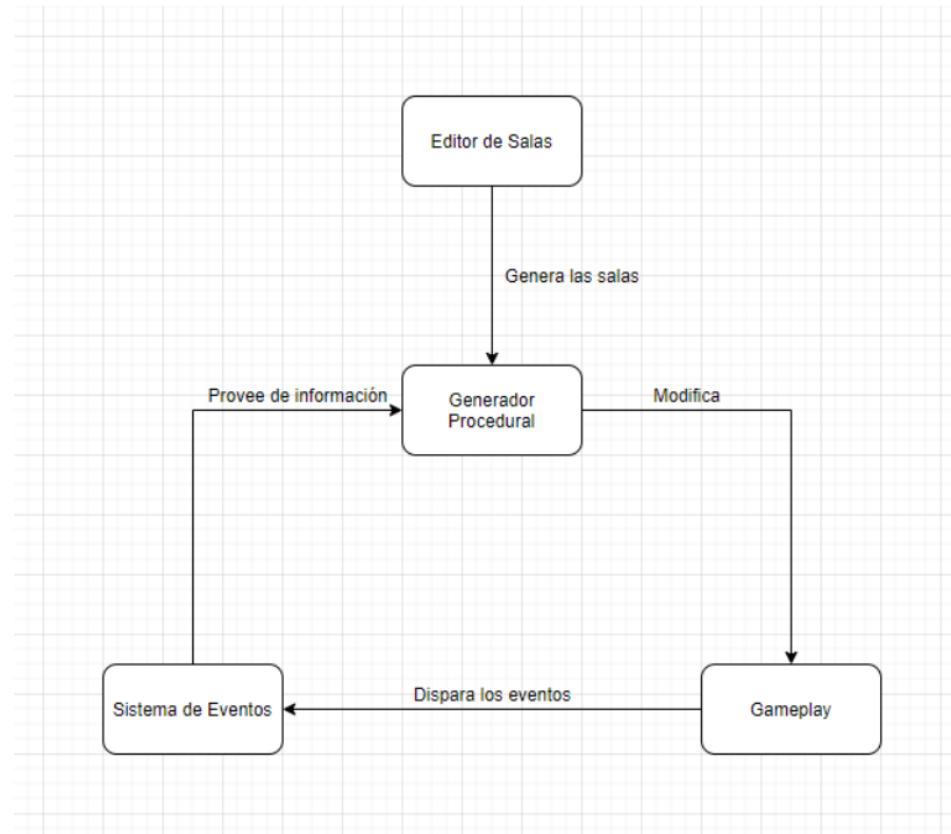


Figura 2. Diagrama de clases de los sistemas del proyecto.

1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

-El primero es el **Editor de Salas** con el cual, como su nombre indica, el desarrollador es capaz de crear nuevas salas que posteriormente serán empleadas por el generador procedural para la creación de nuevos niveles o mazmorras. Cuenta con una interfaz, un sistema de pinceles para hacer más cómoda la creación de las salas y un dropdown con el que podemos seleccionar alguna dala anteriormente guardada para su posible rediseño (entre otras funcionalidades).

-El **Generador Procedural** es el encargado de recuperar la información de los eventos a partir de las instancias de una clase que procesa todos los eventos y devuelve un valor para la generación (Modifiers); y de generar los nuevos niveles utilizando estos valores y las salas del RoomSet (contenedor de salas que permite agruparlas por tipos y/o por características comunes), modificando así la experiencia de juego del jugador. En el siguiente epígrafe se detallará el significado de las clases que se acaban de nombrar.

-El **Sistema de Eventos** almacenará toda la información que el desarrollador haya determinado como relevante para la generación de las mazmorras o niveles. Desde cualquier parte del código del juego del usuario se podrá suscribir a los eventos del sistema (creados a través del editor de Unity) y también se podrá actualizar el valor de cada evento.

-El **Gameplay** representado en este diagrama es el conjunto de sistemas y comportamiento del jugador no controlables y ajenos a lo desarrollado en este trabajo. Podemos decir que se trata de un agente externo que será afectado por las modificaciones del generador y proveerá de un continuo flujo de inputs.

4.2 Diseño Detallado

Como podemos ver en la figura 3 el directorio del proyecto está estructurado en un conjunto de carpetas:

📁 .git	18/06/2020 23:27	Carpeta de archivos
📁 .idea	05/03/2020 22:02	Carpeta de archivos
📁 Assets	24/04/2020 19:09	Carpeta de archivos
📁 Library	10/06/2020 13:20	Carpeta de archivos
📁 Logs	05/03/2020 22:03	Carpeta de archivos
📁 obj	05/03/2020 22:08	Carpeta de archivos
📁 Packages	05/03/2020 22:03	Carpeta de archivos
📁 ProjectSettings	01/06/2020 20:19	Carpeta de archivos
📁 Temp	14/06/2020 4:26	Carpeta de archivos
📄 .gitignore	15/05/2020 19:14	Documento de tex...
📄 Assembly-CSharp.csproj	18/06/2020 23:27	Visual C# Project f...
📄 Assembly-CSharp-Editor.csproj	01/06/2020 20:26	Visual C# Project f...
🔗 TFGproject.sln	24/05/2020 1:48	Microsoft Visual S...

Figura 3. Directorio proyecto

La parte que interesa para esta memoria se encuentra en la carpeta Assets:

📁 Prefabs	21/04/2020 22:38	Carpeta de archivos
📁 Scenes	10/06/2020 13:20	Carpeta de archivos
📁 ScriptableObjects	28/05/2020 20:08	Carpeta de archivos
📁 Scripts	18/06/2020 23:27	Carpeta de archivos
📁 TextMesh Pro	02/04/2020 20:28	Carpeta de archivos
📄 Prefabs.meta	21/04/2020 22:24	Archivo META 1 KB
📄 Scenes.meta	05/03/2020 22:04	Archivo META 1 KB
📄 ScriptableObjects.meta	24/04/2020 19:09	Archivo META 1 KB
📄 Scripts.meta	05/03/2020 22:05	Archivo META 1 KB
📄 TextMesh Pro.meta	11/11/2017 20:24	Archivo META 1 KB

Figura 4. Carpeta Assets

4.2.1 Prefabs

Dentro de la carpeta llamada **Prefabs** encontraremos el conjunto de **12** objetos prefabricados del proyecto. Estos objetos, llamados como la carpeta (prefabs), son un tipo de Asset que permite almacenar un GameObject con sus componentes y propiedades, pudiéndose usar como plantillas para instanciarlas y tener varios objetos de ese tipo de prefab. De esta manera cada vez que se cambie alguna propiedad en el prefab, el resto de las instancias de éste sufrirán el mismo cambio, consiguiendo así mantener un estilo común entre todas ellas.

Dentro de ProDungeon existe el prefab InputText, el cual se compone de una etiqueta y de un campo rellenable. Este prefab es utilizado varias veces en la interfaz del Editor de Salas para recuperar información para la creación de las Rooms. Con la siguiente figura podemos ver visualmente tres ejemplos de InputText:

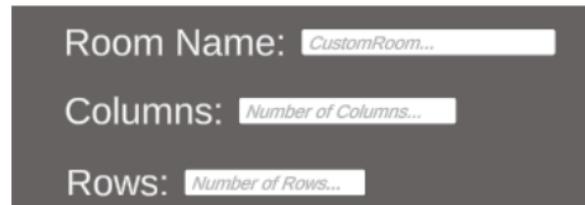


Figura 5. InputText prefab

1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

4.2.2 Scenes

En la carpeta **Scenes** se hallan las distintas escenas del proyecto. Una escena contiene los distintos objetos que se quieren mostrar durante el juego, siendo posible cambiar entre ellas tanto desde el editor como mientras se ejecuta el programa. De esta manera podemos crear una escena para un menú, otra para el inventario del jugador y otra dónde se encuentra el terreno de juego e ir alternando entre ellas. Las escenas de este proyecto son dos, una se corresponde con el editor de salas y la otra con un ejemplo de cómo utilizar el plugin para renderizar la mazmorra. La escena de ejemplo será utilizada para mostrar visualmente cómo funciona la generación.

4.2.3 Scriptable Objects

El directorio llamado **Scriptable Objects** almacena todos los assets de este tipo que se usan en el proyecto. Para ordenarlos existen carpetas dentro de este con el nombre de los distintos tipos de Scriptable Objects creados en el trabajo. Un Scriptable Object es un contenedor de información que permite almacenar grandes cantidades de datos independientemente de instancias de clases o scripts. Uno de los usos típicos de estos objetos es reducir la utilización de memoria del proyecto evitando crear copias de los valores. Además, podemos crear nuestros propios tipos de Scriptable Objects para que estos representen objetos de nuestro programa y dotarlos de la funcionalidad que queramos. Por ejemplo, en este plugin se han creado cuatro clases de Scriptable Objects distintos: Los DungeonTileType, que representan los distintos tipos que pueden representar las casillas de las mazmorras (suelo, puerta, hueco, pared, etc.); los Events, eventos que contienen las estadísticas del jugador; Los Modificators, que establecen las diferentes formas en las qué se puede modificar la generación de la mazmorra y recuperan las estadísticas del jugador; y por último, las Scriptable Rooms, que contienen la información que se desea guardar de las salas. Con estos Scriptable Objects podemos serializar y encapsular la información para que en caso de que el jugador quiera cerrar o pausar la sesión de juego, el progreso y la información de juego se guarden (por ejemplo, la mazmorra creada, las salas generadas, las estadísticas, etc.).

En la siguiente figura observamos la distribución del directorio:

📁 DungeonTileType	10/06/2020 12:44	Carpeta de archivos
📁 Events	10/06/2020 13:03	Carpeta de archivos
📁 Modificators	10/06/2020 13:20	Carpeta de archivos
📁 Rooms	10/06/2020 12:56	Carpeta de archivos
📄 DungeonTileType.meta	30/04/2020 19:36	Archivo META 1 KB
📄 Events.meta	22/05/2020 0:05	Archivo META 1 KB
📄 Modificators.meta	28/05/2020 20:08	Archivo META 1 KB
📄 Rooms.meta	24/04/2020 19:21	Archivo META 1 KB

Figura 6. Carpeta ScriptableObjects



4.2.4 Scripts

La parte más importante del trabajo se encuentra en la carpeta **Scripts** (mostrada en la figura 7), donde se encuentran ordenados en carpetas los distintos sistemas que previamente hemos explicado. Los scripts fuera de las carpetas correspondientes a los sistemas tienen un uso genérico en el plugin, es decir, se accede a ellos desde varias partes del código y no pertenecen a ningún sistema.

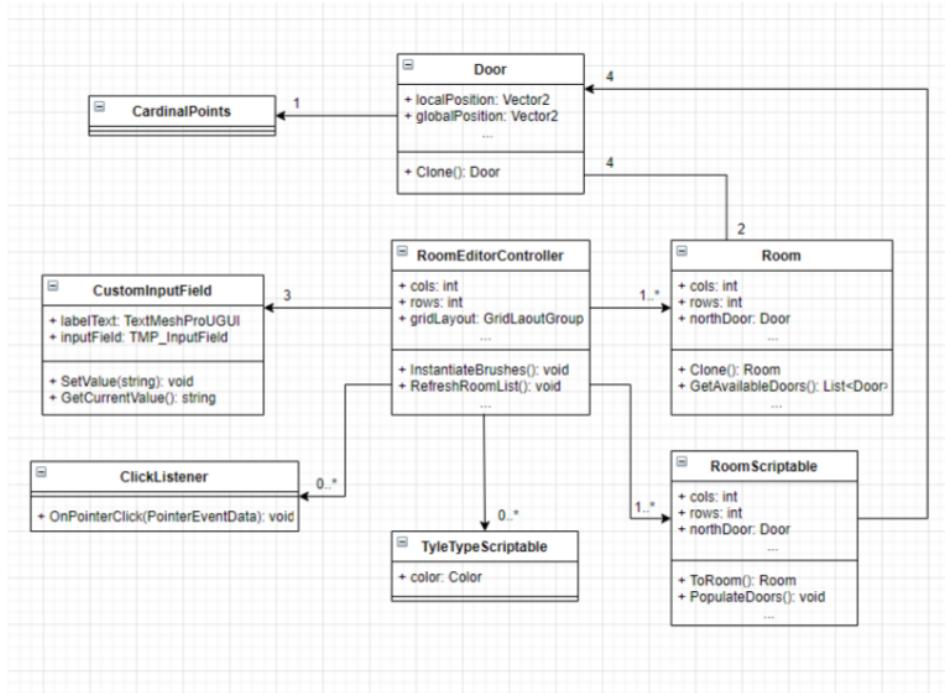
📁 DungeonGenerator	14/06/2020 4:27	Carpeta de archivos
📁 Editor	01/06/2020 20:25	Carpeta de archivos
📁 EventSystem	16/06/2020 0:00	Carpeta de archivos
📁 Extensions	26/05/2020 23:22	Carpeta de archivos
📁 RoomEditor	18/06/2020 23:27	Carpeta de archivos
📄 AssetDatabaseUtils.cs	25/05/2020 23:05	Visual C# Source f...
📄 AssetDatabaseUtils.cs.meta	25/05/2020 23:07	Archivo META
📄 DungeonGenerator.meta	08/04/2020 0:35	Archivo META
📄 Editor.meta	24/05/2020 1:37	Archivo META
📄 EventSystem.meta	20/05/2020 20:14	Archivo META
📄 Extensions.meta	05/03/2020 22:24	Archivo META
📄 FileUtils.cs	21/05/2020 23:42	Visual C# Source f...
📄 FileUtils.cs.meta	21/05/2020 23:30	Archivo META
📄 Matrix.cs	15/05/2020 19:06	Visual C# Source f...
📄 Matrix.cs.meta	05/03/2020 22:23	Archivo META
📄 RoomEditor.meta	01/06/2020 20:26	Archivo META

Figura 7. Carpeta Scripts

4.2.4.1 Editor de Salas (RoomEditor)

El sistema que podemos observar en la figura 8 tiene como clase principal a RoomEditorController, la cual se encarga del comportamiento general de la edición y creación de salas, con métodos como instanciar los pinceles, cargar una sala desde el dropdown, etc. Esta clase tiene una Room, una RoomScriptable, una lista de cada una de las anteriores, 3 CustomInputField (o como antes lo hemos nombrado: InputText) y un ClickListener por cada casilla que tenga la sala que se esté editando o creando. La decisión técnica de tener tanto Rooms como RoomScriptables se debe a que sus utilidades son completamente distintas, el tener las instancias de tipo Room agiliza la edición de las salas ya que permite trabajar con matrices, mientras que para almacenar bien la información de las salas utilizaremos un modo de almacenaje persistente, las RoomScriptables. Las Rooms y las RoomScriptables tienen 4 Doors (puertas), una por cada punto cardinal. Las puertas tienen una referencia a la sala a la que pertenecen, otra a la sala colindante o vecina (en caso de que la tengan) y, además, tienen asignado un punto cardinal que nos indica el lugar en el que se encuentran dentro de la sala a la que pertenecen.

1
Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.



9
Figura 8. Diagrama de clases del Editor de Salas

4.2.4.2 Generador Procedural (DungeonGenerator)

El sistema de generación procedural, esquematizado en la figura 9, tiene dos clases cuya importancia destaca sobre el resto, la primera es DungeonGeneratorController, la cual se encarga de la generación procedural de las mazmorras, y la segunda es DungeonGeneratorPanelController, para controlar el panel o interfaz del que se ha provisto a los usuarios a modo de ejemplo para la utilización de este sistema. DungeonGeneratorController tiene un DungeonTileType que utilizará para llenar todas las casillas de la mazmorra al principio por defecto. DungeonTileType no es más que un ScriptableObject que contiene la información de un tipo en concreto de casilla (suelo, pared, puerta, vacío, agujero, etc.). El usuario es libre de crear sus propios tipos de casillas para su mazmorra personalizada. En el sistema de edición de salas se recuperan todos los tipos de DungeonTileType que hay y se preparan en la pantalla para que el usuario pueda seleccionarlos y pintar las casillas de la sala según convenga. DungeonGeneratorController también tiene Modifiers, los cuales representan los distintos factores que se pueden modificar de las Dungeons, en concreto en el ejemplo de muestra, existen tres: uno representará el tamaño medio de las salas de la mazmorra, otro la cantidad aproximada de salas que deberá tener ésta y el último la disposición que tomarán las salas en la mazmorra, es decir, si la mazmorra va a ser más horizontal o vertical. En el caso en que el usuario del plugin quisiera añadir algún modificador más, deberá crear una nueva clase que herede de DungeonGeneratorController y modificar la generación de la mazmorra sobrescribiendo los métodos ya existentes o la adición de nuevos. Esta decisión técnica fue tomada ya que no se podía modularizar el código debido a que cada modificador afecta a una parte distinta de la generación.



Por ejemplo, la elección de salas a utilizar, la colocación de éstas en la mazmorra o el número total de salas a añadir. Los modificadores recuperan la información de los eventos, la procesan y se la envían al DungeonGeneratorController para modificar la generación.

DungeonGeneratorPanelController es una exemplificación del uso del plugin. Tiene un DungeonGeneratorController y es capaz de utilizar una co-rutina para no entorpecer la experiencia del jugador. La co-rutina se encarga de la generación de la mazmorra mientras el programa se ejecuta en paralelo. Dicho de otro modo, utilizamos la programación concurrente para permitir la ejecución normal del juego mientras se realizan operaciones relativamente costosas simultáneamente en la co-rutina. De esta manera evitamos los cuellos de botella o posibles parones del juego. Como ya hemos descrito anteriormente, esto es únicamente un ejemplo de cómo se podría utilizar el sistema, existiendo infinitas maneras de hacerlo y de personalizar su uso para los proyectos propios. DungeonGeneratorPanelController tiene un DungeonRoomSet, lo cual no es más que un ScriptableObject al cual le podemos añadir la cantidad de salas que queramos para nuestra mazmorra. Con ello permitimos que existan varias colecciones de salas para generar distintos tipos de mazmorras (las salas pueden aparecer en distintos sets o colecciones).

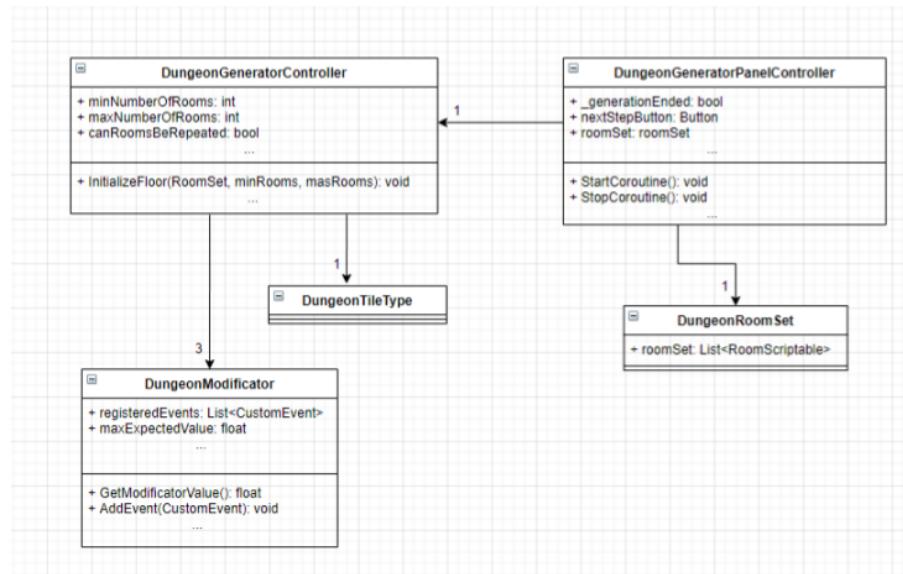


Figura 9. Diagrama de clases del Generador Procedural

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

4.2.4.3 Sistema de Eventos (EventSystem)

El sistema de eventos se basa en el concepto de herencia de clases ya que existe una clase padre llamada CustomEvent (la cual es del tipo ScriptableObject) que posee los métodos abstractos (que posteriormente serán implementados en las clases hijas) necesarios para hacer que el plugin funcione. Se ha preparado el caso de los NumericalEvents, los cuales almacenan un valor numérico, como podría ser la cantidad de veces que el jugador acaba con un enemigo, la cantidad de segundos que ha tardado en recorrer el nivel, los puntos de salud recuperados por algún power-up, etc. El desarrollador puede crear sus propios eventos o incluso sus propios tipos de evento para su proyecto.

En la parte derecha de la figura 10 podemos ver que también se ha empleado el patrón de programación Memento para poder almacenar el estado de todos estos eventos en un momento concreto del juego, como por ejemplo cuando el jugador quiera guardar partida. De esta manera, aunque el dispositivo en el que se está jugando colapse, el último GameStatus se mantendrá intacto facilitando la recuperación del último estado deseado. De este patrón forman parte las clases de GameStatus como Memento y StatusCaretaker a modo de conserje. La clase que tomará el rol de Originador se encontrará en el juego del usuario y será la que llame a StatusCaretaker para almacenar el estado del juego.

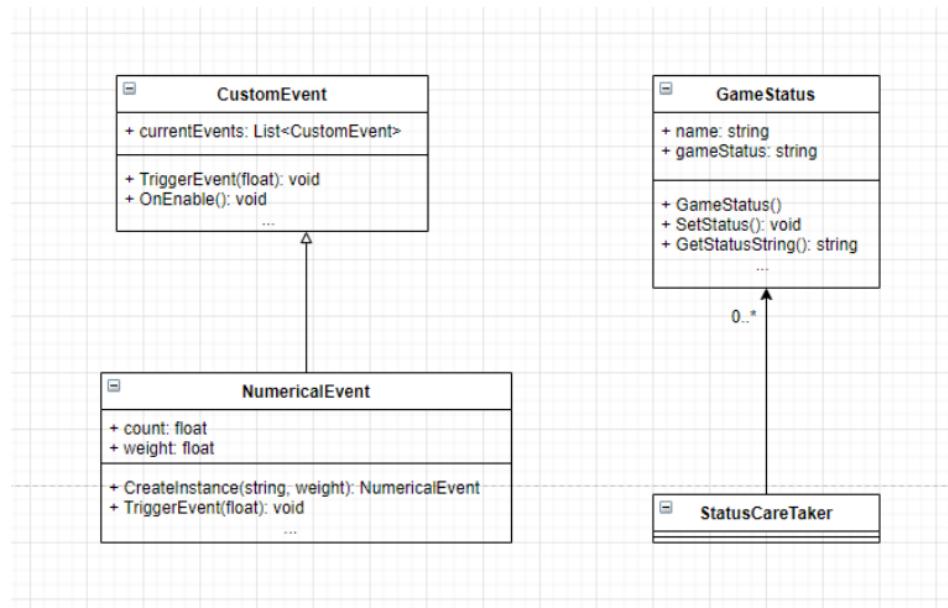


Figura 10. Diagrama de clases del Sistema de Eventos

4.3 Tecnologías Utilizadas

20

En el siguiente apartado se expondrán las herramientas utilizadas para la realización de este proyecto:



4.3.1 Git

Para controlar las versiones del proyecto se utilizó Git [20], software libre diseñado por Linus Torvalds e ideado para el mantenimiento de versiones de manera eficaz y confiable cuyo propósito es el de reflejar el registro de cambios en los archivos anotando toda la información necesaria sobre estos (fecha, autor, número de ficheros alterados, etc.).

En concreto, se utilizó el sistema integrado en el IDE de programación multiplataforma Rider. A pesar de no ser un trabajo en equipo, se tomó la decisión técnica de utilizar dicho controlador de versiones para facilitar la vuelta a un estado deseado en caso de errores fatales durante el desarrollo del mismo, y debido a que éste brinda una visión más clara de las tareas realizadas en cada etapa del proyecto. También, se empleó en los inicios del proyecto GitHub a modo de servidor para almacenar el repositorio. Con ello se aseguró el progreso frente a cualquier tipo de fallo y/o corrupción de la memoria dónde se almacenaba éste.

Al no trabajar en equipo, la metodología de uso de git no seguía un estándar complejo, ya que la única persona con acceso al repositorio era yo y la posibilidad de sobrescribir el trabajo de otra persona era nulo. El uso habitual consistía en cada vez que alguna funcionalidad o tarea era finalizada se procedía a realizar un commit con los últimos cambios realizados, previo a ello (para no perder la costumbre con el punto de vista fijado sobre el futuro laboral) se realizaba un pull, aunque como ya hemos señalado anteriormente no se dio el caso en que la información del repositorio local fuera diferente a la del repositorio online. Finalmente, se realizaba un push para actualizar la información del repositorio online.

Aun así, en determinadas ocasiones se tuvo que hacer uso de la consola de git para solventar algunos problemas causados por el archivo .gitignore.

A continuación, en la figura 11 se muestra un extracto de la consola con los últimos commits realizados.

1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

```
$ git log
commit dc2953ac064e966e5b9bad52e5ffb4f83fc4a37b (HEAD -> master)
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Sun Jun 14 04:30:50 2020 +0200

    Final commit for TFG

commit 8e913f81fb963837f41a7daeb3966b855ecf2dde
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Wed May 27 00:47:57 2020 +0200

    DungeonModificators Implemented
    To Do: Polish VerticalShapeModifierator

commit 90bd955f061279b38fe269931bc2e76609bf8918
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Sun May 24 01:47:21 2020 +0200

    DungeonRoomSet.cs created

commit 06f84c18fda278ab20b24846ac68c2bce81f98e3
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Sun May 24 01:16:19 2020 +0200

:....skipping...
commit dc2953ac064e966e5b9bad52e5ffb4f83fc4a37b (HEAD -> master)
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Sun Jun 14 04:30:50 2020 +0200

    Final commit for TFG

commit 8e913f81fb963837f41a7daeb3966b855ecf2dde
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Wed May 27 00:47:57 2020 +0200

    DungeonModificators Implemented
    To Do: Polish VerticalShapeModifierator

commit 90bd955f061279b38fe269931bc2e76609bf8918
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Sun May 24 01:47:21 2020 +0200

    DungeonRoomSet.cs created

commit 06f84c18fda278ab20b24846ac68c2bce81f98e3
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Sun May 24 01:16:19 2020 +0200

    EventSystem created

commit 6040b0bcf3ce42e07deedf83deca5e1deb93c30e
commit dc2953ac064e966e5b9bad52e5ffb4f83fc4a37b (HEAD -> master)
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Sun Jun 14 04:30:50 2020 +0200

    Final commit for TFG

commit 8e913f81fb963837f41a7daeb3966b855ecf2dde
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Wed May 27 00:47:57 2020 +0200

    DungeonModificators Implemented
    To Do: Polish VerticalShapeModifierator

commit 90bd955f061279b38fe269931bc2e76609bf8918
Author: Alejandro Penya <alexpenya13@gmail.com>
Date:   Sun May 24 01:47:21 2020 +0200

    DungeonRoomSet.cs created
```

Figura 11. Últimos commits del proyecto



4.3.2 Unity

Unity [12] es el motor para el desarrollo de videojuegos más utilizado en la creación de juegos indie (como se observa en la figura 12) ya que provee de varias herramientas y características para el diseño de prácticamente cualquier tipo de videojuego, pudiendo generar gráficos tanto en 3D como en 2D. Empezó siendo utilizado como una manera sencilla de desarrollar aplicaciones móviles, pero actualmente se usa para crear multitud de juegos distintos para diferentes plataformas.

The most popular engines for indie games

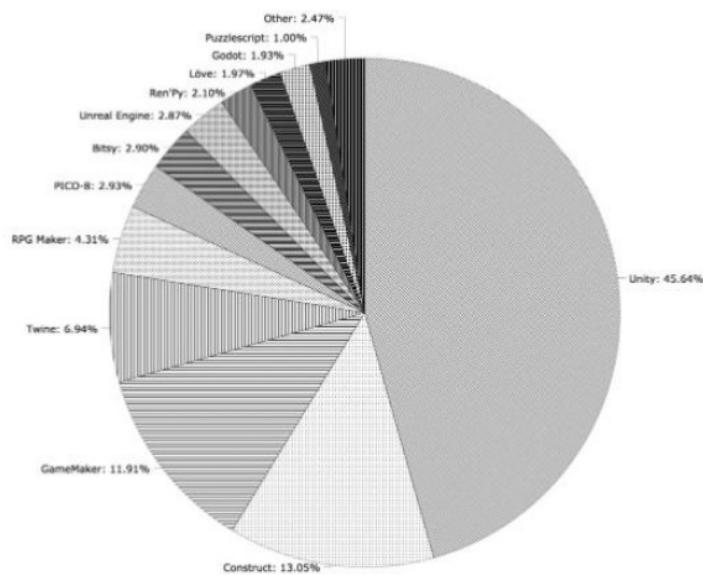


Figura 12. Motores más utilizados 2018. [21]

Con este motor los creadores, según sus propias palabras [22], pretenden democratizar el desarrollo de videojuegos, permitiendo a cualquier persona descargar el mismo y crear sus propios proyectos. La licencia gratuita que ofrece Unity Technologies está destinada a freelancers y empresas indies que no superen una cantidad de ingresos anuales de 100.000 dólares, en caso de superar esta cantidad, la licencia PRO costaría alrededor de 1500 dólares. Dicha información se muestra justo al iniciar por primera vez el programa en los términos y condiciones de uso.

Gracias a esto pequeñas empresas que no sería capaces de crear su propio motor gráfico (ya que tendrían que dedicar años para ello), pueden empezar en la industria de manera gratuita hasta

1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

ser capaz, con los ingresos generados hasta el momento, de pagar la licencia de ese o incluso de otro motor que requiriesen. De esta manera, podemos dar cuenta de la importancia de Unity dentro de este mundo, ya que muchos de los principiantes en el desarrollo de este tipo de software lo tienen como primer punto de contacto. No es únicamente el hecho de que sea el motor más influyente en el sector de los juegos indies, sino que gran parte de la comunidad de programadores de dicha industria conoce, en mayor o menor medida, los entresijos de Unity.

La decisión de utilizar este framework en lugar de otros como Unreal Engine 4 o CryEngine [23], fue debido a tres factores principales:

-El target principal de este proyecto son los desarrolladores freelancers, pequeñas compañías, principiantes, etc. por lo que, como ya se ha comentado previamente, la elección más lógica es utilizar el motor más usado por los mismos.

-Dreamcatchers fue originalmente desarrollado en Unity, con lo que ya tenía alguna noción básica de este (o al menos eso pensaba antes de iniciar el proyecto).

-Era una oportunidad perfecta para prepararme de cara al futuro laboral, no solo mejorando mis conocimientos sobre Unity y el desarrollo general de los videojuegos, sino que además añade mucho valor a mi portfolio personal y currículum.



4.3.3 Rider

Rider [24] es un IDE .NET multiplataforma basado en la plataforma IntelliJ.

Rider posee una herramienta de análisis de código que supervisa la solución entera y muestra las incidencias de todos los archivos incluso si no están abiertos. Además, cuenta con más de 2200 inspecciones de código, con sus correspondientes arreglos rápidos automatizados que permiten agilizar mucho el proceso de la programación.

La edición de código con este IDE es muy sencilla ya que ofrece varias plantillas de código, inserción automática de correspondencias de llaves y directivas de importación, acciones de contexto y diferentes tipos de finalización de código entre otras muchas funcionalidades.

Respecto a la refactorización, Rider tiene más de 450 acciones contextuales. Pudiendo extraer métodos, cambiar nombres, mover y copiar tipos, interfaces y clases, utilizar una sintaxis alternativa, etc.

Una de las características clave para la elección de este IDE es su depurador ya que funciona realmente bien con aplicaciones .NET Framework, Mono y .NET Core. Al utilizar también Unity para el desarrollo del proyecto, la sinergia entre estos dos hizo que la elección del IDE fuese obvia. El depurador permite evaluar, comprobar, ver y entrar a las distintas variables del código de una manera muy sencilla mientras Unity se ejecuta. De esta manera recibimos un input visual muy claro a la hora de arreglar funcionalidades inesperadas (explicado con más profundidad posteriormente en este mismo apartado).

Durante la elección de las distintas tecnologías no sabíamos qué tipos de bases de datos se iban a emplear, pero se escogió ya que Rider permite trabajar con SQL, conectarse con bases de datos, realizar consultas, externas y editar esquemas .

Con Rider la búsqueda dentro del proyecto es muy rápida ya que se puede saltar a cualquier archivo, tipo o miembro de su base de código de manera intuitiva y sencilla. Para ello podemos hacer clic derecho sobre el elemento de interés o utilizar los distintos atajos de teclado disponibles para, por ejemplo, buscar el lugar de declaración de una variable, las distintas llamadas de un método en toda la solución o ir a la clase de la que se hereda.

Rider admite una gran cantidad de complementos desarrollados para IntelliJ. Además de los complementos incluidos hay disponibles complementos compatibles con archivos .gitignore (utilizados para la gestión del repositorio en GitHub), Markdown y secuencias de comandos Python.

Se decidió utilizar Rider debido a la gran sinergia que tiene con Unity. Por ejemplo, para depurar únicamente deberemos presionar el botón <<Debug: Attach to Unity Editor>> (figura 13) o el shortcut Shift+F9 (depende de la configuración), añadir los puntos de parada deseados y automáticamente al ejecutar en el motor nuestro proyecto, cuando se alcance/n la/s línea/s de código marcada/s, Unity se detendrá en el momento exacto y nos permitirá, desde Rider, observar el estado actual de la ejecución y de las variables.

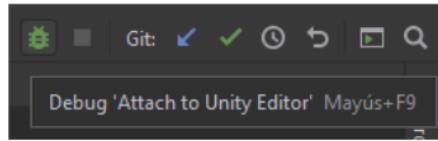


Figura 13. Debug 'Attach to Unity Editor'

Además, su interfaz es notablemente más intuitiva que otros frameworks por funcionalidades como:

Ayudas de refactorización: En muchas ocasiones Rider nos ayuda a generar un código más limpio y funcional con sugerencias sobre cómo refactorizar nuestro código. La parte por modificar queda subrayada en verde indicándonos que existe una mejor manera de escribirlo; a continuación, con un simple clic derecho, se muestran las distintas opciones. En la figura 14 se muestra el caso de un if que se puede invertir para reducir el anidamiento.

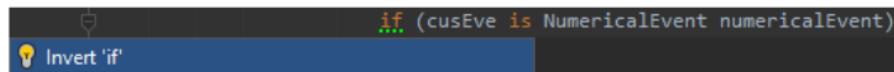


Figura 14. Ayudas Refactorización

1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

Ayuda visual de variables: Al escribir en C# muchas veces declaramos variables sin indicar de qué tipo van a ser las mismas, empleando la palabra reservada <>var<>. Con Rider podremos ver una pequeña etiqueta colocada a la derecha del nombre de la variable con el tipo de valor que se va a asignar a ésta. También sucederá lo mismo dentro de los paréntesis de un método en cada uno de los argumentos que este necesite, haciendo muy visible que información recibe cada uno de ellos. En la siguiente figura podemos ver un ejemplo visual de esta ayuda.

```
var spaceToCheck :Matrix<(TileTypeScriptable, Room)> = _floorMatrix.GetSubMatrix( initialCol: initialPosition.x,
    initialRow: initialPosition.y, colLength: room.cols, rowLength: room.rows);
var result :bool = spaceToCheck.All( selector: (item :(TileTypeScriptable, Room) , col :int , row :int ) => item.Item1 == emptyTile);
var test :int = spaceToCheck.ToList().Count( condition: x :(TileTypeScriptable, Room) => x.Item1 != emptyTile);
return result;
```

Figura 15. Ayuda Visual de Variables

Read-Only Properties: dentro de este framework, es posible utilizar las Read-Only properties de manera automática, lo que aligera de manera notable la creación de clases. Podemos tener variables privadas dentro de una clase, las cuales no queremos que sean modificadas desde fuera de esta clase pero a su vez se puede acceder a estas variables desde distintas partes del código. Para solucionar esta necesidad podemos utilizar las Read-Only properties, las cuales actúan como getters (asignando a estas propiedades una referencia a las variables privadas). De esta manera podremos acceder a la información requerida sin alterarla por error, generando un código más seguro. A continuación, en la figura 16 se muestra la manera de implementar estas propiedades:

```
[SerializeField] private float count; ← Serializable
[2 usages] ↗ Alejandro Penya
public float VisibleCount => count;
```

Figura 16. Read-Only Properties

Configuración: Al instalar el programa se nos da la posibilidad de elegir entre 3 configuraciones de shortcuts (los cuales pueden ser modificados a voluntad posteriormente) y 5 distintos estilos de apariencia: Darcula, Hight Contrast, IntelliJ, Rider Dark o Rider Light. Además, cuenta con un pequeño apartado de Accesibilidad para gente con deficiencia visual: invíidentes y daltónicos. La figura 17 es una captura de varias de las opciones de las que Rider nos provee.



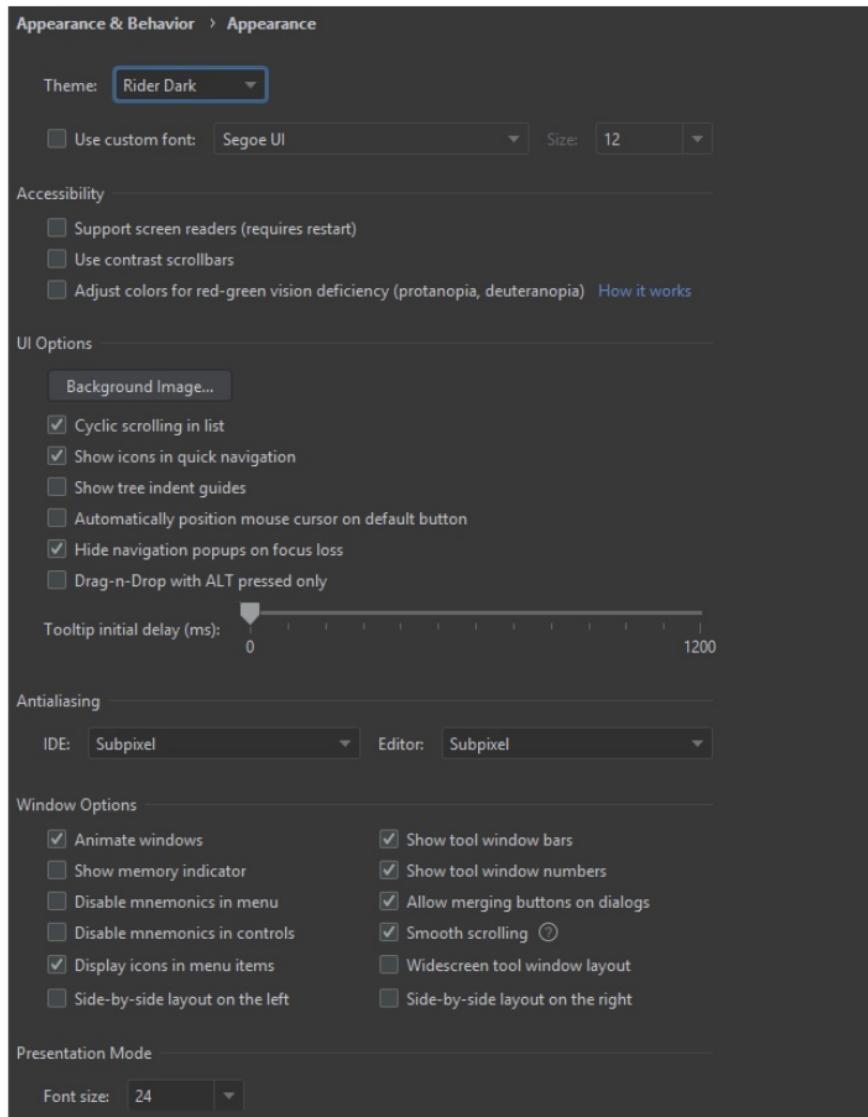


Figura 17. Configuración Rider

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

4.3.4 Trello

Trello [25] es un software de gestión y administración de proyectos que emplea el sistema Kanban para el registro de las distintas actividades a realizar. Éstas se representan con tarjetas virtuales que se desplazan por las listas del tablero, las cuales representan los posibles estados de las tareas y se pueden crear y personalizar por el equipo de trabajo. También se permite agregar comentarios, avisar por email a los involucrados cuando una tarea se desplaza de lista o incluso adjuntar archivos. De esta manera el equipo al completo puede tener una visión global del proyecto en el que trabaja, ayudar a los posibles rezagados, pedir ayuda a sus compañeros, etc.

La aplicación es versátil y fácil de usar, ya que se pueden personalizar los tableros de manera muy sencilla para reflejar la manera de trabajar del equipo. Además, la aplicación consta de una interfaz web y cliente tanto para IOS (Disponible desde 2011) como para Android (desde 2012). Podemos ver como luce la aplicación en la figura 18.



Figura 18. Trello

2

5. Desarrollo de la aplicación

En este apartado abarcaremos el desarrollo del plugin desde el inicio de este, sin tener en cuenta lo ya mencionado en el apartado 1.1 Motivación. Los comienzos del proyecto fueron a principios de febrero, tras estudiar la viabilidad junto con la tutora. En la siguiente figura podremos observar la distribución de tiempo real que ocupó cada paso del proyecto:



Figura 19. Calendario del proyecto

Tras una reunión y un listado de consejos se procedió al **diseño de la estructura del proyecto**. Se subdividió éste en tres grandes partes correspondientes a los distintos sistemas que forman el plugin; el generador procedural, el sistema de eventos y un editor de salas que ayudará al desarrollador a tener una base de datos de salas de manera intuitiva y sencilla. Esta tarea duró

1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

desde el día 12 de febrero hasta el día 20 del mismo mes. Debido a problemas personales graves, hubo un parón durante la realización de este paso y por ello el tiempo empleado superó el destinado para esta actividad.

Con el calendario trastocado se continuó con la siguiente fase del proyecto, el **diseño de un backlog** con las distintas tareas a realizar. Este paso se realizó desde el día 21 de febrero hasta el día 26 del mismo mes, tiempo en el que se desglosó por completo la estructura del proyecto y se creó un plan basado en tres sprints, uno por sistema a desarrollar. Se estimó que el primer sprint, en el que se desarrollaría el sistema de generación procedural, duraría alrededor de un mes y medio (27 de febrero – 9 de abril) debido a la complejidad y a que sería la primera pieza del proyecto, con lo que gran parte del tiempo se dedicaría a la investigación de las distintas herramientas disponibles en Unity y la continuación del aprendizaje de C#. El segundo sprint dispondría de un mes para su realización (un 33% menos que el anterior) desde el 10 de abril hasta el 10 de mayo, ya que, a pesar de su dificultad, la experiencia con el sprint previo debería haber servido para agilizar del desarrollo de esta parte. Y por último el tercer sprint, el cual aunaba tanto el sistema de eventos como la integración de los anteriores, debería llevarse a cabo entre el día 11 de mayo y el 1 de junio (3 semanas) siendo el más sencillo de todos, pero con la dificultad de los potenciales errores al unir los distintos sistemas. Al final de la explicación de cada sprint aparecerá una figura con el tablero Kanban correspondiente de cada uno.



Figura 20. Sprints Trello

El primer sprint comenzó el día estimado y acabó 4 días antes de lo previsto (4 de abril). Tal y como se esperaba, una gran cantidad de tiempo fue invertida en el estudio y aprendizaje de una manera más profunda de Unity y de C#. Aparecieron varios problemas relacionados con las tecnologías ya mencionadas como el caso del mantenimiento de los datos al cerrar el programa. Para solucionarlo se investigó en internet, encontrando un vídeo [26] en el cual se explicaba qué eran los Scriptable Objects y de qué manera se podían utilizar para solucionar el obstáculo que había detenido el desarrollo. La decisión de tener tanto clases normales como clases ScriptableObject que representan la misma información es debido a que las clases normales son más rápidas de procesar y eficientes a la hora de extraer información de ellas en runtime ya que no estarán nunca serializadas, mientras que las clases ScriptableObject sí serializan la información con lo que podemos mantenerla, aunque se apague el dispositivo donde ésta se está almacenando.

La decisión técnica relacionada con los Modificators se encuentra explicada en el apartado 4.2.4.2 Generador Procedural (DungeonGenerator). Otras decisiones técnicas implementadas durante este sprint son, por ejemplo:

- Varianza: Se añadió un valor extra `c25` modifica la cantidad de salas de la mazmorra la cual se aplica al número ya calculado entre el máximo y el mínimo establecido por el desarrollador. Es decir, si la varianza establecida (cuyo nombre en el editor de Unity es Room Count Variance) es de 4 y el número de salas elegido por el algoritmo es 15 (este número se encuentra entre el mínimo y el máximo seleccionados por el desarrollador), el código elegirá de manera aleatoria un número entre -4 y 4 y se lo añadirá a 15, por ejemplo $(-2) + 15 = 13$. De esta manera nos aseguramos de crear unas condiciones difícilmente repetibles para generar una sensación en el jugador de que cada mazmorra que transita es nueva y no aparecen repeticiones que puedan hacer monótona la experiencia.

- Repetición de salas: Siguiendo la misma idea del apartado anterior se quiso dar un paso adicional y valoramos que, aunque la repetición de salas puede generar la misma sensación en los jugadores, también es cierto que crear grandes cantidades de salas resultaría muy costoso para los desarrolladores. Por este motivo se añadió una variable booleana que en el caso de ser true permite la reiterada aparición de una misma sala en la misma mazmorra y en caso de ser false escoge en cada iteración una sala que no se encuentre ya en la mazmorra.

En la figura 21 se muestra como aparecen estos valores en el editor de Unity.

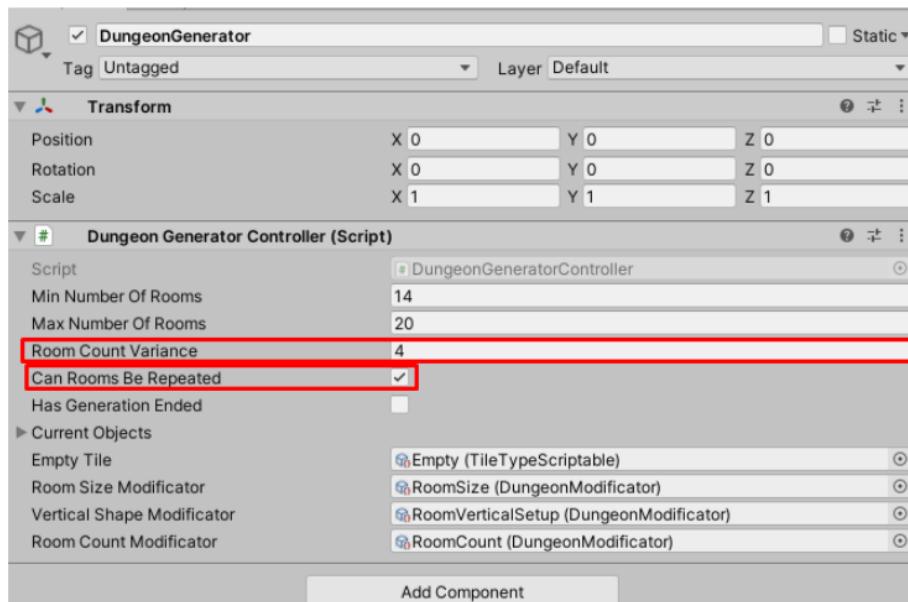


Figura 21. Decisiones técnicas sprint 1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

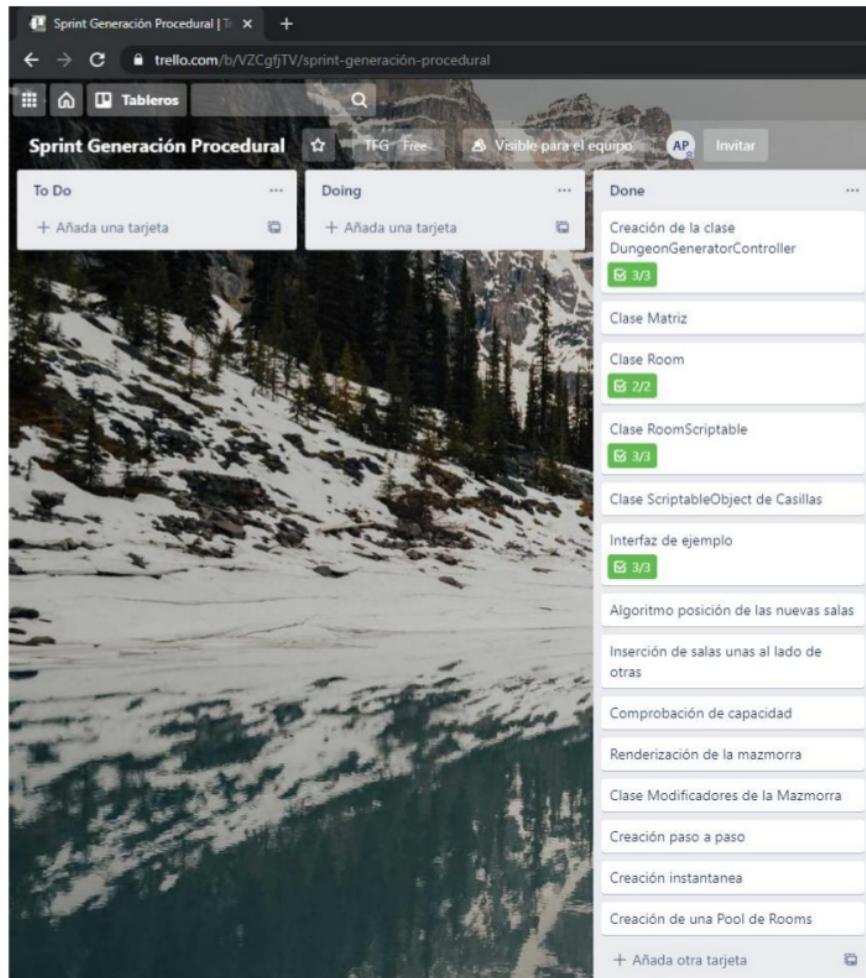


Figura 22. Tablero Kanban Sprint 1

Al finalizar el primer sprint antes de tiempo, se recuperó el perdido durante el primer paso del trabajo. El segundo sprint se inició el día 5 de abril, pero finalizó un día después del estimado a pesar de que comenzó varios días antes de lo previsto. Es decir, se emplearon seis días más de los establecidos para finalizarlo. Esto fue debido a que era la primera vez que creábamos un sistema destinado a ser utilizado desde el propio editor de Unity y surgieron una serie de errores ya que había que buscar una manera de que la parte del plugin que solo se necesitaba en el editor no formara parte de las builds del proyecto donde fuesen a ser utilizadas. El motivo de ello es que a los usuarios de los juegos que se desarrollen usando esta herramienta no les interesa tener un gasto de memoria extra con sistemas y datos que no van a requerir mientras jueguen a estos. Para evitar ese malgasto de memoria se empleó la carpeta “editor” de Unity la cual no se añade al buildear la solución. Pero con ello surgieron otros problemas ya que este sistema comparte ciertas clases, como la clase Room, con el resto de la aplicación así que se tuvo que investigar qué clases podían guardarse en esta carpeta “editor” y cuáles no para mantener el correcto funcionamiento del plugin.

Respecto a las decisiones técnicas del segundo sprint se estableció que el máximo tamaño de las salas sería de 20x20 ya que si fueran más grandes sería muy incómodo utilizar el editor de salas. Esta limitación consideramos que no es crítica ya que existen alternativas para hacer las salas más grandes. Por ejemplo, a la hora de renderizar la mazmorra se pueda crear una equivalencia de forma que cada casilla de la mazmorra equivalga a 4 de las renderizadas. También es cierto que, en principio, y no tiene sentido crear salas inmensas en un juego de mazmorras ya que el propio concepto de mazmorra requiere de pequeñas salas con pasillos laberínticos para dificultar el tránsito por ellas. Los juegos de mazmorras más exitosos como The Binding Of Isaac o las diferentes entregas de Diablo [27] siguen este principio.

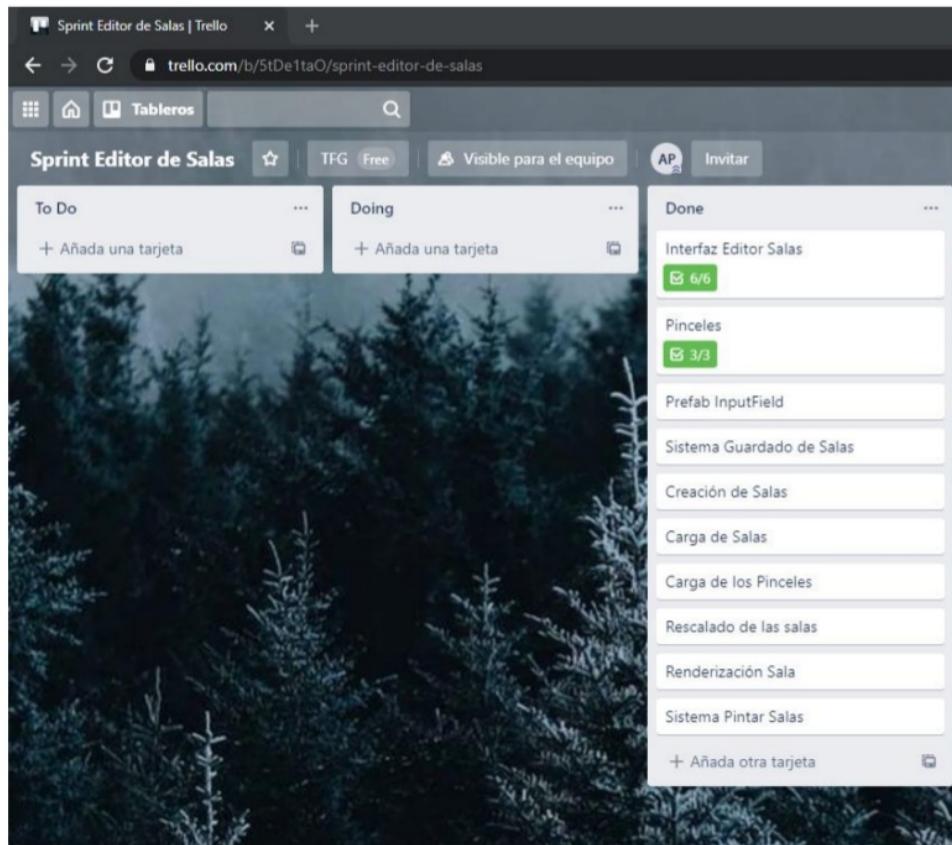


Figura 23. Tablero Kanban Sprint 2

El tercer y último sprint empezó un día después y acabó un día antes de lo previsto dándonos un día extra para el resto de los pasos del proyecto. Como ya se preveía fue el sprint más corto. El desarrollo del sistema en sí fue breve, sin embargo, la mayor cantidad de días de este sprint se emplearon en la integración de todas las partes del plugin. Aparecieron diversos problemas, pero al haber realizado un diseño de la arquitectura previo y un desglose del proyecto bastante profundo, la solución a éstos se halló de forma bastante rápida. Por ejemplo, se tuvo que sustituir

1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

la Pool de Rooms por una manera de darle al generador las salas concretas que formarían parte de las mazmorras. Así surgió la idea de crear Sets de Rooms (RoomSets), los cuales además son ScriptableObjects ya que tienen su propia funcionalidad. Si los colocamos en una carpeta con Rooms y los seleccionamos, observaremos que en el inspector de objetos de Unity aparece un botón que dice: “Add rooms to RoomSet from this folder”. Si presionamos el botón se añadirá una referencia de cada una de las salas del directorio a la lista de Rooms del set.

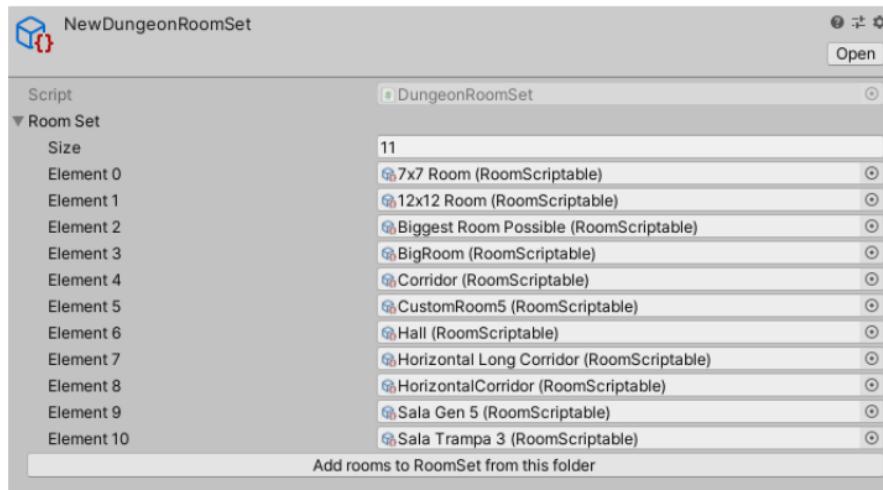


Figura 24. RoomSet

Para mantener la posibilidad de personalización del plugin para los proyectos de los usuarios se tomó la decisión técnica de crear una clase padre de eventos “CustomEvent” y crear una clase que heredase de esta llamada “NumericalEvent”. De esta manera también se permite la creación de nuevos tipos de eventos por parte de los desarrolladores.

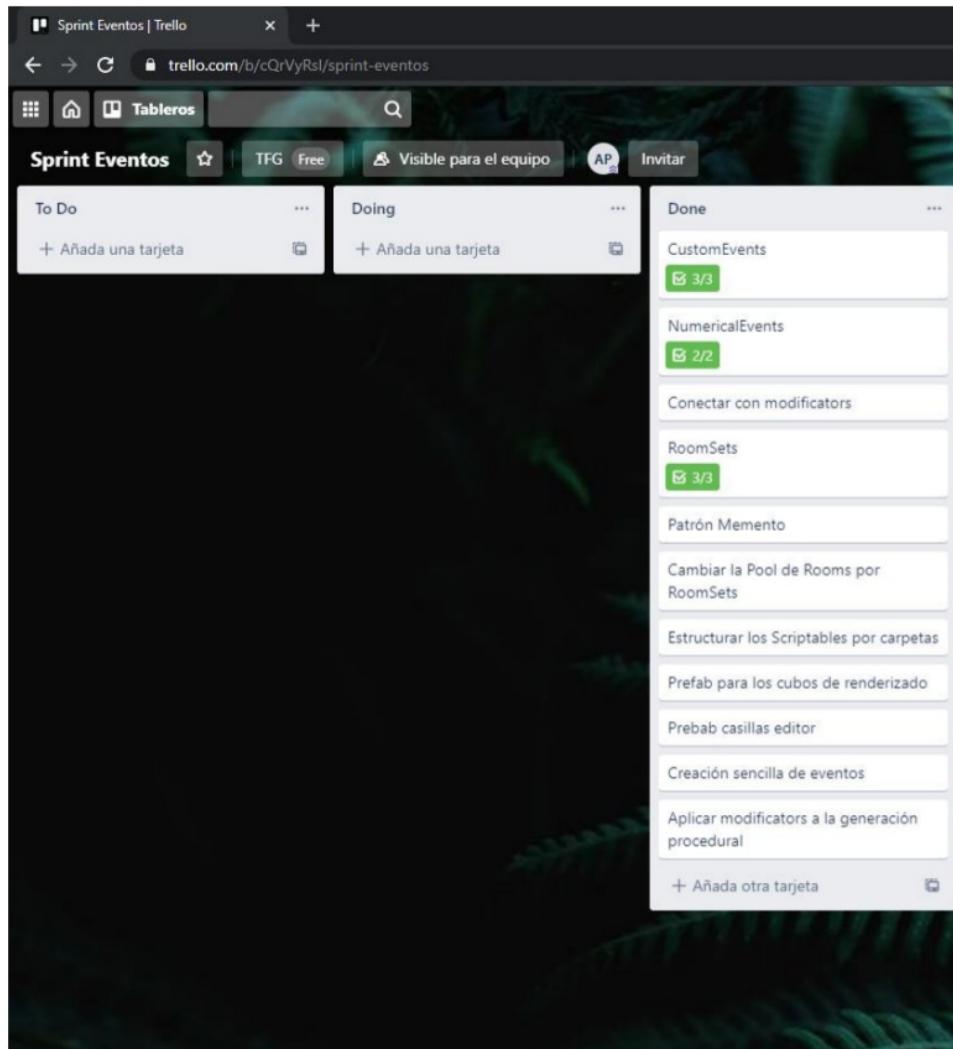


Figura 25. Tablero Kanban Sprint 3

En este punto del proyecto se consideró adecuado desarrollar un pequeño juego que utilizará el plugin para la generación del nivel que pudiera servir de prueba de concepto. El juego consiste en una deconstrucción del buscaminas, donde se pueden recorrer todas las casillas del nivel evitando las minas con la misma mecánica que el juego clásico (aparecen números en las casillas que se van descubriendo que indican la cantidad de minas colindantes que tiene). Con esto mostramos que no es necesario usar el plugin únicamente para videojuegos Rogue-Like o juegos basados en mazmorras, se puede utilizar para múltiples tipos de proyectos debido a la amplia posibilidad de personalización.

Junto al desarrollo de este juego se continuó y finalizó la redacción de la memoria recabando toda la información escrita mientras se realizaba ProDungeon, los UML, capturas de las distintas tecnologías utilizadas y los tableros Kanban correspondientes a cada sprint.

1

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

6. Conclusiones

Para observar los resultados del trabajo primero trataremos cada uno de los objetivos principales por separado para mostrar el progreso relacionado con ellos:

-Proveer de una herramienta útil a los desarrolladores: Este objetivo sí que se ha completado ya que se ha completado la creación del plugin ProDungeon y se ha distribuido entre varios desarrolladores conocidos para su utilización en futuros proyectos.

-Incitar a los desarrolladores a explotar el plugin: Desgraciadamente este objetivo no se ha podido cumplir ya que el plugin no se ha podido subir a la tienda de Unity por un error en la estimación de tiempos. Aunque varios desarrolladores conocidos lo estén probando ahora mismo, no es suficiente como para cumplir este objetivo.

-Disponer de una herramienta para futuros proyectos: se ha completado este objetivo con creces ya que no solo la herramienta queda disponible para futuros proyectos, sino que incluso se ha demostrado su funcionamiento con un pequeño videojuego de muestra.

-Mejorar mis conocimientos del motor: no es cuantificable la cantidad de conceptos nuevos aprendidos, pero sí que se puede demostrar una notable mejoría en los conocimientos del motor ya que sin ellos no se podría haber realizado ProDungeon. Al leer la memoria se demuestra el progreso en las explicaciones de partes técnicas o utilidades nuevas aprendidas a lo largo del proyecto.

-Mejorar la generación de niveles de Dreamcatchers: se está procediendo a insertar el plugin en Dreamcatchers simultáneamente a este trabajo, pero al tener menor prioridad que este todavía se encuentra al 50% aproximadamente.

-Lucro económico: No se ha percibido ningún beneficio del plugin por el momento ya que este objetivo tenía una proyección futura, requiriendo de la publicación de la aplicación en la tienda de Unity, el feedback de los usuarios y su mantenimiento.

Es necesario reflexionar sobre la estimación que se realizó al inicio del proyecto para aprender de los errores.

El primer paso del plan diseñado al comienzo del trabajo (**diseñar la arquitectura producto**) tenía asignado 1 punto y acabó costando el equivalente a 3 puntos por problemas personales. A pesar de ser problemas inesperados, se deberían de dar un 10% o 20%¹⁹ tiempo extra para ser capaces de hacer frente a cualquier inconveniente que se presente en el desarrollo de cada una de las partes del proyecto tanto a nivel personal como a nivel técnico.

El segundo paso, en el que **creó el backlog**, fue etiquetado con un total de 3 puntos y ocupó 1 punto menos de lo estimado, por lo que la estimación en este caso fue superior al coste real por poco. Este tiempo extra se habría utilizado en caso de encontrar cualquier tipo de problema durante la realización del paso, pero acabó sirviendo para equilibrar las pérdidas temporales del anterior.



El tercer y cuarto aparecen unidos en el calendario debido a la distribución temporal en sprints. Se emplearon un total de 32 puntos entre el **desarrollo** y la **integración**, mientras que la estimación inicial se fijó en 31 puntos. Es sorprendente la cercanía entre estos dos valores, ya que al ser la parte más grande y compleja del proyecto era muy difícil dar una estimación exacta. Esto puede ser debido a la experiencia en desarrollo de aplicaciones adquirida en los últimos 2 años de la carrera, las prácticas en empresa y la experiencia Erasmus.

Finalmente, los últimos dos pasos del proyecto no se pudieron llevar a cabo debido al desarrollo del juego usando ProDungeon, la redacción y maquetación de la memoria. En un futuro se debería prometer menos al cliente siendo realistas con las capacidades y recursos del equipo.

A modo de conclusión general, ha sido una experiencia muy enriquecedora ya que no solo he mejorado mi portfolio y mis conocimientos de cara a mi futuro laboral, si no que he sido capaz de rescatar muchos conceptos y prácticas aprendidas a lo largo de la carrera para crear un proyecto por mí mismo que será útil en un futuro próximo. También me ha ayudado a posicionarme desde los distintos puntos de vista de los involucrados en un equipo de desarrollo software, así como el Scrum Master, el productor o producer, el desarrollador, etc. De esta manera en un futuro, cuando me encuentre trabajando con un equipo, será mucho más fácil para mí entender cómo se sienten y cómo piensan los distintos integrantes.

2

6.1 Relación del trabajo desarrollado con los estudios cursados

En este apartado se destacarán únicamente las asignaturas que han impregnado su huella en este trabajo de manera más notoria o bien por su relación directa con este o por los valores instruidos.

En primer lugar, la asignatura Lenguajes, Tecnologías y Paradigmas de la programación nos motivó a aprender diversos lenguajes y tecnologías, a parte de los ya instruidos en la misma. Durante el desarrollo de este trabajo he tenido que aprender a programar en C# y a utilizar el motor Unity siguiendo los valores aprendidos en esta asignatura.

Se emplearon nociones básicas aprendidas en las asignaturas de Concurrencia y Sistemas Distribuidos, y Computación Paralela, de las cuales se tomó la idea de utilizar las co-rutinas de Unity para no entorpecer la experiencia de juego, generando las mazmorras en paralelo.

De la asignatura Interfaces Persona Computador se utilizaron los principios básicos de las interfaces como la “prevención de errores” y el “Reducir la carga de la memoria a corto plazo”. Para ello en las distintas interfaces del plugin únicamente aparecen activas las opciones que el usuario puede utilizar, desactivando el resto como por ejemplo antes de crear una sala, el botón de guardar la sala aparece en gris y no es seleccionable por el usuario. [28]

Para poder gestionar bien la transición de las interfaces para saber cuándo activar y desactivar los distintos elementos de éstas se dibujó un grafo parecido a los que se utilizaron durante la asignatura Teoría de Autómatas y Lenguajes Formales para determinar los distintos estados posibles de las interfaces y entender cuáles serían las posibles transiciones entre éstos. De esta



Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

manera se ahorró mucho tiempo a la hora de probar las interfaces y solucionar errores relacionados.

También se emplearon técnicas aprendidas en las asignaturas de Ingeniería del Software y Análisis y Especificación de Requisitos a la hora de detectar cuáles iban a ser las necesidades de los usuarios del plugin (por todo ello surgió la idea de hacerlo modular y personalizable) y la creación de la arquitectura del proyecto, usando diversos diagramas UML.

Gracias a las asignaturas Gestión de Proyectos, Proceso Software y Proyecto de Ingeniería de Software el proceso de gestión del proyecto, utilización de metodologías ágiles, análisis de mercado, etc. fue sencillo y rápido, al ya haber realizado trabajos de este estilo en cada una de las asignaturas.

Gran parte de este proyecto se basa en un gran sistema inteligente que recupera información y la procesa de manera automática, “aprendiendo” sobre cómo juega el jugador y ajustando la dificultad en base a ello. No se podría haber realizado sin los conceptos aprendidos en la asignatura Sistemas Inteligentes.

Por último, se emplearon varios patrones como el ya explicado anteriormente en el apartado 4.2.4.3 Sistema de Eventos (EventSystem) aprendidos en la asignatura Diseño de Software y durante mi estancia Erasmus.



7. Trabajo futuro

Los futuros proyectos relacionados con el plugin (los cuales no son excluyentes entre sí) que se divisan en el horizonte temporal son:

17

-**Nuevas funcionalidades:** Creación de nuevas features para la mejora de la experiencia del usuario y la satisfacción de las necesidades propias de su proyecto.

-**Despliegue de la aplicación:** Debido a los tiempos de entrega y el coste inesperado de algunas partes del proyecto la parte final de este no se ha cumplido, requiriendo de una posterior subida del plugin a la tienda de Unity.

-**Mantenimiento del Plugin:** Se prevé un mantenimiento de la aplicación basado en las sugerencias de los usuarios a través de la tienda de Unity. Dentro de este mantenimiento se encuentran tareas como la solución de posibles bugs o funcionalidades inesperadas del programa, pulimento de los distintos sistemas del plugin, y la resolución de dudas sobre el mismo.

-**Solución del problema original:** Solucionar el problema existente con el videojuego Dreamcatchers ya que fue una de las motivaciones iniciales de este trabajo.

-**Creación de juegos:** El plugin será empleado para el posterior desarrollo de videojuegos usando la plataforma Unity para mejorar el currículum vitae del alumno.

Fecha de consulta:



1
Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

8. Referencias

- [1] Ribera Vázquez, Víctor. (2018). Dreamcatchers: Videojuego HORDE & ROGUELIKE desarrollado en Unity para dispositivos móviles <https://riunet.upv.es/handle/10251/115084>
- [2] Juegos Horde: <https://www.giantbomb.com/horde-mode/3015-3021/> Fecha de consulta: 20 de febrero de 2020
- [3] Juegos Rogue-lite: <http://www.gamerdic.es/termino/roguelite/> Fecha de consulta: 20 de febrero de 2020
- [4] Modo Zombis, Call Of Duty: <https://www.callofduty.com/es/blackops4/zombies> Fecha de consulta: 4 de junio de 2020
- [5] Plants Vs. Zombies: Garden Warfare 2: <https://www.ea.com/es-es/games/plants-vs-zombies/plants-vs-zombies-garden-warfare-2> Fecha de consulta: 4 de junio de 2020
- [6] Firefight mode, Halo Reach: <https://halo.fandom.com/wiki/Firefight> Fecha de consulta: 4 de junio de 2020
- [7] Warhammer: End Times – Vermintide: <https://www.vermintide.com/> Fecha de consulta: 4 de junio de 2020
- [8] The Binding of Isaac: https://bindingofisaac.fandom.com/es/wiki/The_Binding_of_Isaac Fecha de consulta: 4 de junio de 2020
- [9] Nuclear Throne: <http://nuclearthrone.com/> Fecha de consulta: 4 de junio de 2020
- [10] Dead Cells: <https://dead-cells.com/> Fecha de consulta: 4 de junio de 2020
- [11] HVA Game Technology: <https://www.hva.nl/opleiding/game-development/game-development.html> Fecha de consulta: 7 de junio de 2020
- [12] Unity: <https://unity.com/es> Fecha de consulta: 4 de junio de 2020
- [13] Unreal Engine 4: <https://www.unrealengine.com/en-US/> Fecha de consulta: 4 de junio de 2020
- [14] Estilo Vancouver: https://biblioguias.uam.es/citar/estilo_vancouver Fecha de consulta: 12 de junio de 2020
- [15] Misaki Kaidan, Chun Yin Chu, Tomohiro Harada y Ruck Thawonmas. (2015). Procedural generation of Angry Birds levels that adapt to the player's skills using genetic algorithm. *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*: <https://ieeexplore.ieee.org/abstract/document/7398674>
- [16] Ricardo Lopes, Ken Hilf, Luke Jayapalan y Rafael Bidarra. (s.f.). Mobile adaptive procedural content generation: <https://graphics.tudelft.nl/Publications-new/2013/LHJB13b/LHJB13b.pdf>



- [17] Jonathon Doran e Ian Parberry. (2010). Controlled Procedural Terrain Generation Using Software Agents. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2).: <https://ieeexplore.ieee.org/abstract/document/5454273>
- [18] Travis L. Fort. (2015). Controlling randomness: Using Procedural Generation to influence player uncertainty in video games.: http://etd.fcla.edu/CF/CFH0004772/Fort_Travis_L_201505_BA.pdf
- [19] Martin Jennings-Teats, Gillian Smith y Noah Wardrip-Fruin. (2010). Polymorph: Dynamic Difficulty Adjustment Through Level Generation. *PCGames '10: Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. (11), 1-4. : <https://dl.acm.org/doi/pdf/10.1145/1814256.1814267?download=true>
- [20] Git: <https://git-scm.com/> Fecha de consulta: 14 de junio de 2020
- [21] Fuente de gráfico Unity: <https://boingboing.net/2018/07/17/the-most-popular-engines-for-i.html> Fecha de consulta: 14 de junio de 2020
- [22] Putting the power of Unity in the hands of every mobile developer: <https://blogs.unity3d.com/2013/05/21/putting-the-power-of-unity-in-the-hands-of-every-mobile-developer/> Fecha de consulta: 14 de junio de 2020
- [23] CryEngine: <https://www.cryengine.com/> Fecha de consulta: 14 de junio de 2020
- [24] Rider: <https://www.jetbrains.com/es-es/rider/> Fecha de consulta: 14 de junio de 2020
- [25] Trello: <https://trello.com/es> Fecha de consulta: 14 de junio de 2020
- [26] Scriptable Objects in Unity: <https://www.youtube.com/watch?v=aPXvoWVabPY> Fecha de consulta: 6 de junio de 2020
- [27] Diablo saga: <https://vandal.elespanol.com/sagas/diablo> Fecha de consulta: 6 de junio de 2020
- [28] PDF asignatura Interfaces Persona Computador: <http://personales.upv.es/moimacar/master/download/interfaces.pdf> Fecha de consulta: 25 de junio de 2020
- [29] Height map, documentación Unity: <https://docs.unity3d.com/es/2018.4/Manual/StandardShaderMaterialParameterHeightMap.html> Fecha de consulta: 13 de junio de 2020
- [30] Hardzone.es: <https://hardzone.es/2018/05/06/motor-grafico-juegos/> Fecha de consulta: 3 de junio de 2020

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

9. Glosario

.gitignore

Fichero que indica a Git que archivos ignorar a la hora de guardar el estado actual del proyecto en el repositorio. Suelen ignorarse contenido como compilaciones, archivos temporales, etc.

Backlog

Tecnicismo. Lista ordenada de trabajo pendiente de un proyecto. Ampliamente empleado en ámbitos de trabajo ágiles.

Commit

Tecnicismo. Confirmar los cambios seleccionados del proyecto de forma permanente creando así un nuevo estado del proyecto.

Dynamic Difficulty Adjustment

Tecnicismo. Del inglés: Ajuste de la dificultad dinámico, es decir, mientras que el programa que lo implementa se está ejecutando.

Framework

Tecnicismo. Framework o marco de trabajo es la estructura o esquema que se aprovecha para organizar y desarrollar un programa o software determinado. Hace la programación más sencilla para el desarrollador ya que automatiza varios procesos que pueden resultar repetitivos. Ejemplos: Visual Studio, NetBeans, Eclipse.

Generación Procedural

En informática, la generación procedural es un método de creación de información automático, normalmente generado por la combinación de algoritmos creados por personas y la aleatoriedad y el poder computacional de un ordenador. Con ella podemos crear para nuestros juegos nuevos objetos, personajes, misiones, mapas, etc.



Heightmap

<<El Height mapping (mapeo de altura) (también conocido como parallax mapping) es un concepto similar al normal mapping, sin embargo, esta técnica es más compleja y por lo tanto también más costosa. Los mapas de altura se usan generalmente junto con los mapas de normales, ya menudo se utilizan para dar una definición adicional a las superficies donde los mapas de textura son responsables de representar grandes golpes y protuberancias>> extraído de la documentación de Unity [29]



Figura 26. Height map del planeta tierra

Juegos Indie

11

Los juegos indie son aquellos creados por individuos o pequeños grupos sin apoyo financiero de grandes distribuidores. Generalmente, estos se centran en la innovación dando lugar así a nuevos subgéneros que posteriormente, en determinadas ocasiones, se popularizan, como es el caso del Rogue-lite, derivado del Rogue-like. Ejemplos de juegos indie son: Minecraft, The Binding of Isaac o Super Meat Boy.

Motor Gráfico

4

Según la web hardzone.es [30]: << Los motores gráficos proveen a los desarrolladores de juegos de un motor de renderizado (que permite a los diseñadores el emparejar sus diseños hechos a mano con esquemas en 3D para crear modelos completos), todas las herramientas necesarias para crear la física del juego y ayudan a crear efectos de iluminación; entre otras cosas>>, es decir, es un framework para el desarrollo de videojuegos.

Quality Assurance (QA)

28

16

Tecnicismo. Del inglés: Aseguramiento de la calidad. Conjunto de actividades para asegurar el cumplimiento de los requisitos de calidad de un producto.

Sprint

Tecnicismo. En la metodología Scrum (una metodología ágil) los proyectos se realizan por bloques temporales fijos (dos semanas, un mes, etc.) llamados Sprints.

Unity Plugin para la generación procedural de mazmorras basada en el comportamiento del jugador.

1

Tablero Kanban

Un tablero Kanban consiste en un conjunto de columnas que reflejan los distintos estados por los que pueden pasar las distintas tareas del proyecto. Estas se representarán en el tablero como etiquetas con la información necesaria para resolverlas y se situarán en la columna correspondiente a su estado actual. Los estados más típicos que pueden aparecer en uno de estos tableros son: To Do (pendiente de realizar), Doing (trabajando en ello) y Done (finalizado).

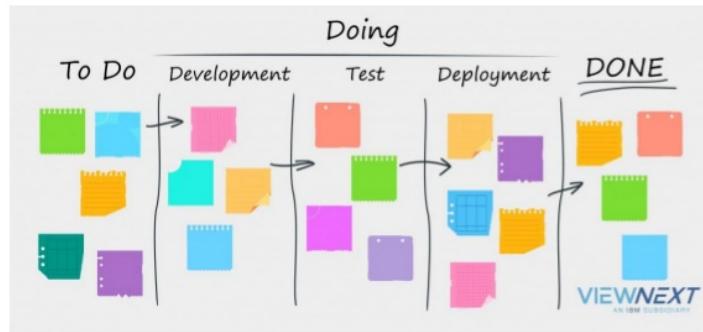


Figura 27. Tablero Kanban

Target

Tecnicismo. Del inglés: público, objetivo (personas) (Se ha utilizado esta palabra en lugar de su traducción al idioma que ocupa este trabajo ya que es un término utilizado en la industria comúnmente).



INFORME DE ORIGINALIDAD



FUENTES PRIMARIAS

- | Rank | Fuente | Porcentaje (%) |
|------|--|----------------|
| 1 | Submitted to Universidad Politecnica Salesiana del Ecuador | 1 % |
| 2 | Submitted to Universitat Politècnica de València | <1 % |
| 3 | graphics.tudelft.nl | <1 % |
| 4 | hardzone.es | <1 % |
| 5 | es.scribd.com | <1 % |
| 6 | sokath.com | <1 % |
| 7 | Submitted to Universidad Estatal a Distancia | <1 % |
| 8 | research-db.ritsumei.ac.jp | <1 % |
- 1 Submitted to Universidad Politecnica Salesiana del Ecuador 1 %
Trabajo del estudiante
- 2 Submitted to Universitat Politècnica de València <1 %
Trabajo del estudiante
- 3 graphics.tudelft.nl <1 %
Fuente de Internet
- 4 hardzone.es <1 %
Fuente de Internet
- 5 es.scribd.com <1 %
Fuente de Internet
- 6 sokath.com <1 %
Fuente de Internet
- 7 Submitted to Universidad Estatal a Distancia <1 %
Trabajo del estudiante
- 8 research-db.ritsumei.ac.jp <1 %
Fuente de Internet

9

Submitted to Universidad Politécnica de Madrid

Trabajo del estudiante

<1 %

10

e-archivo.uc3m.es

Fuente de Internet

<1 %

11

worldwidescience.org

Fuente de Internet

<1 %

12

academico.une.org

Fuente de Internet

<1 %

13

riunet.upv.es

Fuente de Internet

<1 %

14

insis.vse.cz

Fuente de Internet

<1 %

15

lucio.ls.fi.upm.es

Fuente de Internet

<1 %

16

documents.mx

Fuente de Internet

<1 %

17

www.forospyware.com

Fuente de Internet

<1 %

18

www.lamaquila.com

Fuente de Internet

<1 %

19

www.juxtlahuaca.org

Fuente de Internet

<1 %

20

oa.upm.es

Fuente de Internet

<1 %

21 www.narconews.com
Fuente de Internet

<1 %

22 Submitted to Pontifícia Universidad Católica del
Peru
Trabajo del estudiante

<1 %

23 www.uoc.edu
Fuente de Internet

<1 %

24 www.adrianahidalgo.com
Fuente de Internet

<1 %

25 www.captralir.df.gob.mx
Fuente de Internet

<1 %

26 www.softwarelibre.gob.mx
Fuente de Internet

<1 %

27 Submitted to Universidad de Alicante
Trabajo del estudiante

<1 %

28 Submitted to ECCI
Trabajo del estudiante

<1 %

Excluir citas

Activo

Excluir coincidencias

Apagado

Excluir bibliografía

Activo