Generación automática de mapas 2D mediante la adaptación de la dificultad a la habilidad del usuario

Procedural 2D map generation with dynamic difficulty adjustment

Trabajo de Fin de Grado

Curso 2021-2022

Autor

Mario Jiménez Contreras

Director

Prof. Dr. D. Carlos León Aznar

Grado en Desarrollo de Videojuegos

Facultad de Informática

Universidad Complutense de Madrid

Generación automática de mapas 2D mediante la adaptación de la dificultad a la habilidad del usuario

Procedural 2D map generation with dynamic difficulty adjustment

Trabajo de Fin de Grado en Desarrollo de Videojuegos

Departamento de Ingeniería del Software e Inteligencia Artificial

Autor

Mario Jiménez Contreras

Director

Prof. Dr. D. Carlos León Aznar

Convocatoria: febrero 2022

Grado en Desarrollo de Videojuegos

Facultad de Informática

Universidad Complutense de Madrid

Resumen

Hallar el perfecto equilibrio de dificultad para un videojuego es una tarea tediosa que requiere de diseñadores veteranos e incontables iteraciones. Esta tarea puede volverse más sencilla mediante técnicas de inteligencia artificial, utilizando aprendizaje automático. Por ello, este proyecto tiene como objetivo generar mapas con una dificultad adaptada a la habilidad del jugador. Se ha creado un videojuego con vista cenital, de género *Roguelike* (de mazmorras, cuyos niveles son generados procedimentalmente). Cuenta con una mecánica básica de movimiento en ocho direcciones y con disparo en 360 grados. La meta es obtener un número concreto de coleccionables en cada ronda para avanzar a la siguiente, evitando además ser eliminado por los enemigos, que se desplazan hacia el avatar del jugador, restándole vida por contacto.

Durante las sesiones de juego se recogen los valores de las variables que modelan el entorno del juego, como el tamaño de la mazmorra o el número de habitaciones, además de las encargadas de configurar el comportamiento de los enemigos. Esta información se almacena en forma de trazas y se envían a un servidor cada vez que finaliza una ronda, para posteriormente recoger y procesar los datos y con ellos, obtener una fórmula que modela la dificultad de los mapas a generar.

El objetivo de la fórmula es dar con un valor que equilibre la experiencia de usuario entre divertida y desafiante, evitando el aburrimiento y consiguiendo una evolución del dominio del jugador sobre el juego. El resultado obtenido muestra una mejora positiva sin intervención humana del 16% en cuanto a la calidad de la experiencia vivida por los usuarios cuando se adapta la dificultad a la habilidad del jugador.

Palabras clave

Generación procedimental, inteligencia artificial, aprendizaje automático, videojuego, *roguelike*, diseño de videojuegos.

Abstract

Finding the difficulty balance in video games is a tedious task. It requires a group of video game designers working on the mechanics, gameplay, and level design. Designing can be eased with artificial intelligence algorithms, in this case, using machine learning. Hence the creation of a project whose main objective is to generate maps with dynamic difficulty adapted to the ability of each player. The project consists of a video game with top-down view, Roguelike's genre, with levels generated procedurally. Player movement is simple, with eight possible directions and 360 shooting. The goal is to get all the collectibles in the level to get to the next level. Levels also have enemies that run to the player to damage him.

During game sessions, information about the variables that determine level generation is recorded. These variables are dungeon size, number of rooms, collectibles, enemies' behavior, speed, action zone or the amount of damage received. This information will be processed to create a formula that models map's difficulty.

The purpose of the formula is to balance user experience between fun and challenging to avoid boredom and improve the player's ability. The result is a 16% improvement when difficulty adjustment is active.

Keywords

Procedural generation, artificial intelligence, machine learning, video game, roguelike, video game design.

Agradecimientos

A mi director, Carlos León Aznar, por su ayuda desde el primer minuto, por la idea, su guía, por tantas reuniones y por ponerme los pies sobre la Tierra.

A mis padres, por hacer posible que estudiase videojuegos. A mi hermana, por su ánimo y apoyo.

A David, por las cervezas. A Owen, por las películas malísimas. A Lucía, por traducir esta memoria al castellano y después al inglés. Y a Gerard, por las referencias a Los Simpson.

Y agradecer muchísimo al conjunto de profesionales del sector del videojuego por su ayuda en la difusión de este proyecto y por sus buenas palabras.

Índice

Índice de figuras

1. Introducción

No está del todo claro cuándo se desarrolló el primer videojuego debido a la variedad de definiciones que adopta según qué autores, pero podemos afirmar que los videojuegos cuentan con al menos 50 años de antigüedad, por lo que se puede considerar un medio joven, especialmente si lo comparamos con la música o la pintura. Los primeros videojuegos, debido a las limitaciones técnicas de la época, eran muy simples en los aspectos que hoy en día son importantes y le dan valor, tales como los gráficos, la narrativa, la música o el control, pues han ido adquiriendo más complejidad conforme la tecnología avanzaba y se invertía en desarrollo. En los últimos años los videojuegos están logrando captar la atención de una audiencia más variada (Forbes, s.f.), al mismo tiempo que aparecen nuevas propuestas y se innova en la técnica. No todas las mejoras han sido tecnológicas, muchas de las mejoras de las que hoy disfruta el videojuego surgen del perfeccionamiento a la hora de diseñarlos. Tanto sus sistemas como sus mecánicas están más pulidos y generan experiencias más satisfactorias.

El diseño de videojuegos, referido a la creación de niveles y elaboración de mecánicas, era una disciplina que no existía en sus inicios y es uno de los aspectos que también evoluciona con los años, generando nuevas reglas y todo un conjunto de recursos audiovisuales y literarios didácticos, útiles a seguir para la creación de videojuegos.

Para la tarea de diseño es común contratar a un profesional o un grupo de profesionales que se encargue de su ejecución y supervisión, ya que es una tarea complicada que, aunque sigue una serie de fórmulas, no se puede automatizar. En un videojuego, todos sus elementos deben tener cohesión y funcionar correctamente, como la historia, los personajes, la interfaz, las mecánicas y los niveles. El diseñador es el encargado de que conseguir esta cohesión.

Un buen diseño consigue una experiencia sólida, coherente y equilibrada, pero para ello debe contar con una curva de aprendizaje justa. La curva de aprendizaje representa gráficamente el proceso de interiorización de las mecánicas por parte del jugador al mismo tiempo que el juego aumenta el grado de complejidad de la experiencia. Por lo tanto, una curva de aprendizaje justa se encarga de producir un videojuego comprensible en cuanto a sus mecánicas, y exigente en cuanto a su dificultad. Como resultado, se obtiene un jugador que conoce las herramientas a su disposición y consigue tener una experiencia divertida.

El mundo en el que nos movemos maneja cada vez una cantidad de datos mayor. Al mismo tiempo, el mundo de los videojuegos está creciendo con la llegada de más jugadores, se favorece el juego social gracias al establecimiento de Internet y esto genera mucha información útil, en este caso, para los desarrolladores de videojuegos. Esta información puede tratarse mediante técnicas propias de la ciencia de datos, y utilizarse para mejorar un videojuego a través de actualizaciones.

La ciencia de datos surgió en los sesenta y como dice Javier Jiménez “sus profesionales están cada vez más demandados" (Jiménez, Xataka, 2020) en el mundo laboral. Como subdivisión de la Informática, es un campo que requiere de conocimientos de programación y de estadística, basado principalmente en interpretar grandes bases de datos. Para esto se sigue una

metodología de extracción de datos, se modifican mediante operaciones estadísticas que procesan la información relevante, y finalmente estos datos se entrenan para poder predecir un resultado en casos reales.

Los grandes estudios de videojuegos están apostando por esta tecnología para balancear los sistemas del videojuego y para generar contenido. No sólo los juegos multijugador despiertan interés en la recogida de datos, los juegos de un solo jugador o jugador local también generan trazas de información útil para que los desarrolladores puedan mejorar sus videojuegos. Algunas de las empresas que llevan a cabo estas prácticas son Blizzard, con el juego de cartas Hearthstone (Blizzard Entertainment, 2014), que tiene 100 millones de usuarios registrados; o Epic Games, que cuenta con 350 millones de jugadores (Ayora, 2021) en Fortnite (Epic Games, 2017). Cada día salen videojuegos nuevos que entran a competir en el mercado por obtener usuarios. Por esto mantenerse al día y contentar al público es clave para seguir estando en la cima y generar dinero. Esta es la razón principal por la cual las empresas han encontrado interés en el aprendizaje automático.

El aprendizaje automático no solo permite tratar cantidades masivas de datos, también permite generar contenido para los videojuegos, se puede obtener personajes movidos por la máquina con comportamientos más verídicos y complejos, establecer algunos de los parámetros que modelan la experiencia de juego, recoger e interpretar datos sobre lo que gusta o no de los videojuegos para modificar estos, e incluso conocer mejor a los jugadores. Esta tecnología también se está utilizando para conseguir gráficos más realistas, generar raytracing (trazado de rayos) en tiempo real, remasterizar videojuegos, mejorar texturas; generación procedimental de mapas, o incluso generación de armas, tanto su aspecto como sus características, de forma completamente automática.

1.1 Dificultad en videojuegos

Al comenzar un videojuego, uno de los primeros menús que se nos muestra es el menú de selección de dificultad. En este menú se realiza una división de la dificultad que la categoriza en fácil, normal y difícil. Otros videojuegos no tienen esta división ni tampoco la opción de elegir, hay una única dificultad común para todos los jugadores. En cualquiera de las dos situaciones, las reglas del juego se mantendrán en continuo balanceo y en busca de una curva de dificultad justa y satisfactoria. El problema es que en ocasiones esta curva no está bien ajustada. Un mismo videojuego puede ser muy fácil para algunos jugadores, o muy difícil para otros. En cualquiera de los dos casos, el resultado final es la frustración. Para evitar esta frustración, que puede acabar en abandono, algunos juegos hacen uso de la técnica de ajuste de dificultad dinámico.

El ajuste de dificultad de manera dinámica resulta interesante porque está controlando, de forma continua, la habilidad del jugador y adaptándose a esta, de manera que no se produce una experiencia frustrante, ni tampoco aburrida, el nivel de reto siempre es el adecuado para disfrutar del juego.

Pese a que bastantes videojuegos han implementado esta técnica, no es lo más común, requiere un esfuerzo extra para los diseñadores y no todos los estudios están dispuestos a llevarla a cabo, o no encuentran la forma, pues encontrar las fórmulas óptimas de ajuste no es fácil y requiere de mucho testeo. Sin embargo, los complejos algoritmos de aprendizaje automático son capaces de gestionar datos de manera que hace años parecía impensable (Xataka, 2020) y podrían ser capaces de facilitar esta tarea.

1.2 Motivación

Dentro del panorama del videojuego, muchos usuarios sienten curiosidad por probar otros géneros y en ocasiones no pueden superar los retos que algunos videojuegos proponen por no estar acostumbrados, no tener la habilidad suficiente, o simplemente por ser juegos muy exigentes. Esto ocurrió con lanzamientos como Sekiro: shadows die twice (From Software, 2019) o Cuphead (Studio MDHR, 2017), videojuegos con una dificultad tan elevada que remueven el recurrente debate de si todos los videojuegos deberían incluir un modo fácil (Casademont, 2021). Algunos títulos incluyen a posteriori, mediante un parche, una nueva dificultad que permite a los jugadores disfrutar de la historia del videojuego sin preocuparse por su dificultad, como en Celeste (Matt Thorson, 2018), que introdujo el *assist-mode* (modo asistido), un modo en el que el jugador cuenta con poderes ilimitados o SOMA (Frictional games, 2015), un juego de terror que introdujo el *safe-mode* (modo seguro), una modalidad carente de enemigos.

Lanzar modalidades fáciles a su vez reactiva la discusión sobre si facilitar los retos hace que un videojuego pierda su esencia o la intención de su autor. El ajuste de dificultad dinámica podría facilitar o dificultar la experiencia de juego, pero todos los jugadores sentirían el mismo nivel de exigencia. Por este motivo, el ajuste de la dificultad de manera dinámica en función de la habilidad del jugador es una característica que puede acercar los videojuegos a la mayor

cantidad de gente posible. Pero no todo es perfecto con un sistema de dificultad adaptativa (Adams, 2008). Algunos jugadores pueden aprovecharse de las fórmulas de ajuste de dificultad fingiendo que son peores jugando para reducir el nivel de dificultad y pasarse el juego fácilmente, estropeando la experiencia.

El ajuste dinámico puede adaptarse a cualquier género, ser útil, y resultar reconfortante. Y si una IA puede recomendarte productos que no sabías que querías, o un grupo de música que no conocías, podría recomendarte una dificultad especial para ti. Esta situación ha llevado a que el proyecto se centre en la generación de niveles controlada por una inteligencia artificial adecuando su complejidad a las necesidades del jugador, pese a los puntos negativos que puede implicar un sistema de dificultad adaptativa en un videojuego.

Con este panorama, tiene mucho sentido hacer un proyecto que busque automatizar la tarea de adaptación de la dificultad y hacerla única para cada usuario. De este modo, se vuelve posible contentar a un mayor grupo de usuarios y hacer posible que todos disfruten del mismo videojuego, con un grado de exigencia personalizado. Además, automatizar esta tarea de diseño mejoraría los tiempos de coste que supone desarrollar un videojuego, ahorrando así dinero al estudio responsable de crearlo.

1.3 Objetivos

El objetivo del proyecto es la implementación de un sistema de captura de datos y aprendizaje máquina capaz de generar niveles para videojuegos de tipo *Rogue* en función de la habilidad del jugador de manera procedimental.

Para ello se establece un entorno modular y escalable, para ser capaz de obtener múltiples configuraciones.

La inteligencia artificial se encarga de calcular el valor de los parámetros del mapa a generar antes de comenzar la partida. Al terminar la partida, se comprueba si el jugador ha resultado victorioso. Después, se tienen en cuenta variables discretas como el número de disparos que realizó durante la ronda, la precisión del jugador o el tiempo que tardó en completar el nivel para determinar si es necesario llevar a cabo el ajuste de dificultad.

Definir qué aspectos forman parte de la habilidad del jugador o qué elementos de la aplicación tienen impacto sobre el rendimiento del usuario es muy difícil. Por ello, se decide que el sistema de aprendizaje automático reciba la mayor cantidad de información posible sobre la partida para poder decidir qué variables son relevantes a la hora de moldear la generación del mapa y cuáles no lo son.

En particular, se identifican estas tareas como objetivos concretos del proyecto:

- **Generación de un proyecto con jugabilidad suficiente:** se utiliza Unity (Unity Engine, s.f.) como motor de videojuego, para comodidad y rapidez en el desarrollo. Para generar los mapas, se utiliza Dungeon Tools (Itrch, 2015), un asset que genera procedimentalmente escenarios divididos en salas y unidos por pasillos, dando como resultado mapas de tipo mazmorra. Para otorgar al experimento de objetivo y de dificultad, se sitúan en el mapa coleccionables y enemigos que nos impiden alcanzarlos. La partida finaliza cuando el jugador obtiene todos los coleccionables o pierde la vida.
- **Recopilar y enviar información:** la aplicación genera trazas de manera interna donde quedan registrados los datos de entrada, eventos del juego y las valoraciones del cuestionario que se hace al jugador. El archivo de trazas generado es enviado a un servidor que contiene una base de datos en la que se almacenan todas las partidas jugadas de todos los jugadores, cada una con un id único.
- **Procesar la información y entrenar la IA:** un proyecto aparte analiza el archivo con contenido estructurado generado en la anterior etapa y da lugar a unas tablas de información que sirven para entrenar el sistema de aprendizaje automático.
- **Generar mapas ajustados al jugador:** el sistema de aprendizaje automático entrenado es utilizado para generar mapas con parámetros que se ajustan a la habilidad del jugador y le ofrecen una experiencia satisfactoria.

1.4 Metodología

En este capítulo se explica el proceso de desarrollo del proyecto desde la etapa de diseño de la aplicación. Se detallan las tareas de implementación, la captura de datos y el aprendizaje de la IA. Por último, la generación de mapas con dificultad adaptada y las conclusiones extraídas.

- **Determinar el diseño básico de la aplicación.**

El proyecto tiene como propósito desarrollar una inteligencia artificial capaz de ajustar la dificultad de un videojuego de manera dinámica. La primera tarea debe ser decidir qué género es el más adecuado para dicho propósito. Para ello se establecen unos requisitos mínimos como una jugabilidad simple, fácil de identificar y dominar, accesible para un público extenso, con un número reducido de controles y fácilmente escalable en tamaño y complejidad.

Se descarta la idea de crear un proyecto en tres dimensiones por la complejidad que puede suponer jugar y controlar la cámara al mismo tiempo, puesto que produciría sesgo en la muestra de datos. Dar control al usuario sobre la cámara supone arriesgarse a que se pierda información relevante sucediendo en la escena. Esta perspectiva además requiere mayor esfuerzo de cara al desarrollo de la aplicación en cuanto a la programación y la creación de elementos de la escena.

En cuanto a proyectos de dos dimensiones, existen diferentes aproximaciones, como el género de plataformas, videojuegos de desplazamiento lateral como el Super Mario Bros (Miyamoto, 1983); los juegos de mazmorras con vista desde arriba, como el Zelda (Shigeru Miyamoto, 1986), o los juegos de puzles, como Bomberman (Nakamoto, 1983) o juegos de cartas y estrategia. Los juegos de puzles y estrategia se descartan por la variedad de formas en que se pueden abordar a la hora de jugar y su complejidad de desarrollo. El resto de las opciones son fácilmente escalables, tienen una jugabilidad sencilla y controles básicos. La lógica es más simple y los enemigos, de haberlos, no requieren de un comportamiento sofisticado. Además, hay muchos artículos acerca de generación procedimental de mapas en 2D.

Pese a la escalabilidad y los beneficios del género de plataformas, son juegos que requieren de precisión y habilidad sobre el control del personaje. Sin embargo, el movimiento en los juegos de vista desde arriba es constante y sencillo, con un esquema de control simple.

Por estos motivos, tiene sentido elegir el género de mazmorras para la realización del experimento.

- **Idear la mecánica principal del videojuego**

Cada género de videojuego lleva acompañado una serie de mecánicas básicas que le da personalidad y que siempre están presentes en los videojuegos pertenecientes a dicho género. En el género de mazmorras, la mecánica principal es la de exploración, normalmente acompañada de combate. A esta fórmula básica se le puede añadir habitaciones secretas, atajos, un sistema de recompensa o una tienda para añadirle profundidad al diseño.

La exploración puede estar orientada a descubrir el camino desde la entrada hasta la salida para completar el nivel, u obtener ciertos objetivos para poder finalizarlo.

A priori parece buena idea seguir un esquema de jugabilidad como los del género de mazmorras, donde se debe completar una tarea para poder desbloquear el acceso a la siguiente pantalla o nivel.

- **Generación de mapas procedimentalmente**

Existe una gran cantidad de algoritmos para la generación de mapas 2D y es importante elegir cuál produce escenarios con una estructura y apariencia que mejor se adecue al género y a la experiencia que se quiere lograr. Por ejemplo, un mapa generado con autómatas celulares es mucho más orgánico y laberíntico que uno generado mediante árboles que dividen el espacio recursivamente y producen habitaciones unidas por pasillos.

Una vez la generación de mapas está completa, el juego se debe poblar de elementos jugables y darle una funcionalidad y objetivos básicos.

- **Identificar dónde reside la dificultad del nivel y cómo cuantificarla.**

Existen multitud de géneros de videojuegos y cada uno tiene unas mecánicas propias, por lo que la dificultad se mide de forma distinta dependiendo del género al que pertenezca. Por lo tanto, el sistema resultante solo podrá ser válido para un tipo de juego en concreto o juegos con elementos en común.

Para hallar los aspectos sobre los que recae la habilidad del jugador debemos desglosar el diseño del juego hasta hallar los elementos básicos que lo conforman. En el género de mazmorras tendremos en cuenta su tamaño M, el número de habitaciones N, unidas por P pasillos, en las que se encuentran E enemigos y C coleccionables.

Con estos elementos básicos podemos medir el rendimiento del jugador y formarnos una idea de su perfil.

Pese a que todos estos componentes son relevantes a la hora de medir la habilidad del jugador, no todos lo son en igual medida, y aquí es donde entra el sistema de aprendizaje automático. Dado un conjunto de datos, podrá hallar cuáles son más relevantes a la hora de medir la habilidad del jugador y del mismo modo, encontrar qué diferencia a un nivel difícil de uno fácil.

- **Construir un sistema encargado de generar trazas de juego.**

Al utilizar un modelo de aprendizaje automático, es necesario entrenarlo para predecir resultados posteriormente, por lo que es imprescindible almacenar todo lo que pasa durante la partida en un archivo de información organizada para generar un set de datos. En el archivo se recoge información sobre los datos de entrada, los parámetros que generan el entorno y los eventos que tienen lugar durante la partida.

Además, se incluye un cuestionario que aparece al finalizar cada nivel para que el usuario puntúe la diversión y la dificultad de este, así, esta valoración servirá como etiqueta para que la IA evalúe el mapa dado un conjunto de parámetros.

- **Desplegar un servidor online.**

Para facilitar el proceso de recolección de datos y que no se pierdan los de ningún usuario, al terminar cada ronda se suben las trazas a un servidor común.

Este sistema es sencillo tanto para el usuario, que es liberado de la tarea de buscar los ficheros y archivos locales para después enviarlos, como para el autor, que se asegura que los datos de todos los usuarios están en un único archivo.

- **Recoger y procesar datos.**

Una vez recogidos los datos del servidor, los procesamos con un software estadístico para generar tablas que permitan la correcta visualización de los datos y generar gráficas de correlación, que muestran las relaciones entre las diferentes columnas que conforman las tablas.

Con los datos analizados, podemos entrenar el algoritmo de inteligencia artificial.

- **Entrenar el sistema de Aprendizaje automático y generar mapas con dificultad adaptada.**

El entrenamiento de datos da como resultado un modelo entrenado que es capaz de generar niveles que cumplan ciertas condiciones.

El modelo será entrenado para, dado un conjunto de parámetros que generan un nivel, predecir la puntuación de dificultad que le dará el jugador. De esta forma, si al modelo se le facilita una dificultad deseada, podrá proporcionar el conjunto de parámetros encargados de generar un nivel.

- **Analizar la efectividad del modelo.**

En el experimento final se empleará una encuesta de satisfacción acerca del rendimiento de la inteligencia artificial para evaluar su éxito.

Para poder evaluar el rendimiento del sistema, se cuenta con un grupo de usuarios de control, es decir, usuarios cuya aplicación no utiliza IA para generar los mapas, y así contrastar sus respuestas a la encuesta con las del resto de usuarios.

1.5 Planificación/estimación

Tareas	**Meses**									
	1	2	3	4	5	6	7	8	9	10
Determinar diseño básico de la aplicación										
Idear la mecánica principal del juego										
Generación de mapas procedimentalmente										
Identificar dónde reside la dificultad y cómo cuantificarla										
Construir un sistema encargado de generar trazas de juego.										
Desplegar un servidor online										
Recoger y procesar datos										
Entrenar el sistema de Aprendizaje automático y generar mapas con dificultad adaptada.										
Analizar la efectividad del modelo										

Durante los dos primeros meses se lleva a cabo la elección del tema para el trabajo, en este periodo se comienza la estructura de la memoria y se recoge información sobre el estado de la cuestión: juegos con dificultad dinámica y documentos con información sobre generación procedural

En el segundo mes también comienza la implementación del juego, se busca crear un prototipo jugable lo más sencillo posible. Esta tarea a su vez se divide subtareas más simples, como pueden ser el movimiento del jugador, el ataque, seguimiento de la cámara y el comportamiento de los enemigos.

Al mes siguiente hay que lograr generar mapas de forma procedimental. En el trabajo previo se recogen muchos documentos con información sobre generación mediante árboles, agentes y autómatas celulares.

Una vez contamos con las mecánicas básicas y la generación de mapas, hay que posibilitar que los parámetros del juego se modelen en función de la habilidad del jugador. Para esto, se necesita establecer un sistema capaz de cuantificar la habilidad del usuario analizando los datos obtenidos durante las partidas.

Es en el mes cuarto del proyecto cuando se implementa un sistema para generar trazas durante la partida y recoger información de los usuarios. Se añadirá además un cuestionario, pero ha de ser lo suficientemente sencillo como para no interrumpir la experiencia de juego.

Pasado un mes se espera desplegar el servidor encargado de almacenar las trazas generadas. Se lanza el proyecto y se intenta llegar al máximo número de jugadores para recopilar datos.

Se procesan los datos y se entrena el modelo de aprendizaje automático, para poder lanzar una versión con dificultad adaptativa. Se evalúa el rendimiento de la IA y se modifican parámetros del proyecto existente para intentar mejorar los resultados obtenidos.

Se vuelve a recoger datos, entrenar la IA siguiendo los pasos de la primera vez y analiza el rendimiento final. Las conclusiones obtenidas se plasmarán en la memoria tras esta última iteración.

1.6 Estructura del documento

La estructura del documento es la que sigue:

- El capítulo 2 expone el estado del arte en el campo de la generación procedimental en videojuegos, la definición de dificultad, características y posibilidades del aprendizaje automático, las herramientas utilizadas y también videojuegos comerciales que cuentan con dificultad dinámica adaptativa.
- En el capítulo 3 se describe el diseño de la aplicación y el análisis de los datos obtenidos. Además, se detalla el diseño de las diferentes versiones por las que pasa la aplicación.
- El capítulo 4 sirve para exponer los resultados obtenidos tras la fase de experimentación y medimos el éxito del sistema de inteligencia artificial.
- El capítulo 5 detalla la discusión, comparando puntos de unión y conflicto respecto al estado del arte y también se indican posibles mejoras.
- En el capítulo 6 se explican las conclusiones a las que se llega gracias al proyecto, y se razonan los límites y posibilidades de este tipo de sistemas. Quedan reflejados el trabajo futuro, así como las posibles modificaciones.
- Para terminar, en el capítulo 7 se recoge la bibliografía.

1. Introduction

It is not clear which was the first video game in history because of the variety of definitions this concept has depending on the author that refers to it. However, we can state that video games must be 50 years old at least, that is why they are considered a 'young art', especially when compared to paintings and music. The first video games were simple due to the technical limitations of that time. They were extremely simple in aspects nowadays considered quite important such as the graphics, the music quality, and the narrative, or rather, lack of it. Nevertheless, they got better as the technology evolved and improved and its funding was raised. During these past years, they have managed to get the attention of a wider, more varied audience (Forbes, s.f.) thanks to the new approaches, techniques and themes designers work with. However, not every upgrade has been technological. Most of the upgrades video games show nowadays surge from the perfecting of the design. Systems as well as mechanics are more polished and generate more satisfying experiences.

Video game design, especially the creation of levels and the mechanic design, did not exist when the first video games were created. Nevertheless, this aspect has been evolving since, creating new sets of rules and didactic resources used in video game creation.

It is a usual practice to hire a professional or team of professionals to manage and execute the designing aspect of a video game. This is so because the designing of a video game is a complicated task that cannot be automated despite following a series of formulas. In a video game, all the elements (story, characters, interface, mechanics, and levels) must be cohesive and work together. The designer is the person responsible for this cohesion.

A good video game design must provide a solid, coherent, and equilibrated experience. To achieve that, it is imperative to have a fair learning curve. This curve represents the learning process of the player with graphics and increases the difficulty level of the experience. A fair learning curve can produce a comprehensible and challenging video game. As a result, the player will know every tool at his disposal and have a fun experience.

The amount of data the world has access to these days is continuously growing. At the same time, the video games area is also expanding thanks to the ever-growing number of players involved. Social gaming has been favored by the Internet, generating a lot of useful data for the video game developers. This data can be treated with data-science techniques and used to improve a video game through upgrades.

Data science appeared in the 70s. Nowadays, as Javier Jiménez states: "their professionals are becoming more demanded" (Jiménez, Xataka, 2020). It is a field which, as a subdivision of Computer science, calls for programming and statistics knowledge for data treatment purposes. The methodology is as follows: data extraction, statistical calculations over data, and finally, data training to predict real cases.

The big video game studios are investing in this technology to balance the video game systems and generate content. Data collecting is not only focused on multiplayer video games, video games for one player or a local player also generate useful traces of data for their developers

to improve them. This practice is used by big companies such as Blizzard with his card game Hearthstone (Blizzard Entertainment, 2014), which has 100 million users registered; or Epic Games, which has 350 million players (Ayora, 2021) registered in Fortnite (Epic Games, 2017). Everyday new video games enter the market fighting for users, that is why keeping up with the trends and making users happy is key to stay on top and make money. This is the main reason why companies are interested in automatic learning.

Machine learning is used to manipulate large amounts of data, but it can also be used to generate content for video games. Machine learning can produce complex and realistic behaviors for characters, it configures the generation parameters and is able to interpret the players' preferences and get to know them better. This technology is useful for graphics generation too, for example, ray tracing in real time, remastered textures, or procedural map generation.

1.1 Difficulty in video games

When starting a video game, one of the first menus a player encounters is the difficulty menu. This menu divides the difficulty into three levels: easy, regular, and hard. Some other games have no difficulty division so the player cannot choose. There is just one difficulty level designed for all the players. Whichever the situation is, the game rules will stay balanced and trying to generate a fair and satisfying difficulty curve. The issue comes when that curve is not properly adjusted: the same video game may be very easy for some players and super hard for others. Whatever the situation is, it results in a frustrated player that may abandon the game. To avoid this problem, some video games use the dynamic difficulty adjustment technique.

The dynamic difficulty adjustment technique is interesting because it continuously monitors the player's ability, adapting the game rules to it. This way, boredom and frustration are avoided, and the challenge is always adequate.

A few games include this feature, but it is not a common choice because it needs time, effort, and testing sessions, or because the studio cannot find a way to implement it. However, machine learning algorithms are powerful and capable of big data processing (Xataka, 2020), so they can help with this tedious task.

1.2 Motivation

Some players want to try different games and genres, but they find them difficult and whenever they try to play them, and they usually fail. Good examples of difficult games are Sekiro: shadows die twice (From Software, 2019) and Cuphead (Studio MDHR, 2017). They are so demanding that some players find them impossible to finish, but it's frustrating because these games are beautiful and interesting. This event makes some people think: 'should every video game include an easy mode?' (Casademont, 2021). Some of these games update their rules sometime later to reduce their difficulty or to add a new difficulty level. For example, Celeste (Matt Thorson, 2018) with his 'assist-mode', which is a mode with unlimited powers, or SOMA (Frictional games, 2015) introducing the 'safe-mode', without enemies.

Launching easy modes revives the discussion on whether easing the challenges makes a video game lose its essence or the author's intention. The dynamic difficulty adjustment may facilitate or difficult the game experience, but every player would experience the same level of demand. For this reason, the dynamic difficulty adjustment based on the player's abilities is a characteristic that may make video games closer to the biggest amount of people possible. However, the adaptive difficulty system doesn't solve everything (Adams, 2008). Some players may take advantage of the difficulty adjustment formulas pretending they are worse than they really are to reduce the difficulty level and easily complete the video game, thus spoiling the experience.

The dynamic adjustment can adapt to any genre, be useful and comforting. If an AI can recommend products or a music band you do not know based on your preferences and searches, it can recommend a personalized difficulty based on your gaming abilities. This situation has led to the main focus of this project being the generation of levels controlled by an AI that bases the complexity on the player's needs, despite the issues that an adaptive difficulty system may carry for a videogame.

In this setting, it makes sense to try and automate the difficulty adaptation task and make it unique for each user. This way, it is possible to make the biggest number of happy users achievable and make everyone enjoy the same game, albeit with a personalized demand level. Also, automating this design task would make the development of a video game more cost-effective, thus saving money for the studio that creates it.

1.3 Objectives

This project's main goal is to implement a system capable of data capture and treatment to predict players behaviors and generate roguelike levels procedurally, attending their structure and principal elements.

The resulting environment is modular and scalable, compatible with multiple configurations.

Before a level starts, the AI generates a complete set of parameters to it. At the end of the round, player's performance is evaluated, starting with his result (victory or defeat) and then analyzing his precision, number of shoots or time to complete the round.

It is difficult to establish which are the parameters responsible for difficulty and fun so, the procedure is to pass all the parameters to the system and let it decide which are correlated with the player's experience.

The main objectives are the following:

- **Generation of the environment:** the project uses Unity (Unity Engine, s.f.) as its video game engine to save time and ease the work. It also uses Dungeon Tools (Itrch, 2015), an asset that generates 2D rogue maps procedurally. This asset allows editing the number of rooms, their size, and the number of corridors. The resulting map is represented as a grid. The game must have enough complexity and difficulty to produce a rich experience. It must have interesting maps, enemies, and rewards.
- **Data collection:** the software generates play traces every time a user plays a round. These traces contain info about input, game events, physics, dungeon size and player performance. The information is organized by categories and sent to an online server following the REST model where each trace has a unique id.
- **Data treatment and AI training:** another project oversees the conversion of information into tables and then, this information will train a machine learning model.
- **Generate maps adapted to the player:** the automatic trained model will is used to generate maps with parameters that fit the player's ability and therefore provides a satisfactory experience for the player.

1.4 Methodology

This chapter exposes an estimate of the development process, starting with the designing phase. This phase consists of deciding the video game genre, control assignment, perspective and aesthetics. Then, an approximation of the implementation tasks is made.

- **Establishing video game genre**

This project's purpose is to develop an AI capable of adapting video games' difficulty to player's ability. First, it is important to decide a video game's genre. For this purpose, minimum requirements are established, such as a simple gameplay, easy to dominate and accessible to a wide audience. The project must be scalable and flexible too.

The idea of a 3D project is discarded. These projects' controls are too complex because of camera positioning and their development is more expensive than 2D projects.

2D projects main genres are platforms (like Super Mario Bros (Miyamoto, 1983)), dungeons, (like The legend of Zelda (Shigeru Miyamoto, 1986)), puzzle (Bomberman (Nakamoto, 1983)) and strategy games. Puzzle and strategy games are too complicated to escalate them. Platforms and dungeons are good options, but the first one has complicated motion controls, like jumping or lateral movement. Dungeons' controls are easier than platforms and there are a lot of documents about dungeon maps generation.

2D platform games are scalable but difficult to play because of its precision requirement. However, top-down games have constant and simple movement, easy to control.

Because of this, it makes sense to choose the dungeon genre.

- **Deciding the video game's main mechanic**

Each video game genre is associated with a specific mechanic. Dungeons genre's main mechanic is exploration, sometimes accompanied with combat mechanics. This formula can be expanded with secret rooms, shortcuts, or a reward system.

Exploration is normally oriented to find the exit of a level or to collect some objects to complete a task. This mechanic seems adequate to measure players' performances.

- **Generating procedural maps**

There are plenty of 2D map generation algorithms and each one generates a completely different structure, that is why it is important to choose the correct one wisely. For example, a map generated with a cellular automaton is more organic and twisting than a map generated with quadtrees, that is divided in rooms and corridors.

Once we have decided our map generation algorithm, we must implement a method to put elements in it.

- **Identifying what affects difficulty and how to quantify it**

Difficulty resides in different elements depending on the video game's genre. With this established, the resulting project will be useful only for dungeons genre.

To identify the aspects that depend on the player's ability, we must dissect the video game into its basic elements. We have a dungeon with S size, N number of rooms, C corridors and E enemies. These elements are characteristic of a genre, and if we add common metrics like overcoming time, we can measure the player's performance.

All these elements are important but do not share the same degree of importance. To determine their relevance we use machine learning, which can evaluate errors and discarding useless variables.

- **Building a tracing system**

When using a model with automatic learning, it is necessary to train it in order to make it able to predict results. That is why it is essential to track everything that goes on during each game session in an information file to generate a data set. The traces gather information about input, collisions, game events or generation parameters.

Additionally, there will be a little quiz between rounds about the difficulty and fun of the session. The quiz results will be used as a label for the AI to evaluate maps.

- **Online server deployment**

Every time a session finishes, traces are delivered to a server to make the data collection easier. This way is more comfortable for users because they don't have to worry about information files, they only have to play. Also, it is useful for creators because all the information is stored in the same place.

- **Data processing**

Once we have the data, we use a statistic software to process it and remove useless information. This new data serves to create plots and graphs, and to evaluate the correlation between variables. This data will be used to train the machine learning model.

- **Training the Machine learning model and generating maps adapted to user's ability**

The model resulting from data training is capable of generating levels of difficulty given the generation parameters of the map. This way, if the model is given a desired difficulty, it can create the set of parameters needed to generate the level.

- **Analyzing project's success**

The final application will have a poll about the project's performance.

To be able to measure the performance, users will be divided in two groups: the control and the experimental group. The first group won't have dynamic difficulty adjustment, but the second one will. This way, we can compare their ratings.

1.5 Planning/estimation

Tasks	**Months**									
	1	2	3	4	5	6	7	8	9	10
Establishing video game genre										
Deciding the video game's main mechanic										
Generating procedural maps										
Identifying what affects difficulty and how to quantify it										
Building a tracing system										
Online server deployment										
Data processing										
Training the Machine learning model and generating maps adapted to user's ability										
Analyzing project's success										

The decision of the project's topic was made during the first two months. During that period the document structure and information search about procedural generation and dynamic difficulty adjustment started.

Video game's design started the second month. The objective was to create a simple demo that is always playable. To develop the demo, this task was divided into subtasks, such as player movement, camera following or enemies' behavior.

For the next month the project needed to have procedural map generation. There are plenty of information about algorithms of map generation such as quadtrees, agents or cellular automaton in the previous research work.

Once basic mechanics and map generation were implemented, we needed to create a system that generates maps adjusted to player's ability. To do this, we needed information about game sessions.

It is in the fourth month of the project when it is implemented a data collection system that collects information about players and the map generation. Every level of the game had a mini quiz asking about level's experience also.

After a month it is expected to be able to send the data to a server. The game is released then, and we can collect data.

Once we have enough data, we can process it and train the AI to launch a new version with dynamic difficulty adjustment. Data training makes predicting a difficulty given a dataset possible.

Next month we evaluate the AI performance and try to improve it following the previous methodology.

1.6 Document structure

Document structure is the following:

- Chapter 2 explains the state of the art in procedural generation applied to video games, the definition of difficulty and the possibilities of machine learning. It includes the tools I used for the implementation and commercial video games that use dynamic difficulty adjustment.
- In chapter 3 we find the design of the application, its different stages and an analysis of the data acquired in each testing phase.
- Chapter 4 explains the results obtained and the measurement of success.
- Chapter 5 details the discussion, comparing the present project with the state of the art.
- Chapter 6 develops the conclusions, possibilities, and mistakes. It also presents the future work and possible modifications.
- Chapter 7 collects bibliography.

2. Estudio del trabajo previo

Este capítulo comienza con un breve resumen del estado actual de los videojuegos, tendencias y logros alcanzados en los últimos años. Se detalla el estado de la generación procedimental, su origen, metodologías y sus aplicaciones. Después, se recogen documentos relativos a la adaptación de la dificultad al usuario, se habla del aprendizaje automático como rama de la inteligencia artificial y su empleo en el mundo de los videojuegos. También se realiza un estudio sobre la dificultad en los videojuegos y la curva de aprendizaje, se enumeran una serie de títulos comerciales que cuentan con dificultad adaptativa y, por último, se presentan las tecnologías utilizadas durante el desarrollo.

Los videojuegos cuentan con una trayectoria mucho más corta que disciplinas como el cine o la música, por eso no es de extrañar que hoy en día esté buscando su identidad y experimentando con sus posibilidades. Pese a que cada vez encontramos un mayor número de recursos sobre diseño y creación de videojuegos, no hay nada escrito sobre qué se puede o no hacer, no existe un dogma de los videojuegos.

A menudo se confunde generación procedimental con generación aleatoria, y la realidad es que es un sistema que genera contenido siguiendo unas reglas o procedimientos, mientras que los algoritmos de generación aleatoria pueden producir contenido total o parcialmente aleatorio. Si esto fuera así, se generaría contenido incompleto, imposible, o injusto. La generación procedimental se incluye en un videojuego con la intención de proponer un reto diferente y único al jugador cada vez que intenta superar un nivel, ofrecer una experiencia más duradera o aumentar el número de veces que el usuario volverá a jugar. Uno de los primeros videojuegos que cuentan con contenido generado procedimentalmente es Rogue (Michael Toy, 1980). Creado en 1980, fue originario del género que lleva su nombre, *roguelike*, esto es, juegos con mapas de tipo mazmorra, divididos en habitaciones y pasillos, y generados procedimentalmente.

La dificultad adaptativa es otra decisión de diseño. Si bien el tipo de dificultad que tenga un videojuego puede acercar o alejar al público debido a su grado de exigencia, la dificultad adaptativa busca ajustarse al jugador y ofrecerle un reto apropiado para sus habilidades, y así hacer llegar un juego al mayor número de personas.

2.1 Generación procedimental

Al contrario de lo que pueda parecer, la generación procedimental no recurre a la aleatoriedad, sino a reglas y procedimientos, como su nombre indica. Sus aplicaciones abarcan campos como la música (Adam, 2014), creación de personajes (Short, 2016) e incluso narrativa (Gervas, 2009). Sus inicios en el mundo de los videojuegos tienen lugar a principios de 1980, con el juego Rogue (Michael Toy, 1980), del género de mazmorras. Más adelante, en 1984, aparecen el Tetris (Pázhitnov, 1984) y Elite (David Braven, 1984) como otros ejemplos de juegos procedimentales. Años más tarde, uno de los grandes éxitos del género de mazmorras con contenido procedimental fue Diablo (Blizzard Entertaiment, 1996), que consigue popularizar esta técnica para la generación de contenido. En la actualidad, es una característica que incluyen

multitud de juegos, como Returnal (Housemarque, 2021), Binding of Isaac (Edmund McMillen, 2011), No man's sky (Hello games, 2016), Dead Cells (Motion Twin, 2017) o Townscaper (Stålberg, 2020).

Existen diferentes algoritmos para la generación de niveles en videojuegos y el algoritmo utilizado depende del género del juego a desarrollar y de la decisión del diseñador. Hoy en día podemos encontrar generación procedimental en juegos de estrategia, de rol por turnos o *shooters*. En el presente documento nos centraremos en la generación procedimental de mapas en los géneros roguelike y plataformas con Spelunky y Super Mario Bros como ejemplos.

2.1.1 Métodos para la generación procedimental de mapas 2D

En (SHAKER, 2016) se exponen los diferentes algoritmos utilizados para la creación de mazmorras, basados en la división recursiva del espacio siguiendo unas reglas, que da como resultado una jerarquía de celdas o habitaciones en forma de árbol, pero también se muestran algoritmos de generación de mapas mediante agentes y a través de autómatas celulares.

2.1.1.1 BSPs (Binary Space Partitioning)

Este algoritmo comienza con un área que va subdividiendo recursivamente siguiendo unas reglas de tamaño mínimo por celda, dando lugar a dos celdas o áreas hijas, hasta alcanzar el número de divisiones deseado o no poder subdividirse más. Gracias a este tipo de división, las celdas pueden representarse como un árbol binario o árbol BSP. Este algoritmo además garantiza el no-solapamiento entre habitaciones.

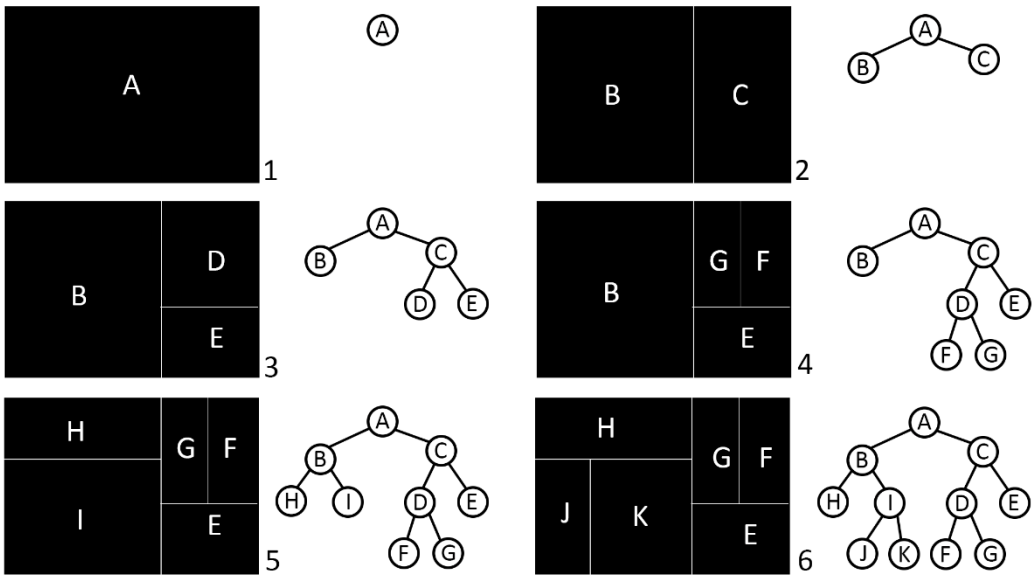

Ilustración 2.1. Árbol binario o árbol BSP. Mediante la repetida subdivisión de un espacio en dos, se genera un mapa y un árbol binario, que sirve como visualización de esta división y posteriormente es útil para incluir pasillos que conecten las habitaciones o ramas.

A partir de esta división, se localizan las habitaciones dentro de los límites de las celdas y a continuación, se unen las habitaciones mediante pasillos, comenzando desde las últimas ramas generadas y subiendo por el árbol hasta llegar a la raíz.

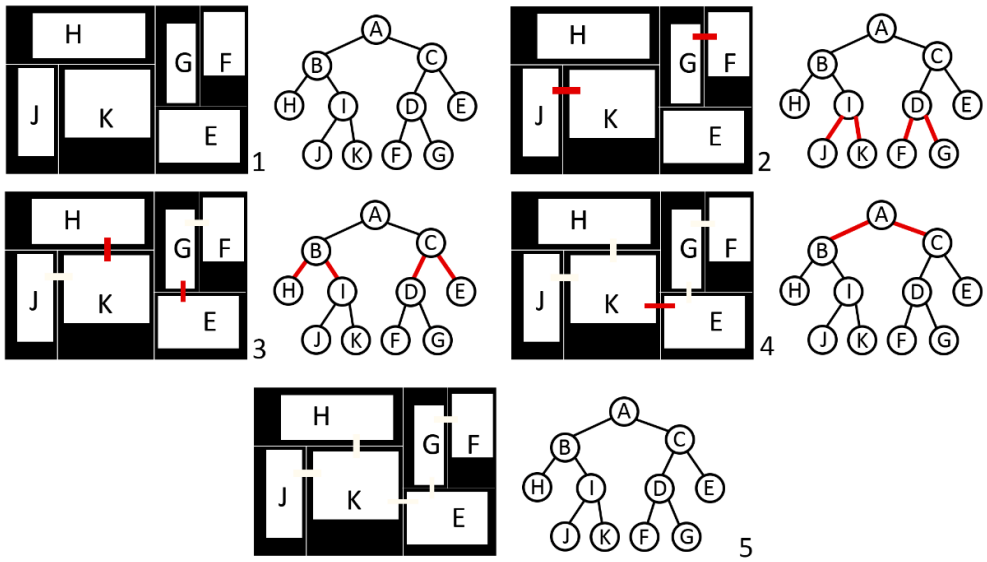

Ilustración 2.2. Ramas del árbol BSP como pasillos de la mazmorra. Las ramas que conectan un nodo padre con sus hijos resultan útiles para generar pasillos que unen habitaciones, para que el mapa resultante sea conexo.

2.1.1.2 Quadtree

Este algoritmo es utilizado frecuentemente en aplicaciones de gráficos para organizar la información de una imagen y manipularla de forma más eficiente. Para la generación de mapas funciona muy parecido a los árboles BSP, solo que, en lugar de generar dos hijos en cada subdivisión, genera cuatro.

Una vez dividido el espacio, la manera en que lo aprovechemos para generar las habitaciones y sus pasillos dará lugar a dos posibles tipos de mapa (Ilustración 2.3). Si tenemos en cuenta cada celda como un espacio en blanco o a rellenar (figura a) obtenemos un mapa robusto. Por otro lado, si generamos una habitación por cada celda (figura b) nos da como resultado una mazmorra mucho más "cuadriculada" y laberíntica.

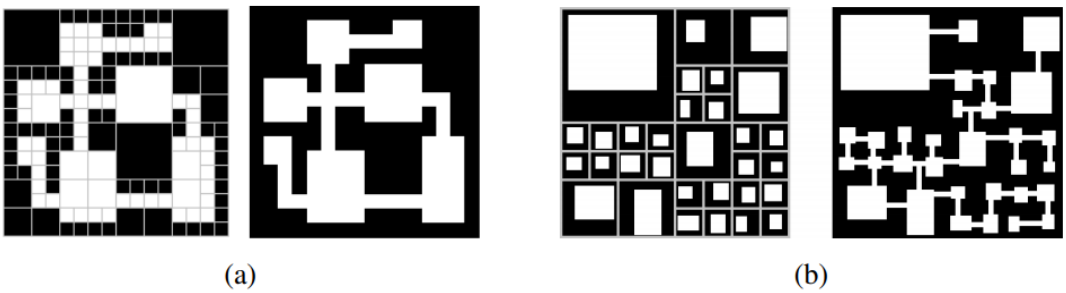

Ilustración 2.3. Generación de una mazmorra mediante quadtrees. Interpretando cada celda como un espacio en blanco o con suelo (figura a). Interpretando cada celda como una habitación cuadrada (figura b). Imagen extraída de (SHAKER, 2016)

2.1.1.3 Generación de mazmorras basada en Agentes.

Se utiliza un agente que "vaga" y al mismo tiempo está cavando el mapeado. Como resultado, aparece una mazmorra mucho más orgánica, pero también caótica. El aspecto dependerá en gran medida de prueba y error y del tipo de agente que la crea, resultando caótico en el caso de un agente de comportamiento estocástico. También podría dar como resultado mazmorras caóticas sin pasillos o sin habitaciones regularizadas. Existen infinidad de comportamientos para agentes, pero el más común es el de evaluar si en cada "palada" existe una probabilidad de cambiar de dirección, o la probabilidad de generar una habitación de tamaño aleatorio. La probabilidad de cambiar de dirección aumenta con cada palada que ejerce el agente, mientras que la de generar una habitación disminuye. Cuando cambia de dirección, ambas se reinician. (Ilustración 2.4)

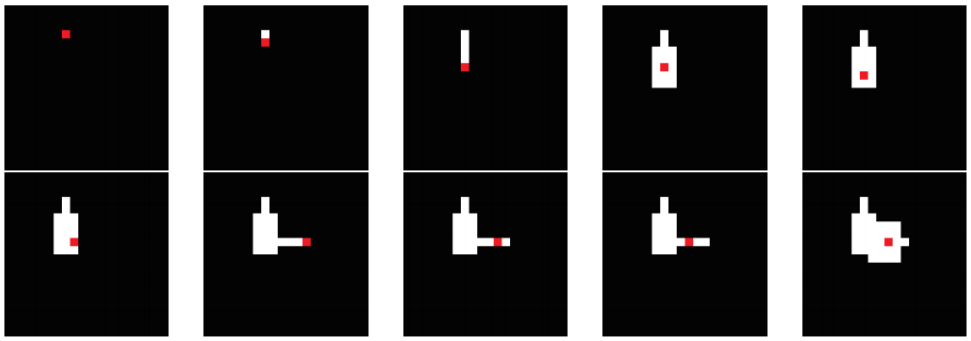

Ilustración 2.4. Generación de una mazmorra mediante un agente. El agente, en rojo, se desplaza en una dirección aleatoria y genera una habitación. No tiene información sobre el entorno, por lo que genera habitaciones solapadas. Imagen extraída de (SHAKER, 2016)

Este método no garantiza generar habitaciones que no se solapen entre sí (Ilustración 2.4) o una mazmorra homogénea, puede incluso que las habitaciones se concentren en una esquina del área del mapa. Para evitar el solapamiento se pueden utilizar agentes con más información sobre el tipo de mazmorra que se quiere generar y dónde están las habitaciones generadas. De este modo, cuando vaya a generar una habitación puede evaluar si resultará en una estructura habitación-habitación o habitación-pasillo (Ilustración 2.5)

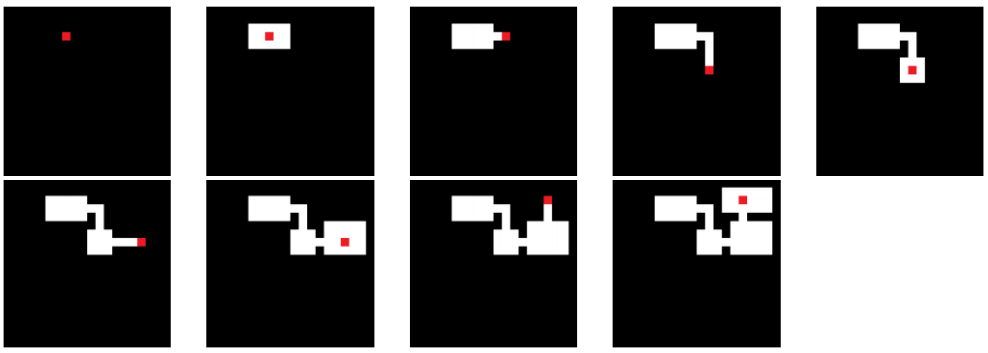

Ilustración 2.5. Generación mediante agente con información del entorno. El agente, comenzando en un punto aleatorio de la mazmorra, genera una habitación siempre y cuando no interseque con ninguna otra anteriormente generada. Imagen extraída de (SHAKER, 2016)

Si todas las posibles habitaciones resultan en intersección, se elige una dirección y una distancia a la que desplazarse que no cause intersecciones con pasillos ni habitaciones para generar otra

habitación. El algoritmo se detiene cuando ya no puede seguir generando habitaciones ni pasillos.

2.1.1.4 Autómata celular

Modelo discreto computacional estudiado ampliamente por la informática, la física y algunas ramas de la biología como modelos de computación, crecimiento y desarrollo.

Un autómata celular consiste en una rejilla de dimensión n, una serie de estados y unas reglas de transición. El conjunto de células comienza en un estado, y en cada paso de simulación deciden si deben mutar en función de las células vecinas. Si el autómata es de una dimensión, cada célula tendrá un máximo de 2 vecinos. Atendiendo a autómatas de dos dimensiones, existen dos modelos posibles de vecinos: Moore y von Neumann. En el primero, cada célula tiene 8 vecinos, en el de Neumann, 4.

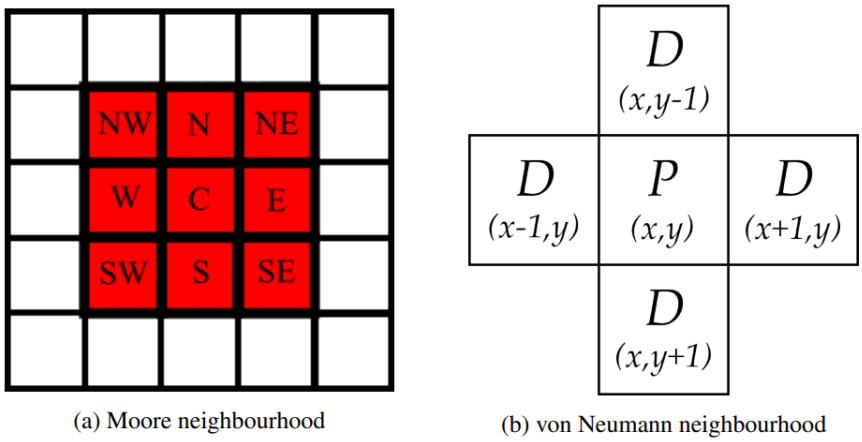

Ilustración 2.6. Tipos de vecinos en una rejilla 2D de células autómatas. Vecindario Moore (Figura a), vecindario Von Neumann (Figura b) Imagen extraída de (SHAKER, 2016)

Cada habitación es una rejilla que, por lo general, se inicia vacía. El primer paso es generar rocas en la rejilla con una probabilidad r, de manera que resulta una distribución uniforme. A continuación, se aplican las reglas de un autómata celular durante un número n de veces, este autómata tiene la única regla de que una célula se convertirá en roca si tiene T vecinos que sean roca.

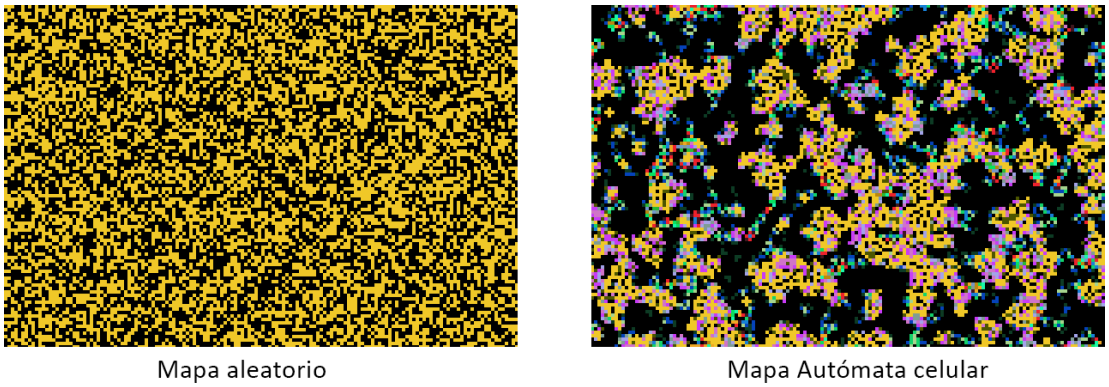

Ilustración 2.7. Mapa antes y después de la interacción de un autómata celular. Imagen extraída de (Hu, s.f.)

2.1.2 Videojuegos y casos de estudio que hacen uso de la generación procedimental

Tanto en los videojuegos comerciales como en el ámbito académico, la generación procedimental ha sido muy utilizada y estudiada para generar experiencias únicas y que favoreciesen completar de nuevo un videojuego.

2.1.2.1 Rogue

Rogue (Glenn Wichman, 1980) es un juego de PC creado en 1980 por Michael Toy y Glenn Wichman, es del género de Mazmorras (Dungeon) y el origen del género *roguelike,* en honor a este. Los juegos pertenecientes a este género se caracterizan por generar sus mapas procedimentalmente, es decir, mediante una serie de reglas se obtienen niveles que cumplen ciertos requisitos y la estructura generada es única.

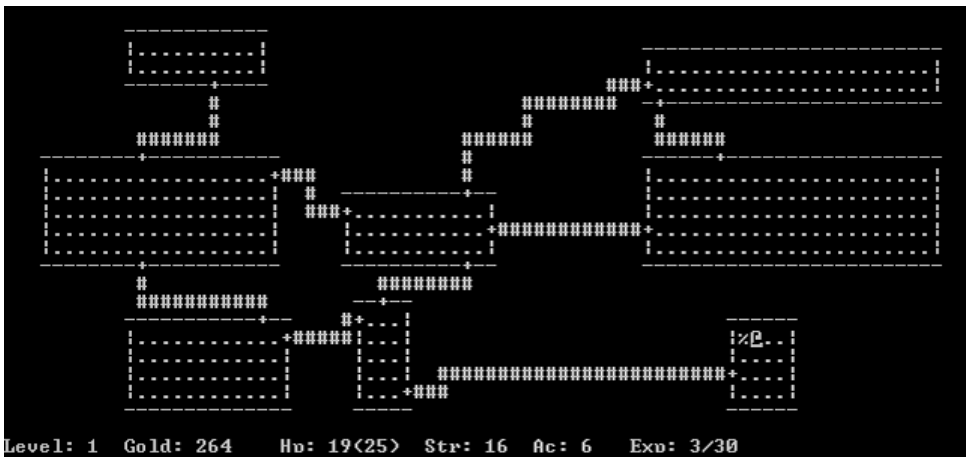

Ilustración 2.8. Captura de Rogue (Glenn Wichman, 1980)

2.1.2.2 Mario Bros

Videojuego creado en 1983 por Shigeru Miyamoto. Este conocido plataformas tiene concursos de algoritmia a nivel mundial para replicar el comportamiento del jugador y superar los niveles autónomamente, pero también hay un largo recorrido acerca de algoritmos capaces de generar niveles para este videojuego.

Aunque el juego original no cuenta con generación procedimental, en (Steve Dahlskog J. T., Patterns and Procedural Content Generation, 2012) se realiza una primera aproximación sobre la generación de niveles mediante patrones de diseño, es decir, identificando elementos comunes en el diseño de la estructura de los niveles o de sus mecánicas. La intención es la de dar con las reglas que definen cómo se construyen los niveles concatenando patrones de diseño, en este caso, mediante una IA. Se identificaron 23 patrones distintos, excluyendo los niveles acuáticos.

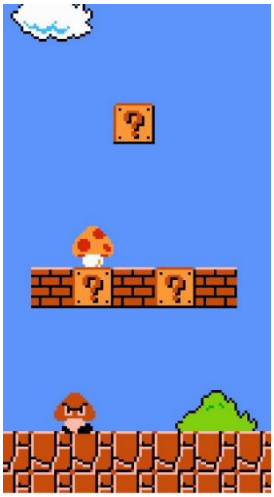

Ilustración 2.9. Patrón de diseño de 3 caminos y riesgo-recompensa (Steve Dahlskog J. T., 2012).

Un año después, en (Steve Dahlskog J. T., Patterns as Objectives for Level Generation, 2013) se continuó esta investigación siguiendo una metodología consistente en dividir los niveles en 200 porciones verticales y examinando cómo esas porciones se interconectaban y qué patrones existían en esas porciones.

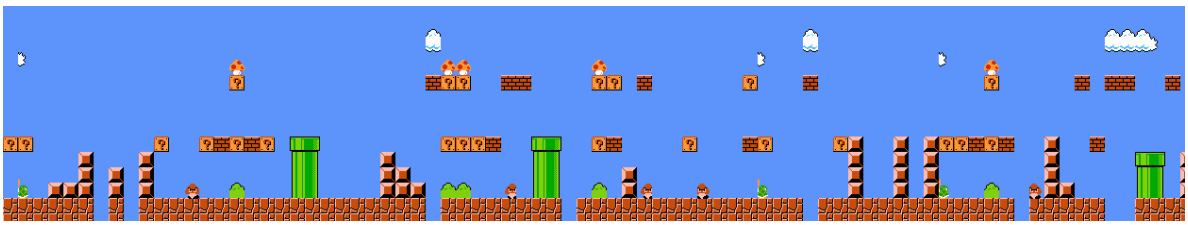

Ilustración 2.10. Nivel de Mario generado mediante porciones verticales (Steve Dahlskog J. T., 2013).

Se generaban niveles de manera exitosa pero el ritmo resultante no se percibía como natural al haber concatenado tantos diseños de manera continua, sin zonas de tranquilidad (Ilustración 2.10).

En 2014 esto derivó en dos proyectos diferentes: uno que revisaba la evaluación de patrones realizada hasta entonces y que era un generador multinivel (Steve Dahlskog J. T., A Multi-level Level Generator) que construía niveles variando las capas de abstracción, desde el diseño de niveles macro, hasta concretar por zonas micro.

Y otro proyecto, (Steve Dahlskog J. T., 2014) en el que utilizaban aprendizaje automático y subsecuencias de n-gramas de porciones verticales para construir un modelo Markov en el proceso de generar niveles. Este modelo se basa en que el estado futuro depende del actual, y no de los pasados. Usando una colección de los patrones más utilizados en los niveles de Mario, el sistema puede analizar un set de niveles y generar niveles que tengan secuencias de n-gramas similares a estos. El resultado es muy similar al de los niveles diseñados por humanos (Ilustración 2.11).

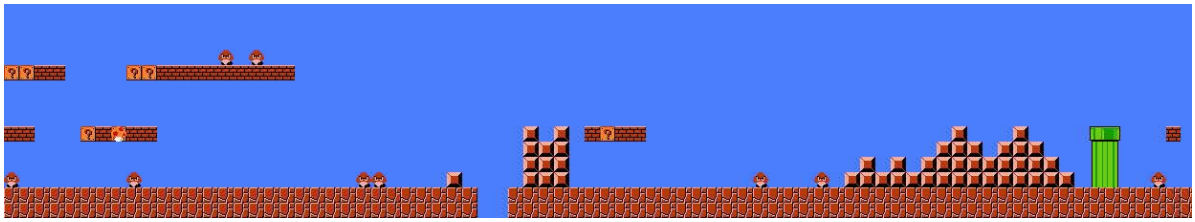

Ilustración 2.11. Nivel de Mario generado mediante n-gramas (Steve Dahlskog J. T., 2014).

Los n-gramas son sacados del juego original y se montan mediante una red neuronal. Uno de los problemas de los n-gramas es que no se garantiza que un nivel generado sea jugable, es decir, posible de llegar hasta el final.

En (Summerville, 2015) se plantea otra alternativa para la generación de niveles partiendo del modelo de Markov, pero utilizando aprendizaje automático y técnicas de Monte Carlo que evalúan si el diseño resultante es jugable y obteniendo otras métricas interesantes.

2.1.2.3 Angry Birds

Angry Birds (Rovio Entertainment, 2009) es un juego de puzzles desarrollado en Finlandia por Rovio Entertainment. Su mecánica básica es la de disparar pájaros a fortalezas que protegen cerdos.

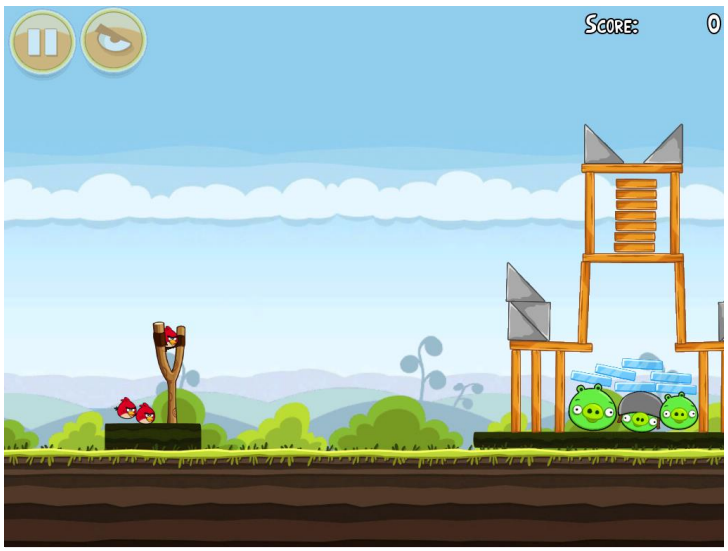

Ilustración 2.12.Captura de Angry Birds (Rovio Entertainment, 2009).

El estado general de una partida de Angry birds: a la izquierda, un tirachinas con un pájaro preparado para ser disparado, y N pájaros restantes. A la derecha, una fortaleza construida a base de bloques de madera y piedra que protege un número concreto de cerdos.

Por falta de código abierto, (Lucas Ferreira C. T., 2014) desarrollan un clon de Angry Birds con Unity. Los niveles son generados con un algoritmo fundamentado en medir el movimiento de los elementos del nivel durante un periodo de tiempo para poder crear estructuras estables.

Este algoritmo representa los niveles con un array de columnas. Realiza una simulación que mide cuánto se ha movido cada objeto con el objetivo es el de reducir la cantidad de movimiento al máximo.

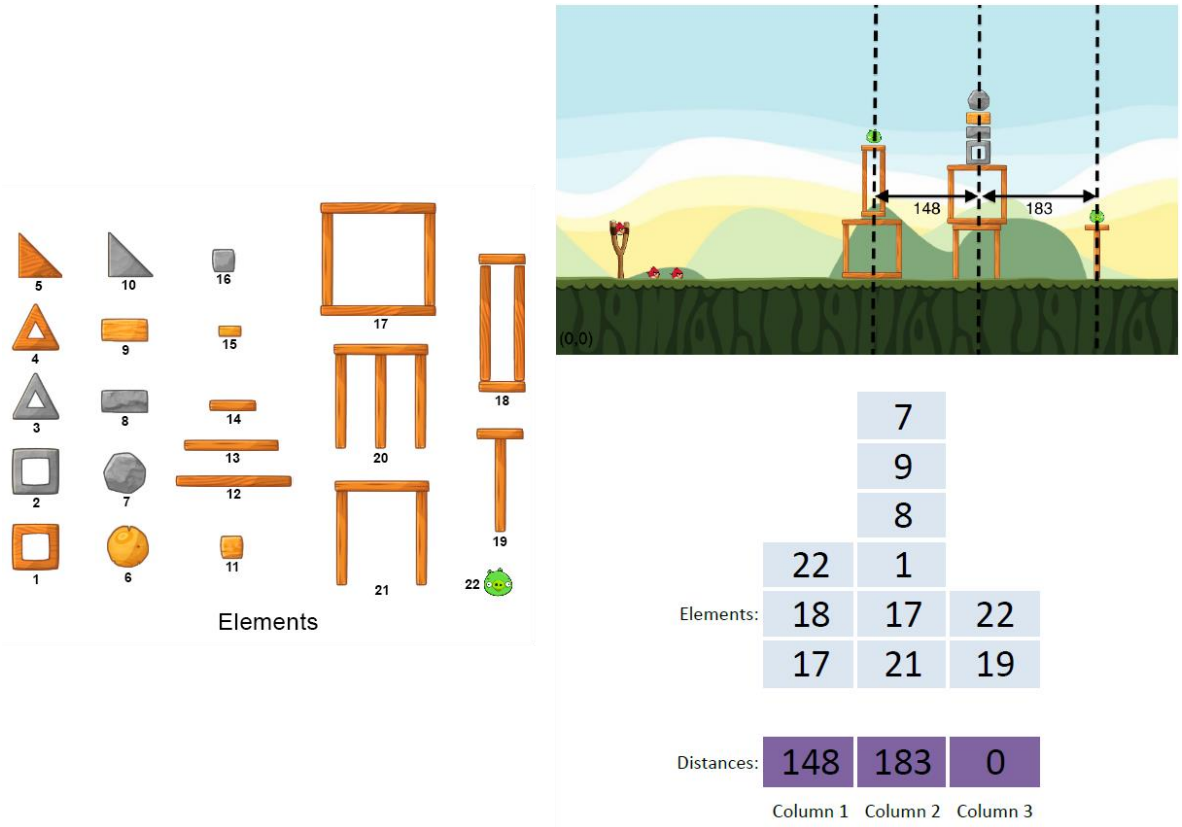

Ilustración 2.13. Elementos de los niveles y matriz de elementos, (Lucas Ferreira C. T., 2014)

Su funcionamiento consiste en generar un conjunto de niveles de manera semi-aleatoria, siguiendo unas tablas, y aplicarles transformaciones, rotaciones y mutaciones, este proceso se ejecuta un número concreto de veces, o hasta que el rendimiento del mejor de todos no se supera después de 10 iteraciones. Las mutaciones se producen fusionando niveles hijos.

2.1.2.4 Spelunky

Videojuego del género de aventuras y plataformas desarrollado por Derek Yu y Andy Hull en 2012. Destaca en la actualidad por su excelente diseño y su generación procedimental de niveles. Fue lanzado de manera gratuita en PC en 2009, y posteriormente tuvo una revisión de gráficos y controles para su lanzamiento comercial de 2012 en Xbox 360. Spelunky pertenece al género de plataformas, pero sucede en una cueva con niveles de una alta verticalidad si los comparamos con los de Super Mario Bros, por lo que podríamos asumir que es la perfecta fusión entre los géneros plataformas y *roguelike*.

Yu explica en su libro Spelunky (Yu, 2014) que para generar niveles crea una rejilla de 4x4 habitaciones, de 10 x 8 tiles (o celdas) cada una. Una de las 4 habitaciones de la fila superior es la entrada, después, se elige uno de los 4 lados de la habitación para que sirva de puerta a otra habitación, esto se realiza de manera recursiva con las siguientes habitaciones, generando un camino, hasta llegar al piso inferior y crear la salida de la cueva. De esta manera se garantiza que hay un camino para llegar al final del nivel. El resto de las habitaciones de la rejilla pueden estar conectadas con el camino o no.

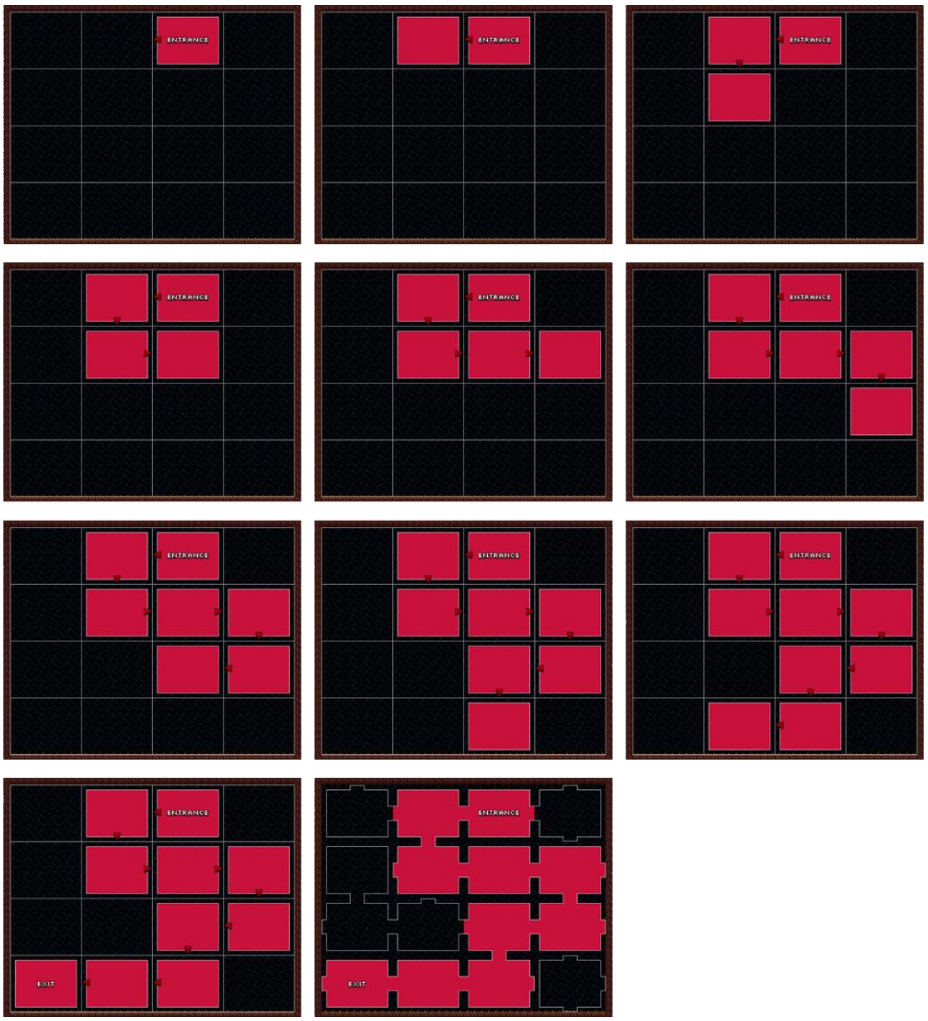

Ilustración 2.14. Secuencia de generación de habitaciones en Spelunky. Cada habitación tiene cuatro posibles salidas, una por cada lado, para comunicar con habitaciones adyacentes. Capturas extraídas de How (and Why) Spelunky Makes its Own Levels (Brown, M.)

En cada imagen hay una subdivisión en 16 partes iguales que se van convirtiendo en habitaciones consecutivas. En la última captura, se generan habitaciones alrededor de ese camino generado, estas pueden ser conexas o no.

El diseño de cada habitación, de manera individual, ha sido creado a mano por un humano, de manera que la IA lo que elige son *plantillas.* Pero estas plantillas sufren ciertas variaciones, la estructura general se mantiene, pero algunos tiles pueden no estar o pueden aparecer nuevos en distintas localizaciones de la habitación.

Por último, un script revisa todos los tiles del mapa y lanza un dado para generar un monstruo, un tesoro u otro objeto. No es del todo aleatorio, los objetos tienen una probabilidad distinta de aparición según los tiles de su alrededor, como, por ejemplo, una cueva rodeada de tierra tendrá mayor probabilidad de albergar un tesoro que un pasillo plano.

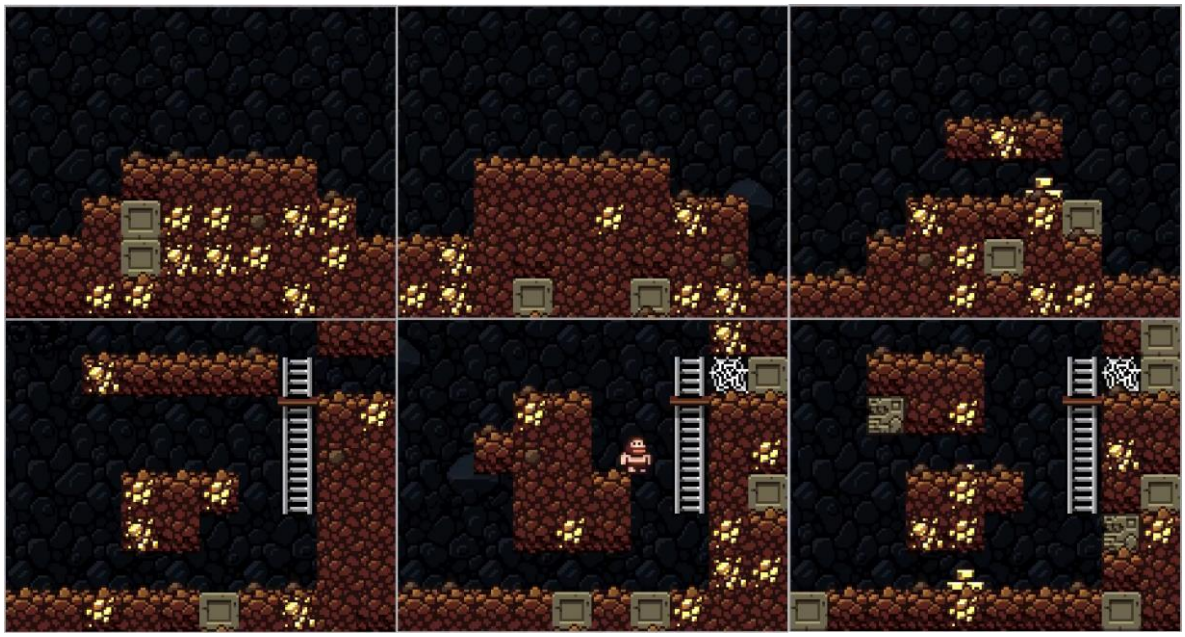

Ilustración 2.15. Ejemplo de plantillas mutadas. Capturas extraídas de How (and Why) Spelunky Makes its Own Levels (Brown, M.)

De este modo, el autor reconoce que no se generan cuevas realistas, y que incluso los jugadores podrían acabar intuyendo la rejilla, pero las mutaciones a las que se somete el mapa le aportan variedad y se crean niveles adictivos y con un buen diseño.

En (David Stammer, 2015) deciden explorar la creación de niveles tomando como motor a Spelunky. Su trabajo se basó en una única regla que podía aplicarse a otros juegos similares: generar un modelo del jugador, estimar su habilidad y generar niveles adaptados a esta. Esto lo hacían mediante un cuestionario, en el que el propio jugador indicaba su estilo de juego (pausado, agresivo o con prisa), y en base a unas tablas y el resultado de la última partida generaban niveles. La reacción que obtuvieron de otros jugadores fue positiva, pero el cambio brusco en la dificultad no terminaba de convencer.

2.1.3 Documentos sobre la generación de contenido adaptado a la habilidad del usuario

La generación procedimental de contenido supone una reducción en el coste y en el volumen de trabajo a la hora de desarrollar un videojuego, puede ser útil para la creación de elementos del juego, para generar estadísticas para los objetos (como armas o herramientas), para generar niveles, o incluso para adaptar la dificultad del juego. Años atrás el contenido resultada ser genérico y repetitivo, pero con el tiempo se está perfeccionando el comportamiento de sus algoritmos. Además, cada vez se está aplicando a más géneros.

2.1.3.1 Juego de tipo Runner

En (Rubem Jose Vasconcelos de Medeiros, 2014) se crea un juego de scroll lateral, con cinco raíles para desplazarse. Se decidió experimentar con la creación de un algoritmo que ajustase diferentes características del juego, como eventos o aparición de enemigos, para adaptar el grado de complejidad a la habilidad del jugador en función de su retroalimentación.

Para realizar el ajuste de dificultad utilizan algoritmos de aprendizaje por refuerzo, un área del aprendizaje automático motivada por la psicología del comportamiento. En su proyecto, un agente genera un nivel combinando diferentes estados, que son configuraciones de las características de la aplicación. Al algoritmo se le pasa un refuerzo cada 30 segundos, que para el usuario es un cuestionario, y para el agente un indicador de su conducta.

Llevan a cabo dos experimentos: el experimento A, que utiliza las secciones de niveles más populares para generar niveles. En otras palabras, los usuarios votan la diversión de cada fase y las menos divertidas se escogen con menos frecuencia. En el experimento B, se utiliza un algoritmo de aprendizaje rápido que observa las diferencias entre fases teniendo en cuenta la *distancia Hamming* entre ellas.

La retroalimentación obtenida se basa en la entrada del jugador, en la cantidad de vida que perdió y en qué momento, y en un cuestionario que recoge información sobre la percepción de la dificultad y la diversión de ese nivel.

Los resultados del experimento muestran que el experimento B, el algoritmo de aprendizaje rápido, proporciona experiencias más divertidas para el jugador. Mediante este algoritmo, el modelo aprende combinaciones de parámetros que generan fases divertidas, por lo que podría generar niveles divertidos que nadie ha puntuado previamente.

2.1.3.2 Juego de tanques para 2 jugadores

Un caso en el que se pretende conseguir una experiencia satisfactoria para dos jugadores al mismo tiempo resulta interesante (Víctor Manuel Álvarez Pato, 2013). En este caso, se escoge un videojuego de tipo tanques competitivo de un jugador contra otro. El mapa es de tipo laberinto, y el jugador que haya disparado más veces al rival se convierte en ganador.

El experimento comenzaba con un cuestionario a un set de 13 usuarios. En la siguiente fase, enfrentaban a jugadores experimentados contra novatos, para que el sistema se activase si notaba mucha diferencia entre marcadores. La forma en que se eligió equilibrar la partida consistía en mejorar las estadísticas del que iba perdiendo, y reducir las del ganador, para que pudiese conseguir puntos.

Es un prototipo sencillo y una aproximación un tanto pobre. Equilibrar de esta forma la partida altera las reglas del videojuego y lo convierte en una experiencia injusta. Para sorpresa de Álvarez y Delgado, tanto jugadores experimentados como novatos se sentían más frustrados y aburridos cuando el sistema de balanceado automático estaba activo: "*Expectations were that when introducing dynamic adjustment, the more experienced players would feel less bored, which is congruent with what is shown in (Fig. 4a). Also, the graph shows that the dynamically balanced game makes them feel greater difficulty, frustration and lack of interest for the game*".

2.1.3.3 Banco de pruebas para el ajuste dinámico de dificultad en juegos modernos

(Christine Bailey) Plantean un sistema que se encarga de monitorizar, analizar y controlar las acciones del usuario para averiguar qué tipo de jugador es.

Para ello, realizan un estudio sobre qué parámetros de un videojuego pueden ser ajustados dinámicamente, así como el cómo y el cuándo. Dividen los elementos del juego en:

- Atributos del jugador: velocidad, habilidad de salto, fuerza, cantidad de vida, ayuda al apuntado… Algunos atributos son comunes en la mayoría de los videojuegos, pero otros muchos están vinculados a un género concreto.
- Atributos de no-jugador: el grado de refinamiento de la inteligencia artificial, como la velocidad en la búsqueda de caminos, precisión del apuntado, velocidad de reacción… atributos que, según su grado, hacen a los enemigos inteligentes o torpes.
- Atributos del mundo y niveles: relativos a la estructura del nivel. Dependientes del género: modificación del tamaño de los saltos en un plataformas o el número de coberturas en un juego de disparos. Se pueden modificar también los elementos que aparecen o la frecuencia con la que lo hacen como los enemigos, los botiquines…

Para detectar el tipo de jugador, utilizan funciones que miden de manera constante el rendimiento de este en su capacidad de resolver puzles, de movimiento, habilidad y precisión independientemente.

Implementan 4 mini-juegos de distinto género y observan las reacciones de los usuarios a los ajustes de dificultad. Los escenarios propuestos son: dos juegos de tipo plataformas con saltos,

uno con penalización de los fallos y el otro no; un juego de navegación en primera persona por un laberinto 3D y, por último, un juego de disparos en primera persona.

Para evaluar el tipo de jugador se puede analizar la presión con la que pulsa las teclas o la frecuencia con la que las pulsa. El movimiento del usuario también puede indicarnos qué tipo de jugador es, si es explorador, novato, o tenaz, es decir, que va directo al objetivo sin examinar el entorno. Evaluando cuánto tiempo dedica el jugador a completar el objetivo también ayuda a generar un perfil.

Se expone que en ocasiones no es necesario hacer ajustes de dificultad si por ejemplo el usuario se encuentra una tarea retadora, pero este es de tipo consigue-objetivos, le gusta superar retos. A otros usuarios no les gusta "que los lleven de la mano".

Los resultados son positivos, pero se evidencia que necesitan más usuarios y mejorar los algoritmos de análisis. También se ve reflejado que, para tener una auténtica experiencia ajustable de manera dinámica, es necesario tener un juego completo, con un diseño complejo y desarrollado, una historia y un sistema de recompensas.

2.1.3.4 Ajuste dinámico de dificultad con redes neuronales profundas

En (Hee-Seung Moon, 2020) se propone un sistema que configura las variables del juego para adaptarse a las habilidades del jugador en el menor tiempo posible y con un banco de datos muy reducido (tan solo 9 usuarios).

Este proyecto propone un modelo de Ajuste de Dificultad Dinámico (DDA) basado en redes neuronales profundas que hace uso del algoritmo MAML (Model-Agnostic Meta-Learning), que busca entrenar IAs con el menor número de datos posible y generalizando parámetros para estar preparado para posibles nuevos problemas.

Además, con los datos que recogen de los usuarios moldean el comportamiento de agentes para que cada jugador se enfrente a una IA que juega parecido a él o a ella, con hasta 9 posibles configuraciones.

A través de un videojuego de *Air-Hockey*, se implementaron dos aproximaciones para el experimento, su modelo con el algoritmo MAML junto a una DDA convencional, y otra que hace uso de redes neuronales profundas. En el primer modelo, existen 9 dificultades diferentes y el jugador cambia de una a otra mediante derrotas o victorias. Para la red neuronal, el sistema extraía información del rendimiento del jugador y lo organiza en capas.

Los tiempos de entrenamiento fueron de 2 horas el sistema de aprendizaje rápido contra 18 horas en el modelo de redes neuronales profundas.

Se midieron los resultados con métricas objetivas y con cuestionarios subjetivos a los usuarios. Los resultados obtenidos fueron positivos. Se logró implementar un sistema que obtenía una satisfacción igual al método convencional de ajuste de dificultad, pero empleando mucho menos tiempo que una red neuronal profunda gracias a algoritmos de autoaprendizaje (Meta-learning).

2.2 Aprendizaje automático. Aplicaciones

La Inteligencia Artificial es una subrama de la informática encargada de la creación de máquinas que imitan comportamientos inteligentes. Estos comportamientos son muy diversos, desde conducir y reconocer patrones, hasta jugar partidas de ajedrez (Santana, 2017). El aprendizaje automático conforma uno de los múltiples campos de la inteligencia artificial, y su comportamiento característico viene indicado por su nombre, el de aprendizaje. Si bien los algoritmos de la inteligencia artificial clásica están diseñados para realizar un comportamiento, los de aprendizaje automático están planteados para aprender a realizarlo. Se puede resumir su actuación como la de generalizar fenómenos para poder predecir resultados. Este propósito se consigue mediante el entrenamiento de unos sistemas a través unos datos de prueba, para después predecir casos reales.

Forman parte del aprendizaje automático sistemas, métodos o técnicas diferentes como los árboles de decisión, modelos de regresión, de clasificación y los más conocidos, las redes neuronales.

Estos modelos tienen aplicación en problemas donde se busca obtener una función que describa un comportamiento, por ejemplo, el precio de las viviendas dado su número de habitaciones. Una vez obtenida esta función, se pueden predecir casos futuros.

Cuanto mayor es el número de variables que modelan la realidad para la cual queremos predecir un comportamiento, más complejo es el problema. Según aumenta el grado de dificultad de los problemas, es necesario recurrir a sistemas más sofisticados y potentes para su solución. Los sistemas más conocidos son las redes neuronales, que generan soluciones en espacios de múltiples dimensiones, o lo que es lo mismo, problemas en los que entran en juego un alto número de variables.

También disponemos de las Máquinas de soporte vectorial, que se utilizan para resolver problemas de clasificación y regresión. Este proyecto hace uso de máquinas de soporte vectorial, cuyos algoritmos funcionan como una especie de caja negra y dan como resultado una salida que es el hiperplano que mejor define un fenómeno y separa los grupos de variables implicados. Esto lo hace mediante un conjunto de vectores que buscan controlar la distancia entre el hiperplano y los grupos de variables que se pretenden clasificar ampliando esa distancia lo máximo posible. Es un sistema robusto y fácil de aplicar.

2.3 Dificultad, curva de aprendizaje y videojuegos comerciales con dificultad adaptativa

El término *dificultad* se entiende como "Embarazo, inconveniente, oposición o contrariedad que impide conseguir, ejecutar o entender algo bien y pronto." (Real Academia Española, 2021).

En videojuegos, entendemos dificultad como el grado de complejidad del obstáculo a superar. Existen una alta variedad de videojuegos, por lo que la dificultad no siempre reside en los mismos elementos. Atendiendo a su género, en (Fernández, 2018) se dan múltiples definiciones de dificultad, referida a diferentes tipos de obstáculo:

- Dificultad física: "condición biológica, de todas las habilidades cognitivas y motoras que posee una persona. De los reflejos, del tiempo de respuesta que tardamos en esquivar, en pulsar X y saltar."
- Dificultad mental: "es un tipo de dificultad deductiva, nuestra agilidad ante la resolución de rompecabezas. No sólo hay que intuir el posible futuro (la solución), sino que hay que llevarla a cabo mediante una estrategia."
- Dificultad emocional: "entran en juego factores ambientales. La empatía, el miedo, la ansiedad. Los mecanismos de cualquier juego de terror".

A medida que se progresa en un videojuego, su dificultad aumenta, pero no lo hace de forma constante en línea recta. En (Csikszentmihalyi, 1990) se recoge por primera vez *La teoría del flujo*, un fenómeno estudiado por psicólogos y utilizado por diseñadores de videojuegos para alcanzar una experiencia de juego óptima. Según el libro, el flujo es *"U*na sensación de que las propias habilidades son adecuadas para enfrentarse con los desafíos que se nos presentan, en una actividad dirigida hacia unas metas y regulada por normas que, además, nos ofrece unas pistas claras para saber si lo estamos haciendo bien. La concentración es tan intensa que no se puede prestar atención a pensar en cosas irrelevantes respecto a la actividad que se está realizando.*"*

Esta experiencia podemos encontrarla en actividades cotidianas como, por ejemplo, contando anécdotas con amigos, pero es mucho más fácil hallar el flujo en actividades estructuradas como pueden ser el deporte, el baile, la escalada o el ajedrez.

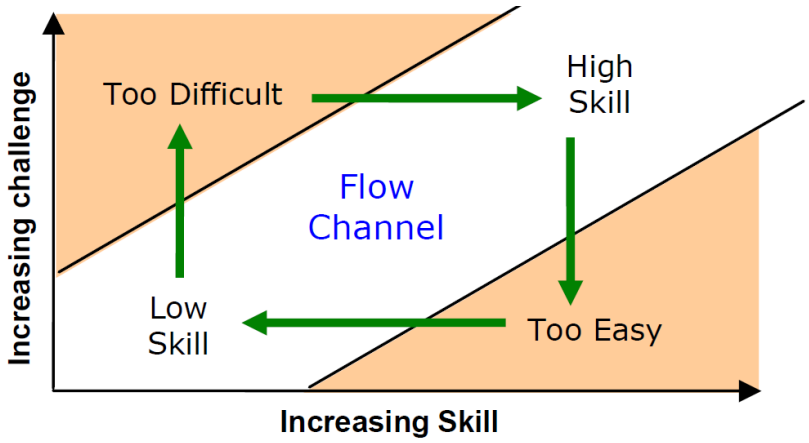

Ilustración 2.16. Representación del flujo, (Robin Hunicke)

A medida que aumenta la habilidad del usuario, debe subir el grado de dificultad de la actividad. Si aumenta de manera desproporcionada puede salirse de la zona de flujo y volverse muy fácil o frustrante.

"Lo que hace que estas actividades produzcan flujo es que se diseñaron para hacer más fácil lograr la experiencia óptima. Tienen unas reglas que requieren de un aprendizaje de habilidades, establecen metas, producen retroalimentación, hacen posible el control", (Csikszentmihalyi, 1990).

Al leer esta definición es inevitable pensar en un videojuego, un entorno en el que se nos plantean unos retos siguiendo unas normas (mecánicas) para obtener una recompensa.

Para poder obtener una experiencia óptima, esta ha de ser adictiva y contar con un buen ajuste de dificultad. Hay que entender qué experimenta un jugador y cómo controla eso un videojuego. Conseguir una experiencia que no sea muy fácil y aburra, pero tampoco sea excesivamente difícil y frustre. Además, el jugador debe entender qué está pasando en pantalla y cómo controlar la acción. Esto se consigue con un diseño que posicione al jugador continuamente en la zona de flujo y a su vez, esto se logra mediante una buena curva de aprendizaje, esto es, mostrándole al jugador el entorno y enseñándole a jugar progresivamente.

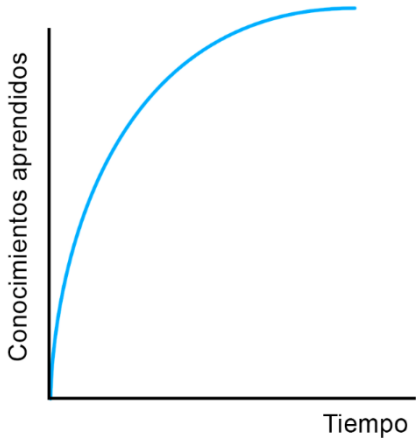

Ilustración 2.17 Gráfica de la curva de aprendizaje. Muestra cómo la cantidad de conocimientos aumenta conforme avanza el tiempo invertido en desarrollar una tarea

Algunos videojuegos hacen uso de la frustración del jugador como forma de enseñarle las reglas del juego, como la saga Dark Souls (From Software, 2011). Mediante ensayo y error, descubre las mecánicas del juego y el entorno. Otros videojuegos, sin embargo, buscan evitar que se produzca esa sensación de fracaso y ajustan sus reglas para favorecer la victoria.

El ajuste de la dificultad o "ajuste de dificultad dinámico" busca adecuarse a las habilidades del jugador para evitar la frustración y sucede de manera invisible para que el jugador no se sienta subestimado y se potencie su autoestima, como sucede en el caso de Mario Kart (Nintendo, 1992-2019), donde se introdujo la técnica *Rubber banding*. Esta técnica modela las reglas del juego para hacer que los jugadores que se encuentran en desventaja vean su suerte cambiada de manera rápida y sencilla. Por ejemplo, mediante la obtención de mejoras mucho más potentes que las del resto de jugadores.

Hay juegos de puzles que, pasado un tiempo sin avanzar sobre el problema, echan una mano al jugador en forma de pista: una pieza brilla o un personaje realiza un comentario en voz alta insinuando una pista.

El ajuste de dificultad dinámico no sucede siempre a espaldas del jugador, hay títulos que muestran este cambio mediante la interfaz para motivar al usuario a que se esfuerce lo máximo posible. Encontramos un ejemplo de esto en God Hand (Clover Studio, 2006), un juego de peleas donde, en función del rendimiento del jugador, aparece por pantalla un mensaje indicando que los contrincantes ahora son más exigentes o lo contrario.

A continuación, se muestra un listado de videojuegos que hacen uso del ajuste de dificultad dinámico de diferentes modos:

Space Invaders (Nishikado, 1978) un clásico juego de naves que se vuelve más difícil cuantas más naves destruyas. El juego no fue programado para comportarse así, este comportamiento surgía por un fallo de hardware.

Zanac (Compile, 1986) juego de naves que cuenta con una IA que adapta la dificultad del juego de manera automática en función de la habilidad del jugador, la cadencia de fuego y el estado de la nave.

Crash Bandicoot (Naughty Dog, 1996) este conocido plataformas 3D ajusta la dificultad con cada derrota del jugador, volviendo los obstáculos más lentos y sencillos, o añadiendo un extra de vida al reaparecer.

Omega boost (Polyphony Digital, 1999) juego de naves en 3D que ajusta la dificultad como Zanac, variando los atributos de los enemigos y el número de obstáculos para volver la experiencia de juego más atractiva.

Max Payne (Remedy, 2001) juego de disparos en tercera persona que, en función de la puntería del jugador, varía el nivel de ayuda al apuntado.

Mario Kart (Nintendo, 1992-2019) introduce el término *Rubber banding:* (goma elástica en español) en el que, de manera dinámica, los NPCs que están situados en desventaja son

ayudados por las reglas del juego, obteniendo mejoras, o siendo penalizados de manera más leve, creando una experiencia muy dinámica.

Resident evil 4 (Capcom, 2005) *shooter* en tercera persona, su dificultad se adapta en función de la puntería del jugador y el daño recibido. El número de enemigos cambia, así como su habilidad, su agresividad y destreza.

Ilustración 2.18: Capturas de Resident Evil 4. A la izquierda, figura original, con 7 enemigos y 2 tiradores. Tras un número determinado de muertes, se realiza el ajuste automático de dificultad, reflejado en la imagen de la derecha, donde no hay tiradores.

God Hand (Clover Studio, 2006) juego de aventuras en tercera persona, adapta la IA y fuerza de enemigos mostrándolo por pantalla, siendo el primer juego que muestra de manera transparente para el jugador este cambio en el sistema, como recurso de diseño.

Ilustración 2.19. Captura del juego God Hand. Se muestra un combate entre el jugador y un enemigo, al mismo tiempo que la interfaz muestra un aumento en el nivel de dificultad del enemigo.

Fallout 3 (Bethesda, 2008) *Shooter* en primera persona de mundo abierto. Si tu personaje sube de nivel, los enemigos también.

Left 4 Dead (Valve, 2008) FPS de zombies. Introduce *The AI director,* una inteligencia artificial que mide el nivel de estrés del jugador en función del daño recibido, el número de zombies que elimina, y lo rápido que avanza, con el objetivo de mantenerle en constante tensión.

Amnesia: dark descent (Frictional Games, 2010) Juego en primera persona de terror, si fallas a menudo los encuentros con enemigos desaparecen.

Metal Gear Solid V: The Phantom Pain (Konami, 2015) juego de sigilo y estrategia militar en tercera persona. En los puestos de soldados que el jugador asalta, el juego evalúa la estrategia llevada a cabo por el jugador y cambiará el escenario y a los enemigos para que tenga que cambiar de estrategia al enfrentarse a estos.

2.4 Tecnologías relevantes utilizadas

En esta sección se exponen las herramientas y librerías de mayor peso utilizadas en el desarrollo de la aplicación y en el procesamiento de los datos.

2.4.1 Unity 2018

Desarrollado por Unity Technologies, es uno de los motores de videojuegos más utilizado en la actualidad junto con Unreal Engine. Incluye motor gráfico, motor de físicas, gestión del input, utilidades para desarrollar juegos online y otras herramientas para que el programador solo deba encargarse de la lógica del juego. Tiene soporte para compilación en múltiples plataformas y el lenguaje utilizado por sus scripts es C#, aunque también puede utilizarse UnityScript y Boo.

Es muy utilizado por la comunidad indie de desarrolladores de videojuegos, con ejemplos como Death's Door (Acid Nerve, 2021) y Hollow knight (Team Cherry, 2017), aunque también ha sido utilizado en grandes producciones como ReCore (Asobo Studio, 2016) o Hearthstone (Blizzard Entertainment, 2014).

Su utilidad para el proyecto es la de servir como herramienta para el desarrollo de la aplicación y su posterior exportación en formato WebGL, así como el ejecutable de la versión de escritorio.

2.4.2 Dungeon Tools

Desarrollado por Shekn Itrch (Itrch, 2015). Es un asset gratuito disponible en la Asset Store de Unity. Genera mazmorras en un plano mediante un conjunto de objetos de tipo suelo o pared que forman una rejilla de casillas. Dados unos parámetros que modelan el tamaño, número de habitaciones y de pasillos, se generan mazmorras con unas características concretas.

Se modifica el comportamiento del asset para adaptarlo a las necesidades del proyecto, integrándolo con las demás clases y gestores de dificultad.

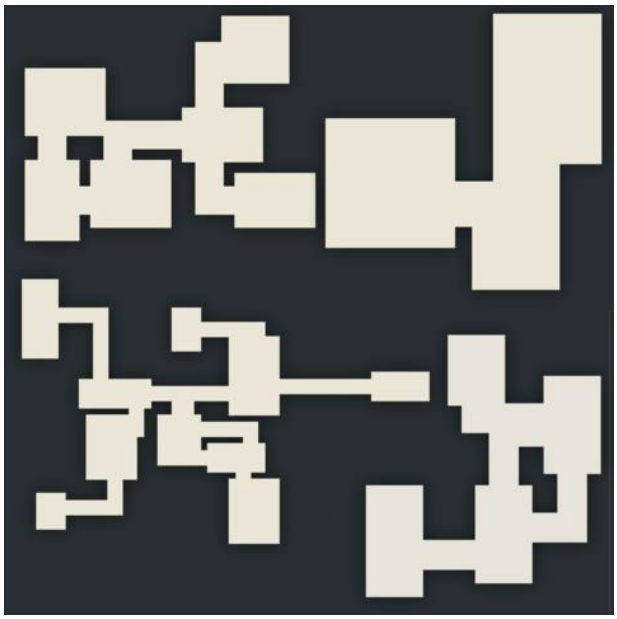

Ilustración 2.20. Mapas generados con Dungeon Tools (Itrch, 2015)

2.4.3 Material-UI

Conjunto de herramientas para generar cuestionarios. En fase beta y desarrollado por InvexGames, contiene diferentes assets con un estilo visual atractivo, utilizado en los formularios de Google. Los assets son insertados en la aplicación de Unity para contar con cuestionarios dentro del juego, sin necesidad de recurrir a un navegador externo. De sus utilidades se han empleado switches, botones, paneles, campos de texto y *sliders* en los cuestionarios, en el formulario inicial y en la encuesta de satisfacción.

2.4.4 Json-server

Desarrollado por typicode, es una REST-API muy fácil de configurar, descargable a través de Node. La API REST (Representational State Transfer) es un estándar que permite la comunicación entre máquinas mediante HTTP. Utilizada por empresas como Twitter, Youtube o Facebook, es eficiente y habitual en la creación de APIs para servicios de Internet (BBVAOpen4u, 2016).

Se utiliza en dos soluciones del proyecto: Por un lado, el videojuego realiza una llamada POST al servidor al terminar una ronda para enviar las trazas capturadas. Por otro lado, la aplicación encargada de procesar los datos almacenados en el servidor y generar un archivo de información csv utiliza la llamada GET para descargar estos datos.

2.4.5 Rstudio y LibSVM

Se recurre al lenguaje estadístico R para el tratamiento de datos una vez recogidos y organizados en un archivo csv. El entorno escogido para su empleo es Rstudio, que facilita las tareas de depuración y de visualización de datos.

La librería empleada para utilizar Máquinas de soporte vectorial (SVM), el modelo de aprendizaje automático escogido, es LibSVM, que lleva a cabo tareas de clasificación y predicción. Gracias a esta librería se ahorra tiempo de implementación y mediante pocas llamadas se genera un modelo entrenado, capaz de producir nuevos mapas.

Las máquinas de soporte vectorial pueden ser utilizadas para regresión y para clasificación. Su funcionamiento consiste en, dado un conjunto de grupos o clases, buscar el hiperplano que los separe o clasifique, maximizando el margen entre estos grupos. Para lograr maximizar el margen se utilizan los vectores de soporte, que controlan la distancia entre el hiperplano y los grupos de variables.

Las ventajas del modelo SVM es que son fáciles de utilizar y multi-dimensionales. Entre sus aplicaciones encontramos reconocimiento de caracteres, detección de caras o filtros de spam (Heras, 2019).

3. Generación de niveles con ajuste de dificultad

En este capítulo se exponen las diferentes versiones por las que pasa el proyecto a lo largo de su desarrollo, así como la recogida de datos en cada una de estas. Con un total de 4 versiones, en este capítulo se detallan su diseño, el modelo de dificultad utilizado, la jugabilidad e interfaz.

Todas las versiones tienen en común el uso de algoritmos de generación de mazmorras ajustando sus parámetros mediante una inteligencia artificial para adaptar la dificultad a la habilidad del jugador, aunque en las primeras versiones estos parámetros se fijan gracias a una fórmula hecha a mano. Las principales características y diferencias entre versiones las encontramos en la siguiente lista:

- El prototipo del videojuego cuenta con la funcionalidad mínima de movimiento, disparo, comportamiento de enemigos y generación de mapas procedimentalmente. La dificultad es elegida siguiendo unas fórmulas hechas a mano. Esta versión es útil para encontrar fallos en el diseño del juego y pulir aspectos de su configuración y jugabilidad. Fue probada por 10 usuarios familiarizados con los videojuegos. Las versiones expuestas a continuación fueron utilizadas para recoger datos que modelasen la inteligencia artificial, al contrario de esta versión.
- En la primera versión de la aplicación los niveles son creados a partir de unas variables generadas de manera aleatoria, dentro de unos rangos. Entre niveles el jugador contesta a un breve cuestionario acerca de la dificultad del nivel y de la diversión que ha experimentado, evaluando ambas con una nota del 1 al 5. La intención es la de construir un sistema que, dado un conjunto de datos relativos a la generación del nivel y las respuestas de los usuarios, detecte dónde reside el reto y lo adapte a los jugadores. Se recogen datos de 327 usuarios. Los resultados obtenidos no muestran correlación entre las respuestas al cuestionario y los parámetros de generación del nivel. Esto se debe a diversos factores: la malinterpretación del cuestionario, pues muchos jugadores entienden que sus respuestas modelan la generación de mapas; la generación mediante variables aleatorias da como resultado mapas muy heterogéneos; y el formulario inicial está incompleto y no aporta información relevante sobre el jugador.
- La segunda versión decide enfocar sus esfuerzos en evaluar la dificultad de los niveles, pues estimar la diversión es más complicado y subjetivo. En (David Stammer, 2015), realizan un estudio previo sobre los sujetos de pruebas. En su trabajo, se exponen características de los usuarios que son relevantes y no se habían tenido en cuenta en la primera versión, como la edad, familiaridad con los videojuegos y el sexo. A raíz de esto, se decide modificar el formulario inicial para conocer mejor al público, ahora se pregunta la edad y si se considera un jugador habitual, el sexo no se considera relevante para esta prueba. El proyecto tiene más control sobre los parámetros que generan los mapas. Para evitar la heterogeneidad, las variables tienen 3 posibles valores, y se eligen teniendo en cuenta el resto de los parámetros, para crear una experiencia más coherente y de dificultad fácil de identificar. Los resultados muestran correlaciones más fuertes y útiles para entrenar a la máquina de soporte vectorial.

- La tercera versión del proyecto hace uso de los datos obtenidos en la segunda versión para realizar predicciones y generar mapas acordes a estas. Esta versión cuenta con una fórmula que modela la dificultad deseada de manera progresiva, tiene en cuenta las respuestas a los cuestionarios y, mediante el modelo de Máquinas de soporte vectorial, genera mapas adaptados. Para contrastar resultados, el software divide a sus usuarios de forma que un tercio de estos constituye el grupo de control, un grupo de usuarios cuya aplicación no cuenta con inteligencia artificial, para poder comparar los resultados con los del resto.

3.1 Diseño del prototipo jugable

La intención del proyecto es la de tener siempre a mano una versión jugable, un juego sencillo que sufre ligeros cambios, pero siempre es funcional.

Para crear el prototipo se cuidaron aspectos del diseño que influyen estrechamente en la experiencia de juego, como la perspectiva, la estética, la selección de colores o el control. El objetivo de los primeros meses era crear una demo lo más aséptica posible para reducir al máximo el sesgo del experimento.

3.1.1 Elección de perspectiva: 2D vs. 3D

El juego tiene una perspectiva 2D por la facilidad de desarrollo que aporta trabajar en dos dimensiones. Además, el punto de vista afecta a la dificultad de control. En un juego 3D, tanto en tercera persona como en primera persona, requieren de la colocación adecuada de la cámara por parte del jugador para obtener una puntería exitosa, y se necesita una coordinación con la que no todos los usuarios cuentan. Del mismo modo, algunas personas no acostumbradas a jugar incluso se marean o desorientan con esta perspectiva.

Dentro del 2D encontramos dos aproximaciones posibles: juegos de plataformas, como *Super Mario Bros* (Miyamoto, 1983), o juegos con vista desde arriba, como *The Legend of Zelda* (Shigeru Miyamoto, 1986).

El proyecto hace uso de la vista desde arriba porque, comparando cualidades entre ambos géneros, el de plataformas resulta más exigente. Introduce el sesgo del control: es más complicado ya que incluye el salto y existen muchas maneras de implementarlo: doble salto, una pulsación, mantener el botón para prolongar la distancia de salto… Además, el movimiento lateral puede incluir aceleración.

Sin embargo, la vista desde arriba simplifica el movimiento del personaje mucho más. La vista aérea ahorra este posible sesgo y permite centrar los esfuerzos en la generación procedimental del mapa y sus obstáculos.

3.1.2 Selección del estilo visual. Estética.

Los elementos del juego se representan mediante formas geométricas básicas para ahorrar tiempo de desarrollo y para que sean fácilmente reconocibles. Esta abstracción surge para evitar añadir elementos distractores o que dificulten la visión. El uso de *sprites* (colección de mapa de bits) también podría no dejar claro qué propósito tienen los elementos en el juego, impidiendo, por ejemplo, identificar a un enemigo como tal. Utilizar formas sencillas permite reconocer fácilmente los elementos del entorno en el que tiene lugar el videojuego.

3.1.3 Control del avatar. Tipo de movimiento y teclas asignadas.

El número de teclas que tiene el juego permite cierto grado de complejidad en la jugabilidad, pero sin resultar abrumador para el público que no está acostumbrado a jugar. El proyecto utiliza las teclas ‘w’, ‘a’, ‘s’ y ‘d’ como flechas de dirección, las teclas más utilizadas en la mayoría de los juegos para ordenador. El clic izquierdo del ratón se utiliza para disparar proyectiles, cuya trayectoria sigue una línea recta desde el avatar hacia el cursor del ratón. Para beneficiar la accesibilidad, no es necesario hacer clic repetidamente sobre el ratón para disparar, basta con mantenerlo pulsado y se obtiene la misma cadencia de fuego.

Durante el desarrollo surgen dudas sobre cuánto rango de visión debe tener el jugador y sobre si darle control o no sobre el tamaño de la región de visión de la cámara. En esta fase se experimenta con la posibilidad de otorgar control total sobre el campo de visión en un grupo reducido de personas para observar los valores elegidos. En la totalidad de los casos los jugadores hacen girar la rueda del ratón para aumentar la distancia de la cámara hasta el máximo permitido. De esta manera, pueden ver a los enemigos y las habitaciones adyacentes, así como sus coleccionables.

Una distancia corta de la cámara al jugador resulta agobiante y exige reacciones inmediatas para defenderse de los enemigos, mientras que una distancia excesivamente larga permite visualizar el mapa completo y anticiparse ante cualquier sorpresa, así como poder planificar previamente la ruta hacia los coleccionables y perder el factor exploración. Por este motivo, en la siguiente iteración se fijará un valor para la cámara no modificable por el jugador.

3.1.4 Desglose de elementos del videojuego

Para dar un funcionamiento y objetivo básico al proyecto, existen diversos elementos que se encargan de generar respuestas en el usuario, los enemigos y los coleccionables. Los primeros suponen una amenaza para el jugador, mientras que los segundos representan el objetivo a cumplir durante el nivel.

Se sopesa también la posibilidad de ampliar la variedad de enemigos, introduciendo algunos que puedan disparar y, por último, que las balas, tanto enemigas como propias, reboten, pudiendo dañar al jugador en el camino de vuelta, pero se descartan por simplificar el juego lo máximo posible y por falta de tiempo.

Si hacemos un análisis de los elementos básicos del juego y de la experiencia que formarán parte de la ecuación de dificultad, podemos dividirlos en dos grandes grupos, elementos del entorno, y atributos de rendimiento del jugador.

Los atributos del entorno son:

- Tamaño de la mazmorra.
- Número de habitaciones de la mazmorra.
- Tamaño mínimo de una habitación.
- Número de pasillos.
- Número de coleccionables.
- Número de enemigos.
- Daño efectuado por los enemigos.
- Velocidad de los enemigos.
- Área de activación de los enemigos.
- Tiempo de recuperación de los enemigos.

Mientras que el rendimiento del jugador se medirá teniendo en cuenta:

- Número de partidas jugadas
- Racha de victorias o derrotas
- Enemigos derrotados
- Precisión
- Tiempo de ronda
- Victoria/Derrota
- Número de coleccionables recogidos

3.1.5 Generación de mapas

Creado sobre el motor de videojuegos Unity, el primer paso es la creación de un entorno sencillo 2D. La acción tiene lugar sobre el plano, que hace de suelo y un triángulo hace de jugador. Lo primero en implementar es el movimiento, después el seguimiento de la cámara y, por último, el disparo.

Para generar mapas lo suficientemente refinados se recurre al libro *Procedural content generation in games,* (SHAKER, 2016) para consultar los diferentes métodos de generación procedimental de mapas, como quadtrees, árboles BSP, sistemas basados en agentes…

Tras unos días dedicados a la implementación, es evidente que recurrir a un asset para resolver este problema ahorra tiempo y se utiliza Dungeon Tools, (Itrch, 2015), de acceso público, que genera mapas de tipo mazmorra mediante quadtrees. La herramienta permite modificar los parámetros de la mazmorra como el número de habitaciones, de pasillos, el tamaño mínimo de ancho, largo y el tamaño general de la mazmorra. El número de pasillos que unen las habitaciones se fija a uno. Se modifica el código de este *asset* para que se integre con la aplicación y sea el código el que llame a las funciones del generador de mazmorras, sin

necesidad de abrir la ventana del editor. Los parámetros de generación se transforman en atributos públicos de fácil acceso, se añaden nuevos elementos al generador, como el jugador, los coleccionables y los enemigos, así como funciones que se encargan de colocarlos en el mapa.

3.1.6 Mecánicas

Para superar los niveles se debe conseguir un número concreto de coleccionables repartidos por el mapa. Dicho número aparece en la esquina superior izquierda de la pantalla.

- Coleccionables: están representados como un cubo dorado que rota sobre sí mismo y se recogen entrando en contacto con ellos. Están distribuidos de manera aleatoria por el mapa.
- Enemigos: representados como triángulos de color negro, sirven como impedimento para lograr superar el nivel. Se lanzarán directos hacia el jugador cuando este entre dentro de su rango de acción. Si logran colisionar con el jugador, este recibe daño. Al recibir un disparo, parpadean en rojo. Cuando los enemigos pierden por completo la vida, desaparecen. La velocidad de los enemigos, el daño infligido, así como la velocidad de reacción, aumentarán junto a la dificultad.
- Movimiento: cuenta con 8 posibles direcciones: horizontales, verticales y diagonales. La velocidad es constante y se ve limitado por los muros del mapa.
- Disparo: útil para deshacerse de los enemigos. El número de balas es ilimitado, no requiere recargar el arma y tiene una cadencia fija. El daño que hacen los proyectiles sobre el enemigo disminuye conforme sube la dificultad.

3.1.7 Modelo de dificultad

La dificultad es representada como un entero comprendido entre 1 y 10 y se modifica en función del éxito del jugador, es decir, si supera o no el nivel. Para dar valor a la dificultad se hace uso de las variables *racha de derrotas y racha de victorias*.

Se elige el 2 como punto de partida para la dificultad. Cuando se pierde 3 veces seguidas en la misma dificultad, esta disminuye una unidad y la *racha de derrotas* se reinicia a 0. Si la partida finaliza con una victoria, la dificultad aumenta un nivel. Si se superan 2 niveles seguidos, la dificultad aumenta 2 posiciones para experimentar un cambio más notable en las reglas del juego y para evitar el aburrimiento de los jugadores experimentados. La racha de victorias se reinicia.

Cada configuración de la dificultad resulta en un mapa con características diferentes, variando el número de habitaciones, el tamaño de la mazmorra y número de enemigos.

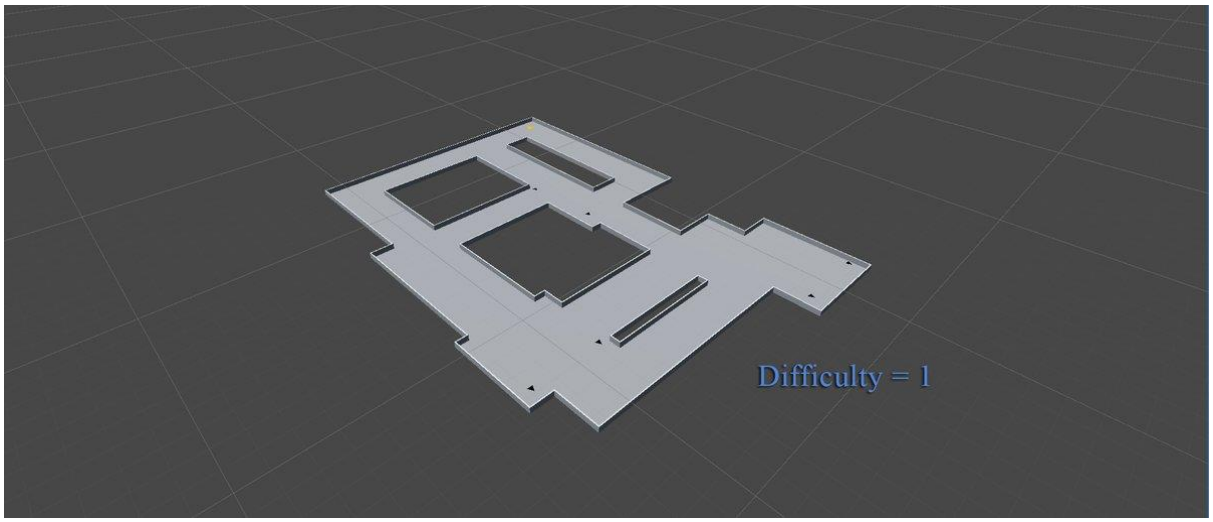

Ilustración 3.1. Mapa generado con dificultad = 1.

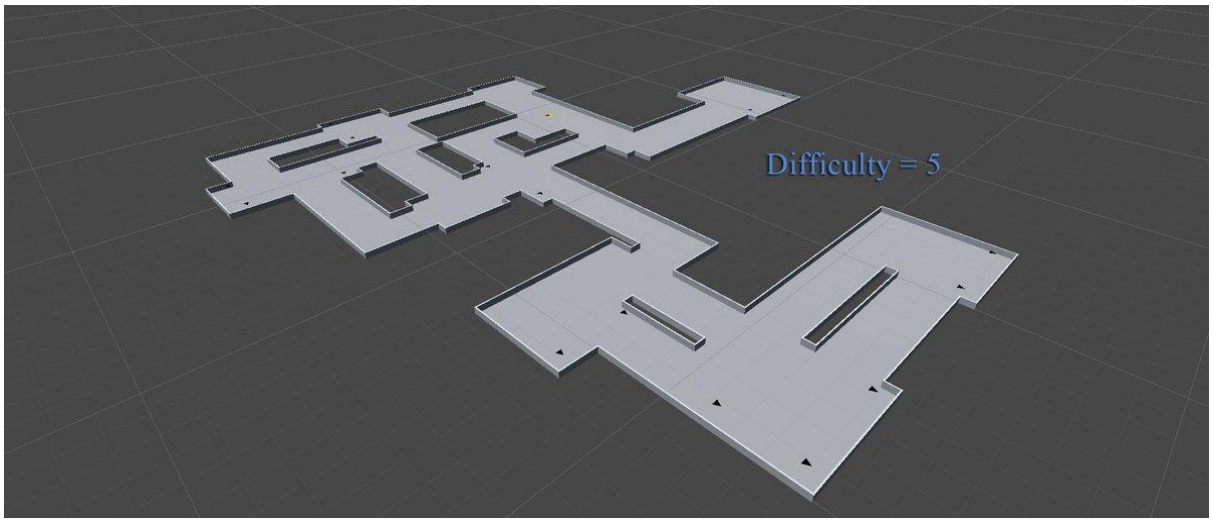

Ilustración 3.2. Mapa generado con dificultad = 5.

3.1.8 Fórmulas de la dificultad

Los parámetros que generan los mapas siguen unas fórmulas diseñadas a mano que tienen en común la participación de la variable *dificultad.*

En un juego como el propuesto, el espacio y los enemigos forman el núcleo de las mecánicas y, por tanto, del reto. Por ello, la mayoría de las fórmulas están enfocadas a estos dos apartados.

El valor asignado a la vida, tanto del jugador como de los enemigos, es de 100 siempre.

Para calcular el daño que hace el jugador se sigue la siguiente fórmula:

- Daño del jugador = $50 - dificultad * 2$. Así, a medida que la dificultad aumenta, el daño que hace sobre los enemigos es menor y requiere más disparos para derrotarlos.

Los parámetros de los enemigos son sencillos, así como su comportamiento implementado. Dada una mayor dificultad, estos se comportarán de manera más agresiva y su salud será mayor, pero su inteligencia no se ve afectada por esta variable. El daño de los enemigos y su velocidad aumentan con la dificultad.

- Daño de los enemigos = $10 + dificultad * 2$.
- Velocidad de los enemigos = $15 + dificultad * 2{,}5$. $Si\ dificultad > 6,\ velocidad = 25 + dificultad * 2$. Como referencia, la velocidad del jugador siempre es 25.

Atributos como la velocidad o la cadencia de disparo del jugador no se ven afectados por la dificultad, pues sería un modo de alterar la mecánica del juego y adaptarse continuamente a estos cambios produciría incomodidad sobre los jugadores.

Fórmulas relativas a la generación de la mazmorra:

- Número de enemigos por habitación = 4. $Si\ dificultad > 6,\ número\ de\ \frac{enemigos}{habitación} = 6 + Aleatorio(0,3)$
- Número máximo de enemigos totales = $10 + dificultad * 5$. $Si\ dificultad > 6,\ enemigos\ totales = 12 + dificultad * Aleatorio(4,7)$.
- Tamaño de la mazmorra = 30. $Si\ dificultad > 4,\ tamaño\ mazmorra = 60$.
- Número de habitaciones máximo = $2 + dificultad * 2$. $Si\ dificultad > 4,\ número\ de\ habitaciones\ máximo = 2 + dificultad * Aleatorio(2,4)$.
- Número de coleccionables máximo = $4 + dificultad/3$.

Estos valores se establecen con la intención de poder observar con pocas partidas cómo afectan los parámetros a la generación de niveles y, además, ver la reacción de los jugadores. Le otorgan una funcionalidad básica que es suficiente para esta temprana fase de testeo en un grupo reducido de personas.

3.1.9 Interfaz

Se recurre a elementos comúnmente utilizados en videojuegos para hacer fácilmente comprensible la información, como una barra para representar la salud.

El resto de la información se representa mediante texto, como los coleccionables, del modo *coleccionables* recogidos dividido *por coleccionables totales*.

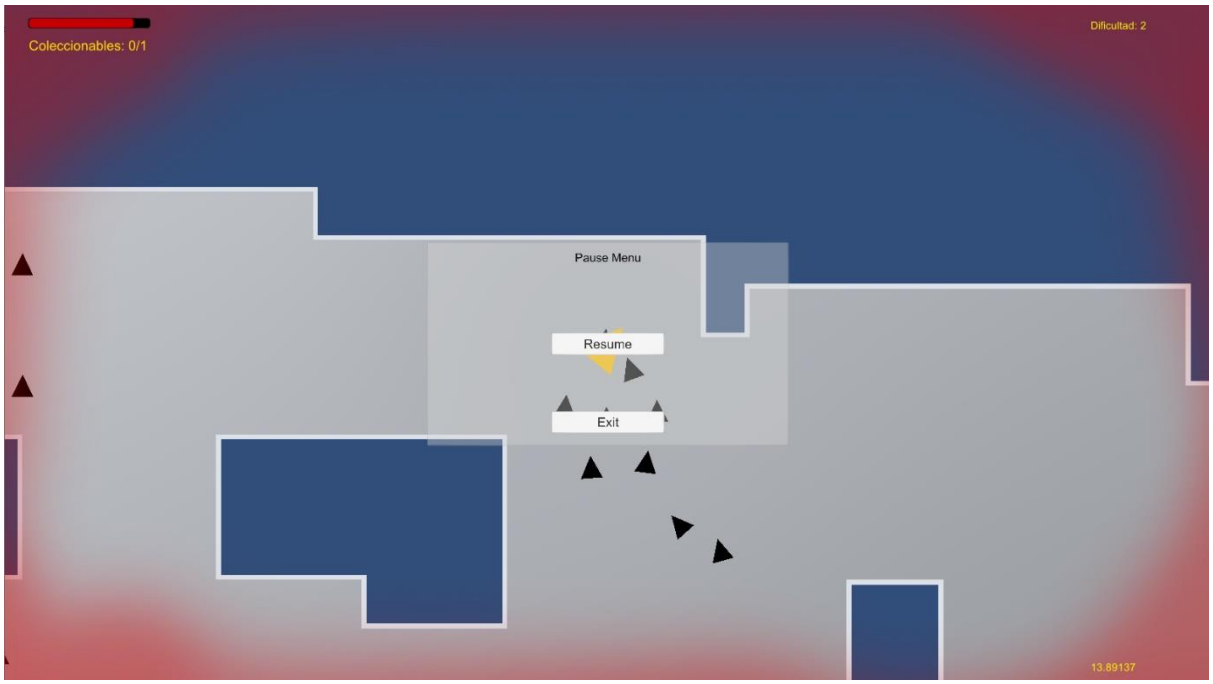

Ilustración 3.3. Captura in-game de la aplicación, primer prototipo.

En la esquina superior izquierda se muestra la barra de vida y un contador de coleccionables. En la esquina superior derecha, un texto muestra la dificultad actual del nivel. En la zona de la derecha, en la parte inferior, aparece un contador en segundos del tiempo total jugado.

Cuando el jugador recibe daño, la pantalla parpadea en rojo y la barra de vida disminuye proporcionalmente.

Además, integrándose con el escenario, aparece al inicio de los niveles un panel con una representación del teclado y el ratón explicando los controles.

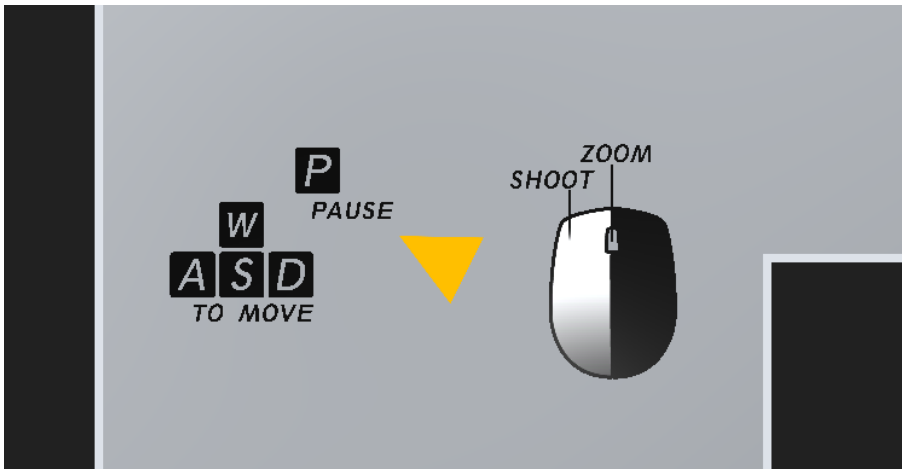

Ilustración 3.4 Panel de controles. Aparece en la posición inicial del jugador y permanece durante toda la ronda.

El menú de pausa se puede activar con la tecla ESC o la tecla P. Es sencillo y transparente, tiene dos botones que permiten reanudar el juego o salir del mismo.

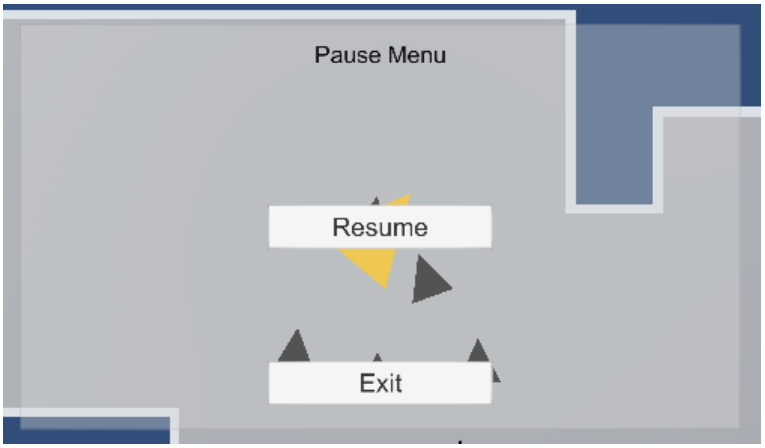

Ilustración 3.5. Captura de la primera versión del menú de pausa. Cuenta con dos botones, uno para reanudar la partida y otro para cerrar la aplicación.

3.1.10 Pruebas preliminares de efectividad y rendimiento del prototipo de juego

10 usuarios familiarizados con los videojuegos prueban esta primera aproximación del experimento para evaluar el funcionamiento de las mecánicas y su interfaz. La prueba se realiza en persona, por lo que se puede observar a los usuarios jugar y reaccionar. Se intenta no facilitar información para analizar si el objetivo de la aplicación se entiende por sí solo. Se generan trazas, pero aún no se envían a un servidor, se almacenan en el ordenador alojador.

La mayoría de los usuarios cambiaban la configuración inicial del tamaño de cámara, ampliando el campo de visión. A raíz de casos extremos, donde algunos usuarios observan el mapa en su totalidad, se decide limitar el valor máximo del campo de visión.

La dificultad parecía escalar bien con las fórmulas, aunque a partir del nivel 7 la velocidad de los enemigos se volvía inestable, hasta el punto de que atravesaban muros, y los parámetros de daño y salud del jugador hacían la experiencia cercana a imposible de superar.

El objetivo del proyecto es generar niveles sin intervención humana, por lo que en la siguiente versión de este se eliminan las fórmulas que modelan la dificultad.

3.2 Diseño de la primera versión con recogida de datos mediante trazas para alimentar el modelo de inteligencia artificial

Tras el primer contacto con sujetos, se observan aspectos de la aplicación a mejorar, modificaciones que pueden hacer más accesible al público la aplicación, así como errores o características de dudosa comprensión que en esta etapa intentan corregirse.

3.2.1 Modificaciones en la jugabilidad

Puede darse el caso de que al empezar una partida el jugador esté rodeado por enemigos, en el prototipo estos se abalanzan sobre el jugador desde el primer momento, sin permitirle leer las instrucciones siquiera. Se implementa, por tanto, una funcionalidad para que los enemigos permanezcan inmóviles hasta que el jugador tome el control sobre la acción, es decir, presione una tecla o dispare.

También se añade un nuevo comportamiento a los enemigos. Cuando reciben un disparo, pierden vida y sufren un *tiempo de recuperación*, durante este tiempo, el enemigo permanece inmóvil y vulnerable. Sirve como indicador de que ha recibido un disparo, y además permite darle un respiro al jugador si se ve rodeado por enemigos.

3.2.2 Introducción de audio para mejorar la retroalimentación

Durante la sesión de prueba algunos usuarios echaban en falta efectos de sonido, o música de fondo. En el caso de los efectos, para tener cierta retroalimentación de sus acciones y, en el caso de la música, para percibir la presencia del jugador en el escenario, o incluso indicar que el juego está activo.

Se introdujo música para hacer el videojuego una experiencia más amena y además se le dio un uso jugable. El comportamiento del volumen de la música iba ligado a la muerte de los enemigos, de manera que el volumen aumentaba cuando el jugador los eliminaba, y el resto del tiempo disminuía progresivamente.

La configuración por defecto tiene desactivado el volumen de música y efectos. Se activa mediante un interruptor situado en el menú de opciones.

3.2.3 Corrección estética

El comportamiento y aspecto de los coleccionables fue modificado. Se elimina el comportamiento que los hace rotar sobre sí mismos para evitar llamar la atención del jugador. Pasan a ser unos rombos inanimados, para hacer la práctica lo más neutra posible.

3.2.4 Cambios en el modelo de dificultad

Se elimina la variable dificultad para moldear la experiencia. Los niveles son generados aleatoriamente, dentro de unos rangos. La dificultad del nivel recae en los parámetros de generación de este. De esta manera, con los resultados obtenidos tras la fase de pruebas por parte de los jugadores, se puede analizar qué dependencias existen entre las variables y hallar cuáles contribuyen a ajustar la dificultad midiendo su correlación.

Cada variable se genera aleatoriamente dentro de unos rangos:

- El daño que infringe el jugador se encuentra entre 25 y 50.
- El daño de los enemigos entre 10 y 30.
- La velocidad de los enemigos puede ser desde 15 hasta 40.
- La salud de los enemigos está comprendida entre 75 y 150.
- El tamaño de la zona de activación va de 40 a 70.
- El tiempo de recuperación es de 0,55 a 1,25.

Los atributos del jugador son fijos, de manera que la vida es siempre 100 y su velocidad 25.

Rangos de los parámetros de generación del mapa:

- El número máximo de enemigos por habitación va desde 1 a 8.
- El número de habitaciones máximas de 4 a 35.
- El número de enemigos máximo de 15 a 60.
- La cantidad de coleccionables máxima es de 4 a 10.
- El tamaño de la mazmorra desde 30 hasta 80

3.2.5 Inclusión de cuestionarios

Para poder conocer y relacionar la opinión del jugador con lo que está pasando en pantalla, se introducen una serie de formularios y cuestionarios que aparecen durante la sesión de juego, al iniciar o finalizar la ronda.

La aplicación cuenta con un formulario inicial obligatorio para recoger el nombre del jugador y asignarle un id (Ilustración 3.6 Formulario inicialIlustración 3.6)

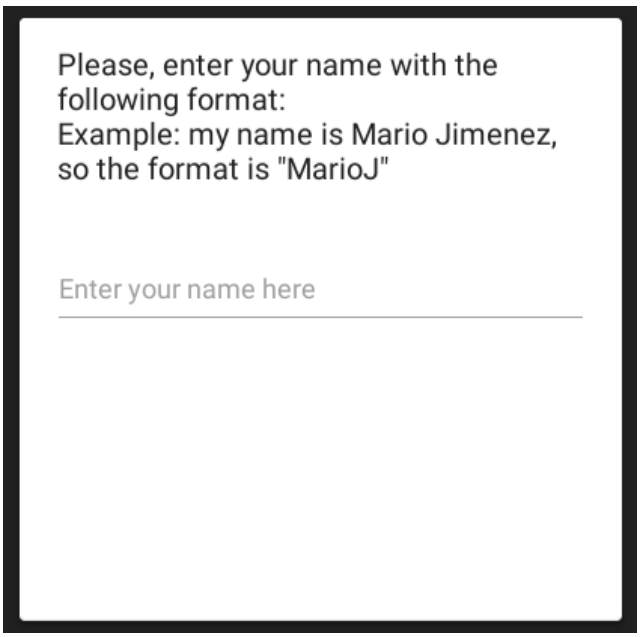

Ilustración 3.6 Formulario inicial con el que abre el videojuego. El usuario ha de introducir su nombre y apellido de la forma "NombreA"

Para poder colmar la máquina de soporte vectorial de datos, se graba todo lo que el jugador hace en un archivo de información ordenada. Se almacenan variables como los enemigos derrotados, tiempo de ronda, precisión o victoria/derrota. Además, esta información va acompañada de las respuestas a un cuestionario de dos preguntas que se muestra entre rondas. Estas dos preguntas cuestionan, cómo de divertido ha sido el nivel y su grado de dificultad, siendo la primera de tipo sí o no, y la segunda una respuesta numérica, con valor comprendido entre el 1 y el 5.

El cuestionario se utiliza para posteriormente relacionar esta información con los datos almacenados acerca de la generación del nivel, que sirven a la inteligencia artificial para generar casos de características similares.

El botón para avanzar de nivel permanece oculto hasta que se interactúa con la segunda pregunta. De este modo, resulta imposible evitarla.

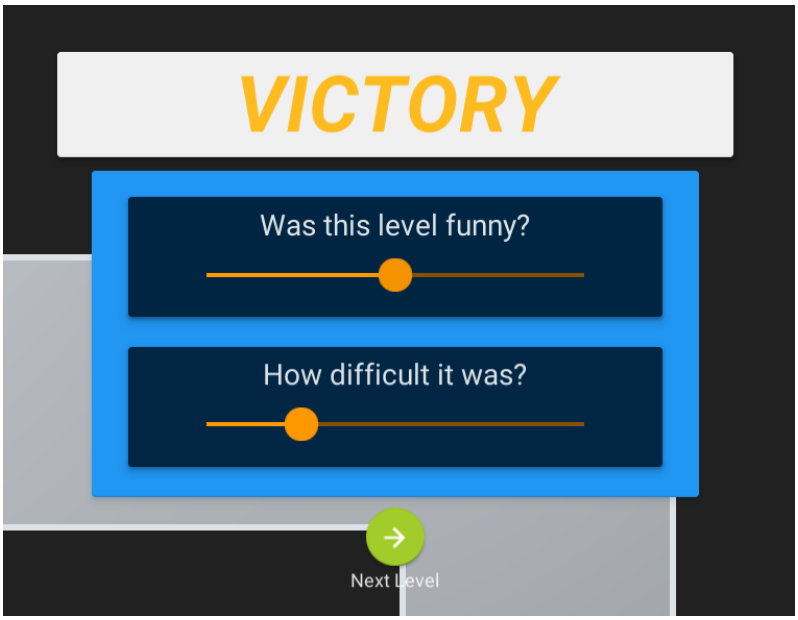

Ilustración 3.7. Segunda versión del cuestionario. Contiene 2 preguntas acerca de la diversión y la dificultad del nivel. Las respuestas son numéricas, en un rango del 1 al 5, siendo 1 el valor más bajo y 5 el más alto.

3.2.6 Modificaciones de la interfaz

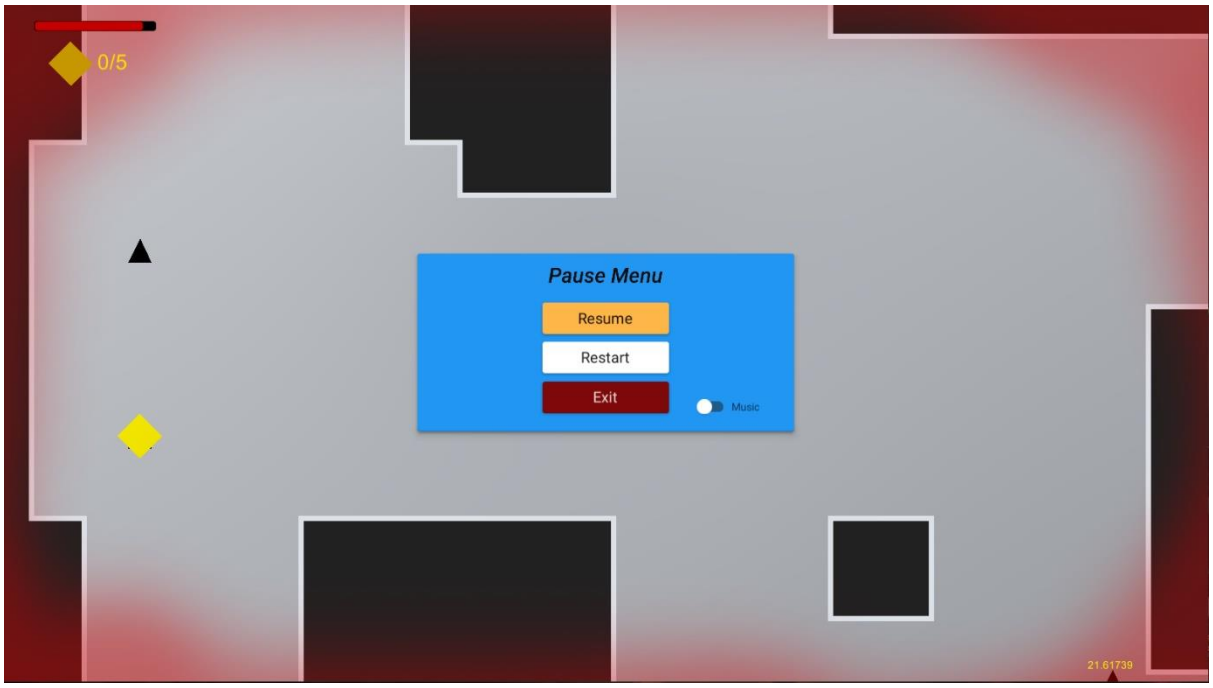

Ilustración 3.8. Elementos de la interfaz general. Primera versión del proyecto.

En la esquina superior izquierda se encuentran la barra de vida y un contador de coleccionables, se ha sustituido el texto “coleccionables” por un icono de estos. En la esquina derecha inferior aparece un contador en segundos del tiempo total jugado.

Cuando el jugador recibe daño, la pantalla parpadea en rojo y la barra de vida disminuye proporcionalmente.

El texto de coleccionables ha sido sustituido por el icono del coleccionable. Se muestra también la cantidad de coleccionables recogidos, dividida por los totales.

Los elementos de la interfaz sufren un lavado de cara, se utiliza Material UI (InvexGames, s.f.), unos recursos en fase beta de Google. Se sustituyen los botones y resulta útil para realizar cuestionarios, pues introduce sliders, campos para introducir texto, botones radiales y, además, mejora su estética.

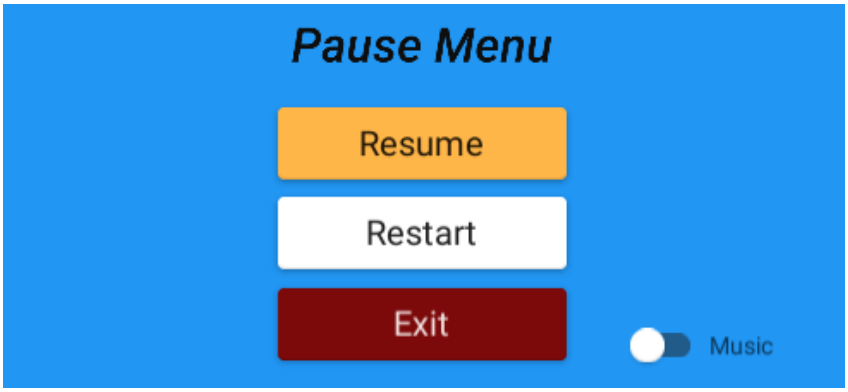

Ilustración 3.9 Menú de pausa rediseñado. Contiene 3 botones, cada uno de un color para diferenciarlo mejor. Posibilitan reanudar la partida, reiniciar el nivel, o salir del juego. Además, se incluye un interruptor para encender o apagar la música del juego, por defecto apagada.

El menú de pausa muestra 3 botones. Respecto a la anterior versión, se ha añadido el botón de reiniciar, que permanece oculto hasta pasados 15 segundos de una ronda. Resulta útil para generar un nuevo nivel, en caso de estar en uno con habitaciones inconexas y, por tanto, de imposible solución. También se le añade un interruptor para activar y desactivar el sonido.

3.2.7 Pruebas con usuarios. Impresiones sobre la jugabilidad

El comportamiento añadido a la música hace que el videojuego sea más divertido, pero pierde su intención inicial, servir como experimento. Si se quiere obtener un entorno imparcial y aséptico, se debe eliminar.

Asimismo, los jugadores destacan dos fallos más importantes: cuando los enemigos van muy rápido atraviesan las paredes, concretamente, las esquinas; y el jugador se mueve más rápido en diagonal, pues se suma la fuerza vertical y horizontal del movimiento.

También se dan casos en los que los jugadores premian lo bien que adapta el sistema la dificultad o, por el contrario, usuarios que se quejan de que sus respuestas a los cuestionarios no están modificando el comportamiento del sistema de inteligencia artificial. En ambos casos, son fruto de una malinterpretación, pues en esta fase del desarrollo la generación no se adapta al jugador, sino que se realiza aleatoriamente.

3.2.8 Captura de datos para análisis

Al mismo tiempo que se ejecuta la aplicación se almacenan trazas en un archivo de tipo JSON con información categorizada en:

- Entrada, datos recogidos por el teclado y el ratón, referentes al movimiento y a los disparos del jugador.
- Sistema, información sobre los procesos y fallos en las funciones.
- *Setup*, trazas con relación a la configuración del juego, como activar la música o cambiar el tamaño de la cámara.
- Físicas, información relativa a colisiones. Registra cuando una bola colisiona con un enemigo, o cuando un enemigo golpea al jugador. Cuando un enemigo es derrotado, actualiza la variable que guarda el número de derrotados total y se hace un recuento general de los que quedan en pie.
- *Quiz*, sirve para almacenar las repuestas al cuestionario entre niveles.
- Datos, categoría que recoge los parámetros con los que se ha generado el nivel, como el número de habitaciones y enemigos. También recoge el rendimiento del jugador, como los disparos realizados y los disparos acertados. Estos datos son los que se utilizan en las tablas, junto con las respuestas al cuestionario.

3.2.8.1 Creación de tablas para un sistema de aprendizaje automático

Se decide utilizar un modelo de aprendizaje automático como núcleo de la IA que diseñe la dificultad de la aplicación. Para entrenarlo, se recoge el mayor número posible de datos durante las partidas. Se generan unas tablas en las que cada columna es una variable implicada en la generación de niveles, como el número de habitaciones, el número de enemigos o la cantidad de vida del usuario. Posteriormente, se evalúa si existe correlación entre estas columnas. Las variables que muestren poca correlación pueden ser descartadas.

3.2.8.2 Despliegue del servidor para almacenar trazas

Durante la ronda se recogen trazas de información, que se serializan en un archivo JSON para ser enviado al servidor al finalizar la ronda. Las trazas admiten diferentes categorías, de manera que por cada usuario se generan varias listas, pertenecientes a las físicas, al daño de los enemigos u otra para los parámetros de generación del mapa. En la implementación, cada traza es una lista de un tipo propio compuesto por un encabezado, de tipo string, que da nombre a la categoría, y un formulario web, de WWWForm, para almacenar la información. Cada usuario es identificado con un id único.

Se utiliza el modelo REST-CLIENT para la comunicación con el servidor. El cliente, programado en Unity, hace uso de UnityWebRequest para envíar POST al servidor al final de cada ronda con toda la información almacenada durante esta.

Para el host se utiliza json-server (typicode, 2013), una API falsa que da la funcionalidad REST (Representational State Transfer) para la comunicación entre máquinas mediante HTTP, sin necesidad de programar nuestra propia API.

Las API falsas emulan el comportamiento de una API verdadera, pero se utilizan normalmente en estado de desarrollo o prototipado porque resultan útiles para realizar pruebas localmente. Del mismo modo, permiten su alojamiento en web.

Json-server se encarga de recibir los paquetes y darles un id. Permite la concurrencia, está diseñada para prototipar debido a su sencillez, pero cumple los requisitos para este proyecto y es fácil de usar.

Se utilizaron las versiones de escritorio para Windows y Linux como primera aproximación. En las siguientes versiones se pretende sacar una versión WebGL, jugable desde el navegador, que requiere alojar json-server en una dirección https para poder realizar llamadas entre páginas web.

3.2.8.3 Análisis de datos

La publicación del videojuego se comunicó a través de Twitter. 675 personas descargaron el juego, de las cuales 327 jugaron. La media de partidas jugadas es de 6.44, con una duración por ronda de 59.54 segundos de media. El porcentaje de victorias es del 88.18%. Por último, la precisión de disparo es del 37.72% de media.

Se obtienen los datos relevantes para las tablas y las respuestas a los cuestionarios establecidos entre rondas se utilizan como *“etiqueta”* para el entrenamiento, esto es, diversión y dificultad.

Surge un problema relativo a las respuestas de estos cuestionarios, pues se malinterpreta la prueba. Algunos usuarios indican que, por muy bajas que pusieran las notas al nivel, el siguiente seguía siendo fácil, o viceversa, cuando se especificaba que la generación era completamente aleatoria.

Después de tratar los datos con R, un software dedicado a operaciones estadísticas principalmente, se obtienen estas distribuciones en los cuestionarios:

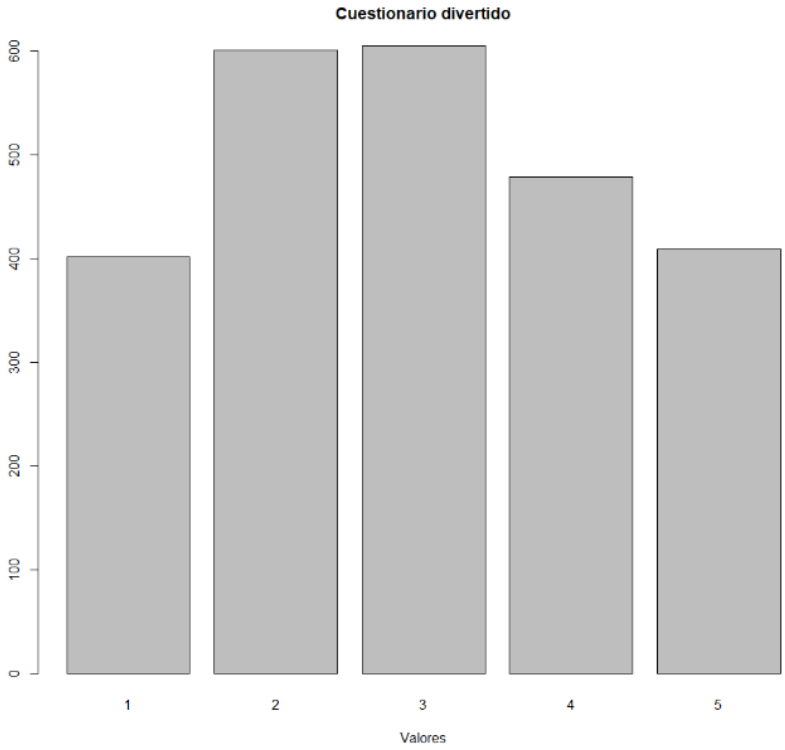

Ilustración 3.10 Distribución de diversión. En la gráfica se muestran las respuestas de los jugadores a las rondas jugadas, valorando su diversión del 1 al 5. La barra más veces votada es la 3, el término medio.

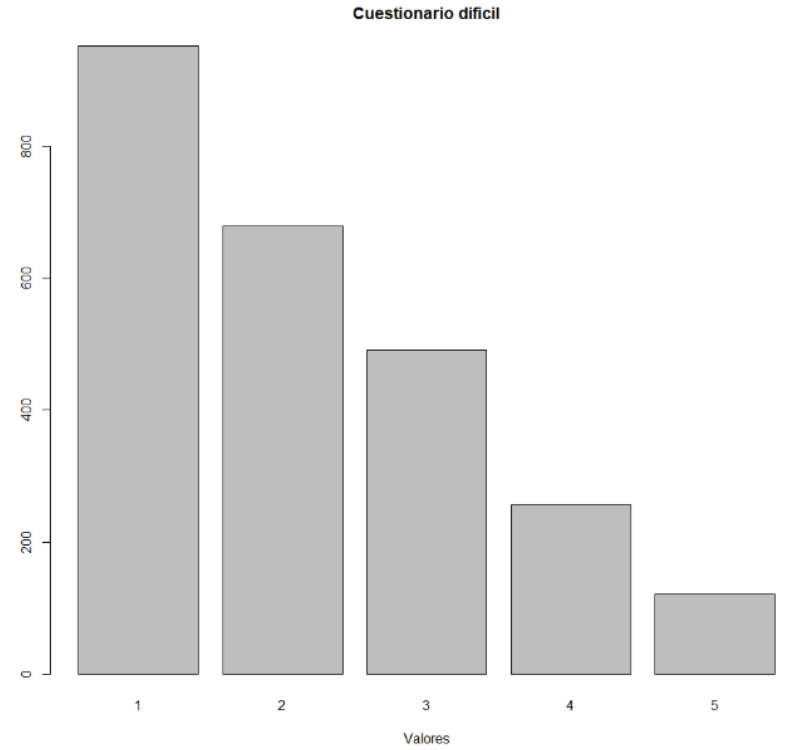

Ilustración 3.11 Distribución de dificultad. En la gráfica se muestran las respuestas de los jugadores a las rondas jugadas, valorando su dificultad del 1 al 5. La barra más veces votada es la 1, por lo que el juego resulta muy fácil.

3.2.8.4 Creación de programa para interpretar el archivo de trazas

Se crea un proyecto aparte programado en C# para procesar la información del servidor. El proyecto obtiene el archivo de información mediante una llamada GET al servidor.

Se trabaja sobre el JSON obtenido y se itera sobre cada traza para obtener la información útil de cada una de estas, eliminar residuos, disminuir el tamaño del archivo, y transformar los nombres de usuario en valores numéricos. De esta forma, se genera un nuevo archivo CSV (Comma Separated Values) para su posterior tratamiento de los datos en R Studio.

Este proyecto cuenta con dos archivos principalmente, Base.cs, en el que se define un objeto cuyos atributos detallan el contenido de una traza, es decir, un id, nombre de usuario, lista de datos de mapa y el código del programa principal.

En el programa principal, mediante una llamada GET al servidor donde se envían los resultados de las partidas, se obtiene el JSON generado para su posterior análisis. Iterando sobre cada traza y mediante la deserialización, se consigue un objeto de tipo Base. De este modo, cuando se termina de procesar el archivo, se obtiene una lista de objetos de tipo Base.

A continuación, los datos de cada objeto se escriben siguiendo la forma de un archivo CSV, de manera que cada variable de *datos y quiz* se transforma en una columna de la tabla y cada partida es una fila.

Para la deserialización del archivo json se emplea JSON.Net (Newtonsoft, s.f.)

3.2.8.5 Búsqueda de correlaciones entre las variables

Una vez los datos han sido almacenados en un CSV y posteriormente procesados y cribados, se buscan correlaciones entre las variables con el software estadístico R (r-project.org, s.f.).

Mediante la extensión Ggally se dibujan las matrices de correlación. Estas matrices muestran la interrelación entre las variables de generación del nivel formando un triángulo, de manera que, a la altura que aparece una variable, la fila o columna que forman en la gráfica muestra una serie de valores, comprendidos entre -1 y 1, que indican la cantidad de correlación entre esa variable con el resto. Para poder afirmar que existe correlación entre dos variables, el valor absoluto de esta debe ser al menos de 0.7 para evitar confundirlo con correlaciones fortuitas o inintencionadas. Así mismo, el signo de la correlación indica si esta es proporcional o inversamente proporcional.

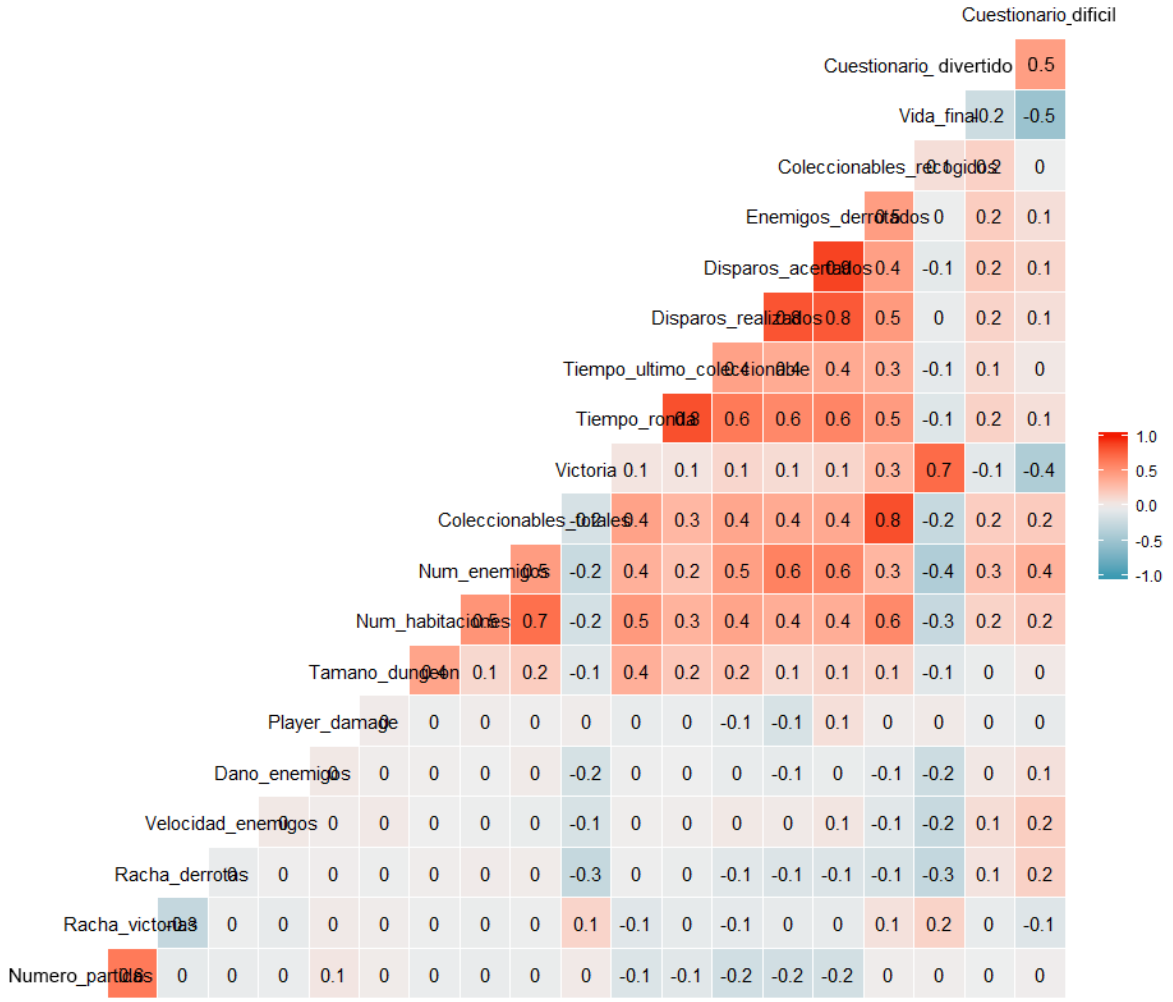

Ilustración 3.12 Gráfica de correlaciones. El rango de valores posibles está comprendido entre -1 y 1. La gráfica recoge la relación entre variables y se expone en cada cuadro, siendo relevantes aquellos cuyo valor absoluto sea igual o mayor a 0.7. Las dos últimas columnas muestran las correlaciones de los cuestionarios con el resto de las variables: Cuestionario_divertido y Cuestionario_dificil.

Con los resultados se puede observar que no había una fuerte correlación entre los cuestionarios, situados en las dos últimas columnas, con ninguna del resto de las variables. Las correlaciones que aparecen son bastante evidentes e irrelevantes, como la relación entre

Disparos acertados y Enemigos derrotados o *Coleccionables totales y Coleccionables recogidos*, que eran correlaciones que se esperaban a priori.

Se siguen trabajando los datos para analizar las valoraciones proporcionadas en los cuestionarios a nivel de ronda, en lugar de manera general, como ocurre en el gráfico anterior, para intentar encontrar cierta progresión en las respuestas ofrecidas por parte del jugador a medida que avanzaba en las partidas. Para esto se tienen en cuenta los valores de *Cuestionario difícil y Cuestionario divertido* por separado y de manera progresiva, por rondas, generando nuevas columnas en la matriz.

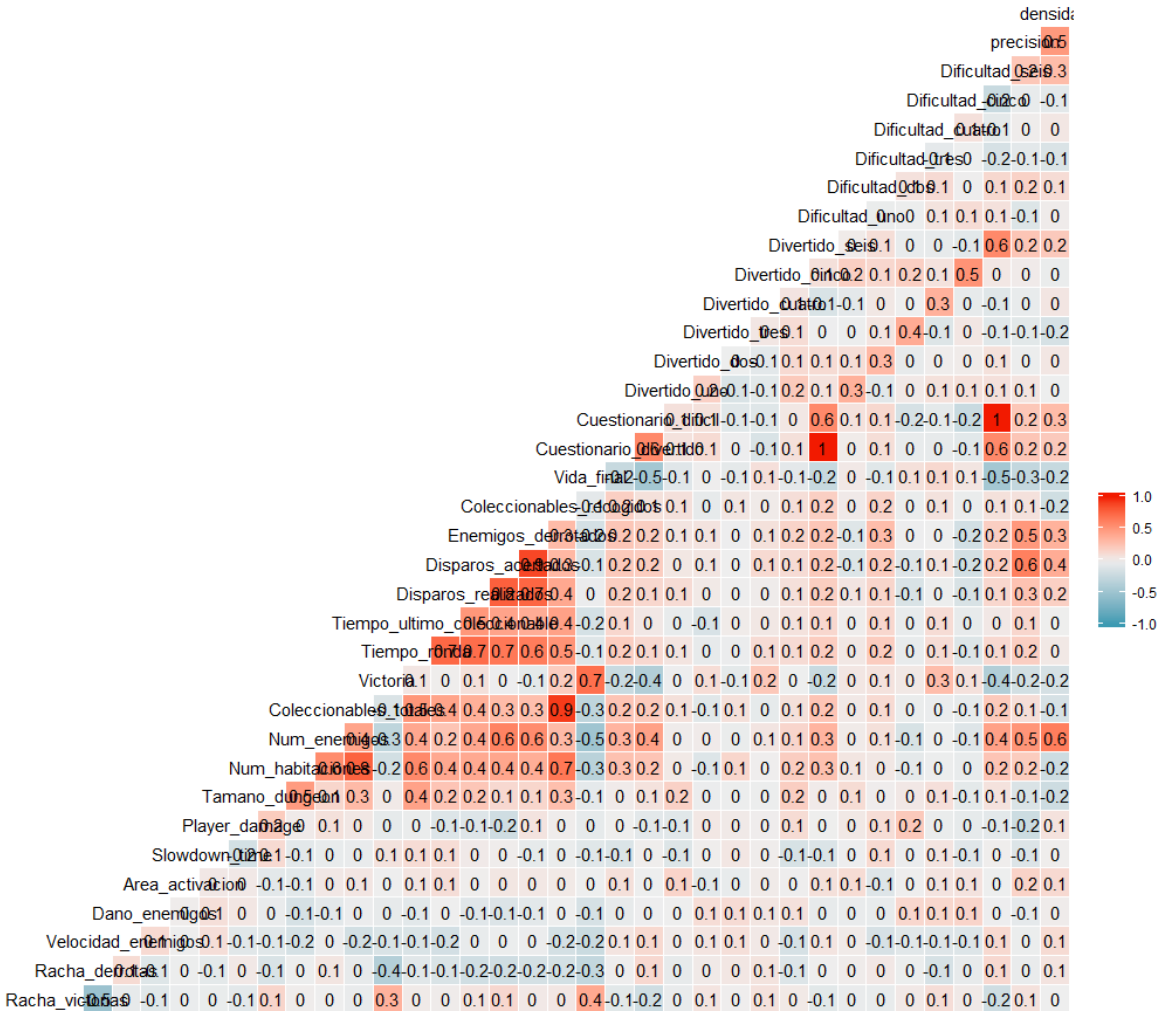

Ilustración 3.13: Gráfica de correlaciones con partidas consecutivas. Se expone cada partida como 2 columnas, una para la diversión y otra para la dificultad, un total de 6 partidas por usuario.

Las variables *Dificultad_uno, Dificultad_dos, etc. y Divertido_uno, dos...* Se refieren a las respuestas que dieron los usuarios a los cuestionarios conforme avanzaban las rondas, en este caso, se recoge el público que había jugado al menos 6 partidas.

Los resultados siguen sin mostrar fuertes correlaciones, por lo que se opta por buscar un nuevo enfoque para el experimento.

Estos resultados evidencian que programar una IA que genere niveles divertidos y con una curva de aprendizaje justa es una tarea que abarca muchos campos y es muy complicada. Es una tarea que pretende eludir la participación de diseñadores en un videojuego y sustituirlos por una IA. Irónicamente, se necesitaría un equipo de profesionales del campo del diseño, la psicología y la informática para llevar a cabo una hazaña de estas dimensiones. El juego necesitaría un diseño más profundo, más usuarios para pruebas, más tiempo y un cuestionario más completo, que recogiese aspectos de la personalidad del usuario e hiciese un seguimiento más exhaustivo de su rendimiento y opinión.

Además, para el propio usuario es complicado evaluar si una experiencia es divertida, ya que la pregunta es vaga y carece de contexto. Y al mismo tiempo, la diversión es subjetiva y difiere mucho entre jugadores.

Por eso, aunque el proyecto pierda cierto valor, se decide enfocarlo al ajuste de dificultad de una manera más agresiva para obtener resultados más visibles y correlaciones más fuertes.

3.3 Segunda versión del proyecto

Resultado de la falta de correlación entre las variables del programa y los resultados obtenidos en los cuestionarios, se decide modificar el proyecto existente y enfocarlo exclusivamente al ajuste de la dificultad, por lo que se elimina la diversión del cuestionario entre rondas.

3.3.1 Modelo de dificultad. Cambios en la generación de variables

Uno de los fallos de la anterior versión recae en la generación aleatoria de las variables que moldean el nivel y los elementos de este. Esto se debe a que cada variable se genera aleatoriamente dentro de unos rangos, sin tener en cuenta el resto de las variables, de modo que puede dar lugar a niveles donde los enemigos tienen un comportamiento agresivo, difícil, y al mismo tiempo, el mapa es pequeño y simple, de fácil superación. Esta mezcla de parámetros fáciles con difíciles da como resultado niveles muy heterogéneos, pero poco consecuentes, que dan como resultado coeficientes de correlación muy bajos.

Por este motivo, en esta versión del proyecto se opta por generar un número aleatorio del 0 al 2, dando lugar a 3 posibles configuraciones de dificultad, siendo 0 la más fácil y 2 la más difícil. A partir de esta dificultad, se escogen los valores del resto de parámetros del mapa, del comportamiento de los enemigos y de los atributos del jugador.

3.3.2 Modificación de las trazas de información y de la configuración

En la primera aproximación del experimento se optó por exportar el máximo de información posible. Como tras esta primera iteración información como la entrada, las físicas o la configuración no se habían utilizado, se decidió exportar solamente los datos y el quiz, que era con lo que se estaba trabajando en R.

Para asegurar que todos los usuarios utilicen la misma configuración del entorno se elimina la posibilidad de activar la música del juego. Se establece una distancia y tamaño de cámara fijos, sin posibilidad de modificarlos con la rueda del ratón.

3.3.3 Modificaciones en la jugabilidad

Se cambia el script de movimiento del jugador para arreglar el movimiento en diagonal y que no se sume la fuerza horizontal con la vertical, como ocurre en la primera versión.

En el primer experimento se observa que los jugadores disparan pulsando repetidas veces el clic del ratón en lugar de mantenerlo pulsado. Por este motivo, en esta versión del experimento se decide fijar el disparo como manual, descartando la variable *velocidad de fuego máxima* que había en la anterior versión. De este modo, es la mano del jugador la que decide la cadencia de disparo. Esto reduce la accesibilidad del proyecto, aunque es durante una fase breve del desarrollo.

3.3.4 Cambio de API para el almacenamiento de trazas

Para esta versión, se cambia además de API REST. Se utiliza *json-serverless* (pharindoko, s.f.), una extensión de *json*-server (typicode, 2013), que utiliza AWS (Amazon Web Services) para

alojar su funcionalidad y dirigir a este servicio las trazas enviadas por los usuarios. Al utilizar https, se puede enviar las trazas desde otra web. Se decide hacer una build en WebGL para poder jugar desde el navegador y se aloja en itch.io, una plataforma para desarrolladores de videojuegos de carácter independiente.

Como es una extensión de json-server, se mantiene el mismo formato para las trazas y la implementación no varía.

3.3.5 Refinamiento de los cuestionarios

El formulario inicial ha sido modificado, se le ha añadido un campo para la edad y otro para indicar si el usuario es jugador habitual, con el fin de conocer mejor a los usuarios y dar explicación a los datos obtenidos. En esta versión también se recoge el número de pasillos que unen las habitaciones, el número de enemigos derrotados por minuto y se cuentan los golpes recibidos, en lugar de la vida al final de la ronda.

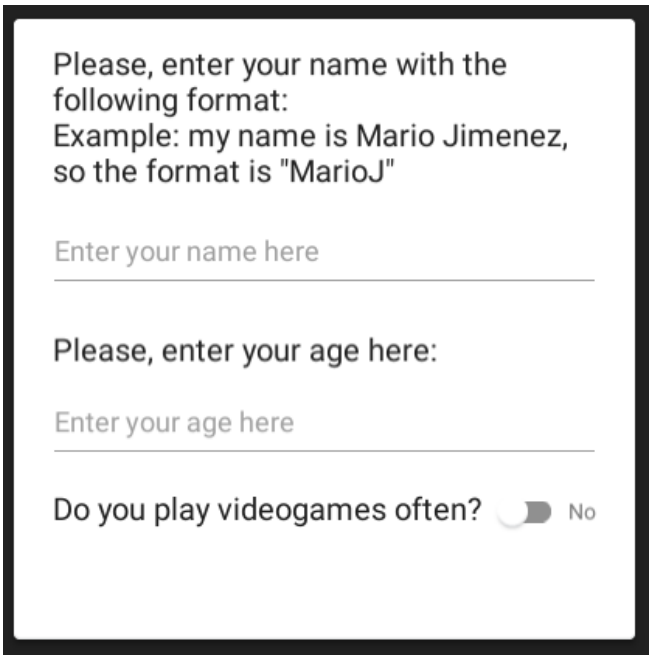

Ilustración 3.14: Formulario inicial, segunda versión del proyecto. Ahora se recoge información acerca de la edad del usuario y de su familiaridad con los videojuegos.

También se ha reformado el cuestionario, ahora solo se pregunta por la dificultad, con valores comprendidos entre el 1 y el 5.

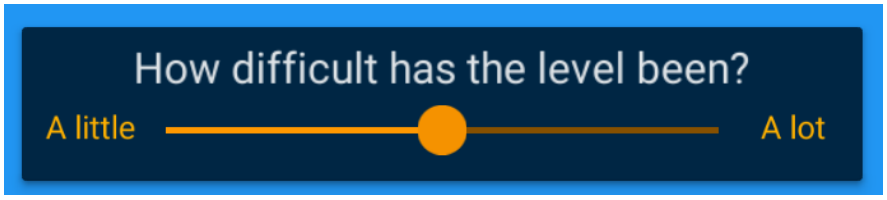

Ilustración 3.15: Cuestionario modificado, segunda versión del proyecto. Se retira la pregunta acerca de la diversión del nivel y permanece la de la dificultad. Se sigue respondiendo con valores de entre 1 y 5.

3.3.6 Análisis de datos

Esta vez el número de usuarios es 67. La media de edad de los jugadores es de 24.57 y la mediana 24, siendo 16 la edad mínima registrada y 41 la máxima. El 92% se consideran jugadores habituales.

La media de partidas jugadas es 4.59, la mediana 3. La dificultad media es 1, con un 65% de victorias y un tiempo medio de ronda de 47.19 segundos, 28.56 como mediana. La precisión tiene un valor similar a la del anterior experimento, 40.77%, pese a haber cambiado la mecánica de disparo. Se han recibido 1.7 golpes de media por ronda.

En el análisis de datos se observa que la valoración de los jugadores parece coincidir con la dificultad establecida por el juego, siendo la correlación del cuestionario y esta de 0.8 y el p-value < 0.000.

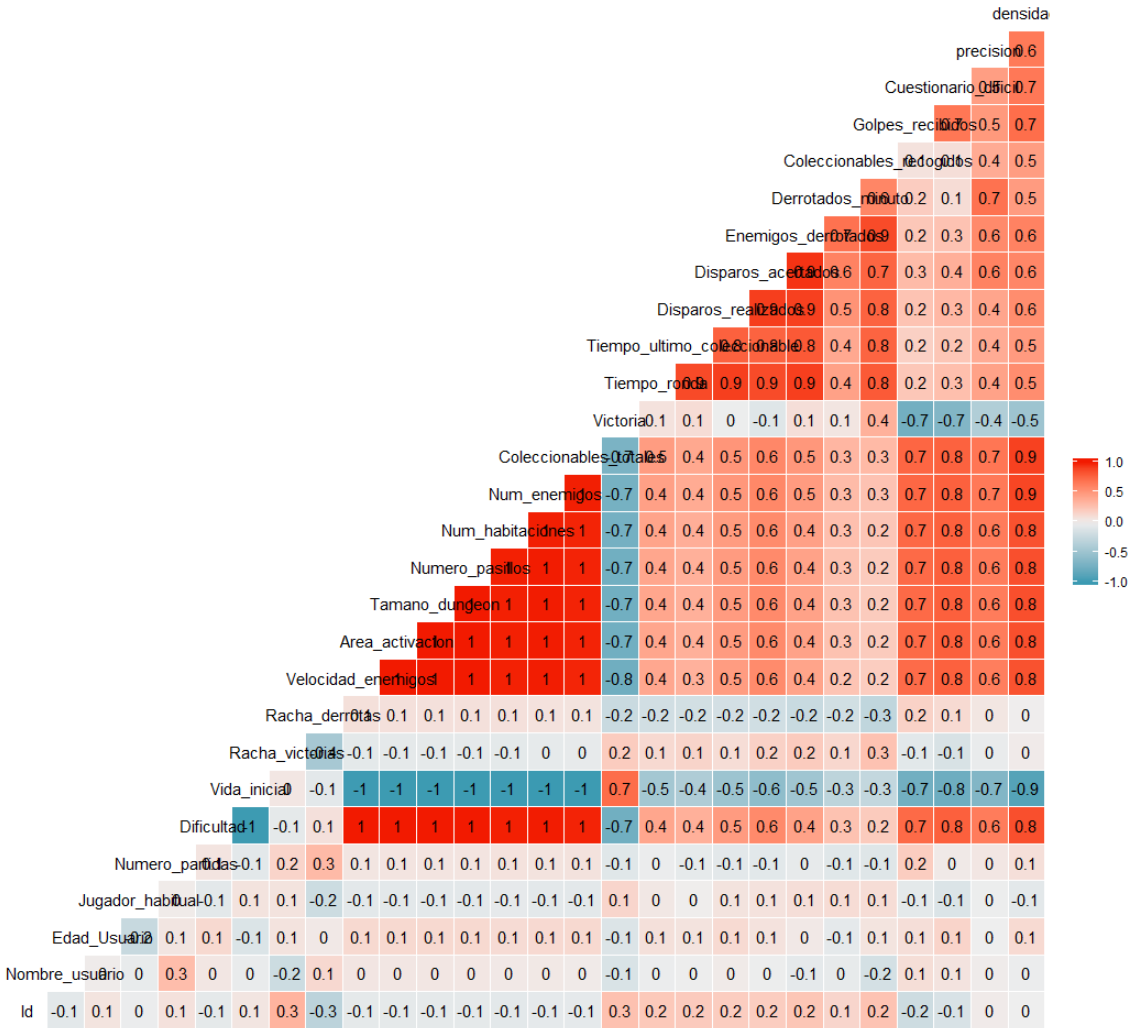

Ilustración 3.16: Gráfica de correlaciones, segunda versión del proyecto. Las correlaciones entre las variables que generan el nivel son mucho mayores respecto a la primera versión. La antepenúltima columna (Cuestionario_difícil) recoge las correlaciones entre las respuestas al cuestionario y el resto de las variables, con valores de 0.8 en los parámetros que modelan la generación del nivel.

Las correlaciones positivas se muestran con un color rojo fuerte, mientras que las negativas se representan con color azul.

Esta versión del experimento muestra correlaciones lo suficientemente fuertes como para comenzar a entrenar un sistema de aprendizaje automático capaz de predecir la dificultad que estimaría un usuario, dados los parámetros para generar la partida.

3.4 Tercera versión del proyecto

Una vez tratados los datos, se llevan a cabo una serie de pruebas para escoger el algoritmo de inteligencia artificial que realice mejores predicciones. Se ensaya con algoritmos de random forest (árboles de decisión) y también con un sistema de máquina de soporte vectorial. El primer algoritmo otorga porcentajes de acierto de entre el 54% y el 57%, dependiendo de la ejecución, mientras que la máquina de soporte vectorial logra crear un modelo del comportamiento de las respuestas a los cuestionarios con un 64.40% de éxito, utilizando un kernel lineal.

Por lo tanto, se opta por utilizar una máquina de soporte vectorial para realizar generalizaciones sobre los datos y poder elaborar una fórmula con la que generar mapas que sigan una dificultad progresiva.

El planteamiento es el siguiente: dada una dificultad D, se busca un conjunto de mapas entre las predicciones del modelo que cumplan con esta configuración y se escoge uno al azar.

Para esto se emplea la librería LibSVM en el proyecto de R. De un total de 280 partidas jugadas por 67 usuarios, se "barajan" los datos para que la distribución sea aleatoria y se divide la muestra en dos porciones, una con el 75% de los datos, que sirve para entrenar la máquina, y otra con el 25% restante, que sirve para evaluar las predicciones realizadas.

Mediante un kernel lineal, se realizan predicciones para averiguar, dado un conjunto de parámetros del mapa y datos sobre el usuario, qué dificultad estima el usuario que tiene dicho mapa. El hiperplano resultante tiene un porcentaje de aciertos del 64.40% a la hora de modelar el comportamiento de las respuestas al cuestionario.

Para incluir el modelo SVM en el juego se crea una fórmula que modela el comportamiento de la variable dificultad, que tiene un rango de valores de 1 a 5. Esta variable se pasará por parámetro a la SVM para que encuentre un mapa dentro del set de datos que cumpla con los requisitos. Como en la anterior versión del proyecto solo había tres configuraciones posibles de dificultad, las dificultades posibles que ofrece el set de datos son drásticamente diferentes entre sí, por lo que, para modular el valor de la dificultad, se crea una variable extra llamada posición, que puede tomar 3 posibles valores: 1, 2 y 3. La configuración inicial al comenzar una sesión de juego es 1 como valor de posición, y 2 como valor de dificultad.

La fórmula de la dificultad evalua en primer lugar si la última ronda ha resultado victoriosa.

- En caso afirmativo, se incrementa en una unidad el valor de la dificultad. Además, si el jugador lleva una racha de victorias divisible entre 3, la posición aumenta un valor más. Si el usuario indica que el nivel ha sido fácil, es decir, un 1 ó 2 en el cuestionario, la dificultad aumenta una unidad extra.

- En caso negativo, la dificultad y la posición no se ven alteradas. La dificultad disminuye cuando el jugador lleva 2 partidas perdidas. La posición disminuirá cuando la racha de derrotas sea 3. Si el usuario indica que el nivel ha sido difícil, un valor de 4 ó 5 en el cuestionario, la dificultad disminuye una unidad. Además, si al dar esta respuesta su racha de derrotas era mayor o igual que 2, la posición también disminuye una posición.

Mediante estos dos posibles escenarios, se descartan respuestas al cuestionario poco coherentes como que suceda una derrota y el usuario vote el mapa como muy fácil. Se busca realizar ajustes de manera notable y rápida, con pocas partidas. Los jugadores pueden ser muy diferentes asi que es mejor comenzar con cambios bruscos y según nos aproximamos a averiguar qué tipo de jugador es realizar cambios más sutiles

De este modo, se modela de manera matemática qué es un mapa exigente en este contexto de videojuego.

3.4.1 Modificaciones

La tercera versión del juego mantiene las características jugables de la anterior versión, pero moldea la dificultad del juego con el modelo calculado gracias a la máquina de soporte vectorial.

Esta versión del proyecto no sufre cambios en la jugabilidad, ni en la interfaz o apartado visual. Se introduce un cuestionario de satisfacción que aparece después de jugar 5 partidas para preguntar al usuario por el ajuste de dificultad llevado a cabo por la inteligencia artificial.

Para poder evaluar las respuestas correctamente, un grupo de usuarios, aproximadamente un tercio, formará parte del grupo de control, este grupo no cuenta con ajuste automático de dificultad, si no que se establece aleatoriamente al comenzar la partida.

3.4.2 Análisis de resultados

Para evaluar el éxito del sistema se emplea una encuesta de satisfacción para los usuarios, que aparece tras jugar 5 partidas. Esta revela una mejora del 16% de la satisfacción de los usuarios sin intervención humana cuando se ajusta la dificultad del videojuego, frente a la versión con generación aleatoria. No es una mejora importante, pero es valiosa, ya que supone un resultado positivo.

La encuesta fue cumplimentada por un total de 72 usuarios divididos en dos grupos: los usuarios de control (23) y el resto de los usuarios (49).

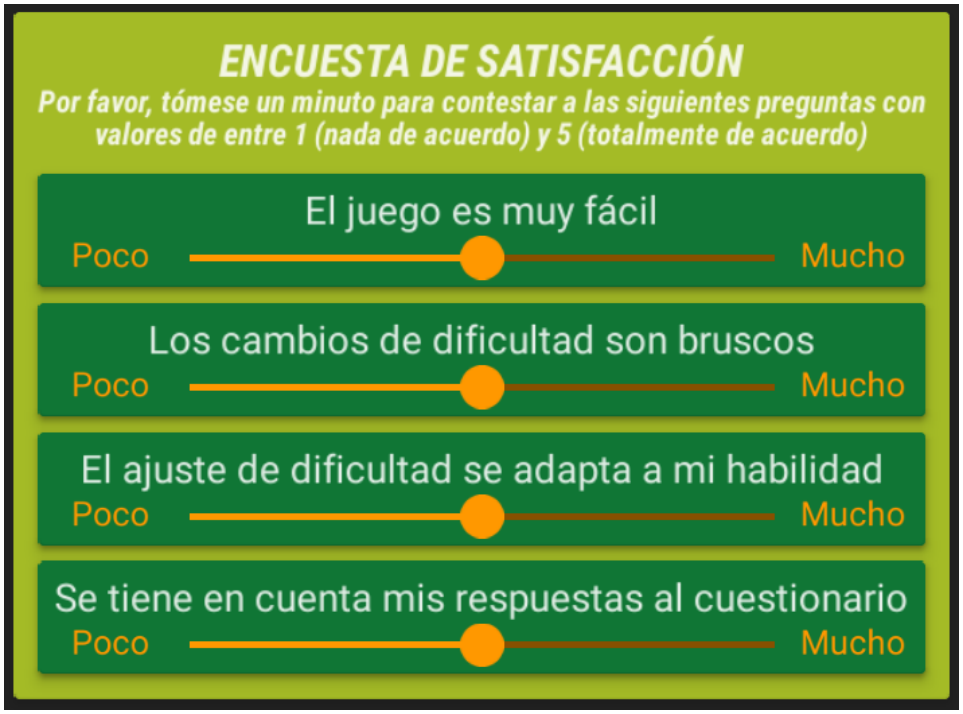

Ilustración 3.171 Encuesta de satisfacción. Aparece tras jugar 5 rondas al videojuego. Recoge información acerca de la dificultad general del juego, el ajuste de la dificultad, la sutileza de estos cambios y la importancia de las respuestas al cuestionario entre rondas.

Los resultados fueron los siguientes:

En el grupo de control:

- La primera pregunta tuvo una media de 2.91 y una mediana de 3.00
- En la segunda, la media fue de 3.56 y la mediana de 4.00
- La tercera pregunta obtuvo una media de 2.56 y una mediana de 2.00
- Por último, la cuarta pregunta tuvo una valoración de 3.08 de media y una mediana de 3.00

El resto de los usuarios puntuaron del siguiente modo:

- La primera pregunta obtuvo un 3.00 tanto de media como la mediana.
- La segunda puntuación reflejaba 3.36 de media y 4.00 de mediana.
- En la tercera pregunta la media es de 3.42 y la mediana es 4.00.
- Mientras, la cuarta pregunta posee un 3.95 de media y 4.00 como mediana.

Encontramos valoraciones muy similares en las dos primeras preguntas. Estas atienden cuestiones de jugabilidad general, como la impresión de la dificultad y la percepción de los cambios en la dificultad. No son las más relevantes para medir el rendimiento de la IA.

3.4.2.1 Encuesta de satisfacción: gráficas de las respuestas

Las mayores diferencias entre ambos grupos las encontramos en las respuestas proporcionadas en las preguntas 3 y 4, que se refieren a la adaptación por parte del sistema a las necesidades del usuario, y a la respuesta del sistema a las indicaciones del usuario en el cuestionario entre rondas, respectivamente. Además, estas preguntas resultan las más relevantes para medir el rendimiento del sistema puesto que se refieren a la capacidad de reacción de la inteligencia artificial frente a los estímulos proporcionados por usuarios.

En la tercera pregunta, el grupo de control da un resultado de 2.56 de media, mientras que el resto de los usuarios da 3.42. En el gráfico de cajas se observa como el resto de los usuarios concentra sus valoraciones entre el 3 y el 4, mientras que los usuarios del grupo de control se dispersan entre el 2 y el 3.5.

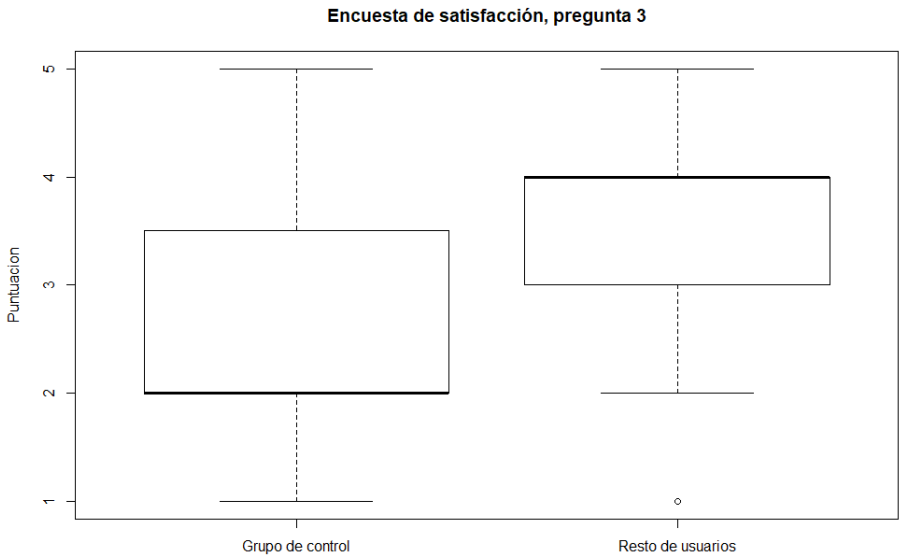

Ilustración 3.18 Diagrama de cajas y bigotes de la pregunta 3: "El ajuste de dificultad se adapta a mi habilidad". En el eje de la izquierda, la posible puntuación del 1 al 5. Cada caja representa un grupo de usuarios, el de la izquierda, el grupo de control, el de la derecha, el resto. En el primer grupo, el tercer cuartil llega a 3.5, mientras que, en el segundo grupo, el primer cuartil es 3, y el tercer cuartil, así como la mediana, son 4. Es una respuesta más positiva de media al comportamiento de la IA.

En la cuarta pregunta, la media del primer grupo es de 3.08 y del segundo grupo es de 3.95. La caja llega al 4 en el grupo de control, mientras que en la del resto de usuarios llega al 5.

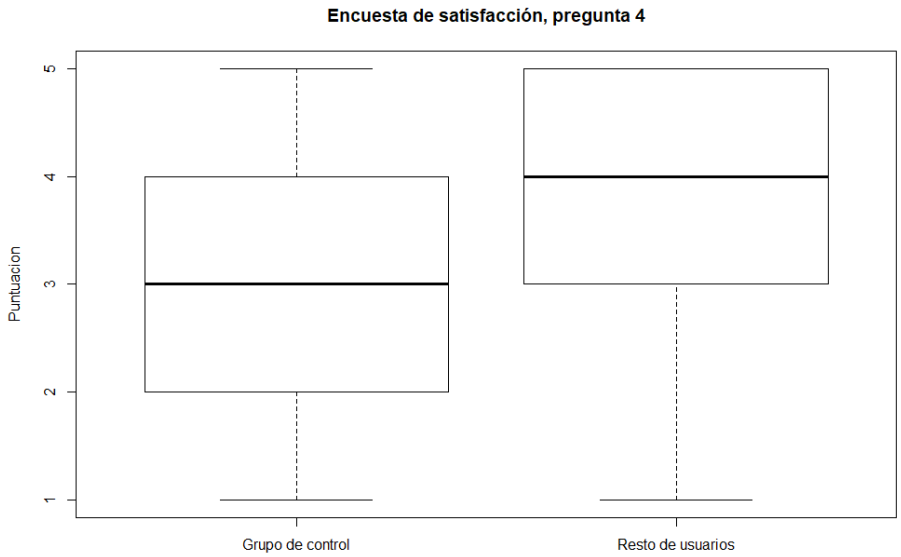

Ilustración 3.19 Diagrama de cajas y bigotes de la pregunta 4: "Se tiene en cuenta mis respuestas al cuestionario". En el eje de la izquierda, la posible puntuación del 1 al 5. Cada caja representa un grupo de usuarios, el de la izquierda, el grupo de control, el de la derecha, el resto. En el primer grupo, la mediana tiene un valor de 3, y su tercer cuartil llega al 4. En el resto de los usuarios, la mediana es de 4, y el tercer cuartil llega a 5. La respuesta es más positiva en el resto de usuarios que en el de control.

Si comparamos las medias obtenidas en cada pregunta, la mejora en la tercera pregunta es del 16%, y en la cuarta del 20%. El sistema demuestra ajustarse a la habilidad del usuario y reducir su frustración.

3.5 Cuarta versión del proyecto

Una vez probada la efectivad del sistema para ajustar la dificultad a la habilidad del jugador, se decide mejorar el proyecto para incrementar el porcentaje obtenido de las encuestas. Para ello, se refina el funcionamiento del proyecto, se introducen novedades jugables, como enemigos que disparan y se tienen en cuenta nuevos aspectos en la recogida de trazas de información durante las partidas.

3.5.1 Calidad del código

En esta versión se mejora el código existente, afinando el modo en que se tratan fallos. También mejora la estructura del código, las clases y el rendimiento. Ahora, en lugar de invocar y destruir objetos continuamente, como las balas, se implementan *pool* de objetos que activan y desactivan estos en función de las necesidades de la aplicación en tiempo de ejecución para mejorar el rendimiento.

3.5.2 Nuevas mecánicas jugables

Se modifica el comportamiento de los enemigos, refinando la manera en la que detectan al jugador. Además, se introduce una nueva habilidad, ahora los enemigos pueden disparar. De este modo, se introducen nuevas variables a tener en cuenta en la generación de niveles, como la cadencia de disparo enemiga, la velocidad de las balas y la distancia de detección. Los enemigos pueden desplazarse hacia el jugador o pueden dispararle, pero no las dos a la vez. En función de la dificultad en la que se encuentre el jugador, el porcentaje de enemigos disparadores variará.

Otra nueva característica del proyecto es que se modifica el tamaño del collider de los enemigos en función de la dificultad de la partida, siendo el doble del tamaño original en la dificultad más baja y el original en la más alta. De esta forma, se ayuda a mejorar los datos de precisión del jugador en dificultades bajas y puede derrotar enemigos más fácilmente.

3.5.3 Nuevas fórmulas de generación

Al haber introducido nuevos comportamientos a los enemigos, la IA tiene que ser entrenada de nuevo, por lo que se necesita una nueva ronda de recogida de datos de los jugadores, generando partidas aleatoriamente para ser evaluadas por los jugadores con el cuestionario (Ilustración 3.15).

La dificultad de cada partida es aleatoria a elegir entre 4 posibles dificultades: un número de 0 a 3, siendo 0 la dificultad más fácil y 3 la más difícil. Los parámetros de generación del nivel se escogen en función de esta dificultad. De esta manera obtenemos los siguientes posibles valores para las variables de generación de la partida:

- La vida del jugador es siempre la misma, independientemente de la dificultad: 5 golpes de vida.
- Velocidad de los enemigos: 20, 27, 30, 35.
- Vida de los enemigos: 2, 3, 4 y 5 golpes.
- Distancia de detección de los enemigos: 40, 50, 60 y 70.
- Tiempo de recuperación de los enemigos (tras un golpe): 1.25, 0.95, 0.75 y 0.7 segundos.
- Número máximo de enemigos por habitación: 3, 4, 6 y 8.
- Número de habitaciones máximas de la mazmorra: 9, 18, 27 y 34.
- Número máximo de enemigos total: 20, 32, 40 y 70.
- Número máximo de coleccionables total: 4, 4, 7 y 7.
- Tamaño de la mazmorra: 40, 55, 65 y 80.
- Número máximo de pasillos conectando las habitaciones: 2, 3, 3 y 4.
- Frecuencia de disparo de los enemigos (en segundos): 2, 2, 1.5 y 1.25.
- Velocidad de desplazamiento de las balas enemigas: 3.25, 4.1, 4.85 y 5.75.
- Tamaño del collider de los enemigos: 2.5, 1.75, 1.25, 1.
- Probabilidad de enemigos disparadores: 35, 30, 40 y 50.

3.5.4 Cambios en el cuestionario

Para tener un cuestionario lo más claro posible, se sustituyen los textos “Un poco” y “Mucho” del cuestionario (Ilustración 3.15) por “Muy fácil” y “Muy difícil”. De nuevo las respuestas son números enteros entre 1 y 5.

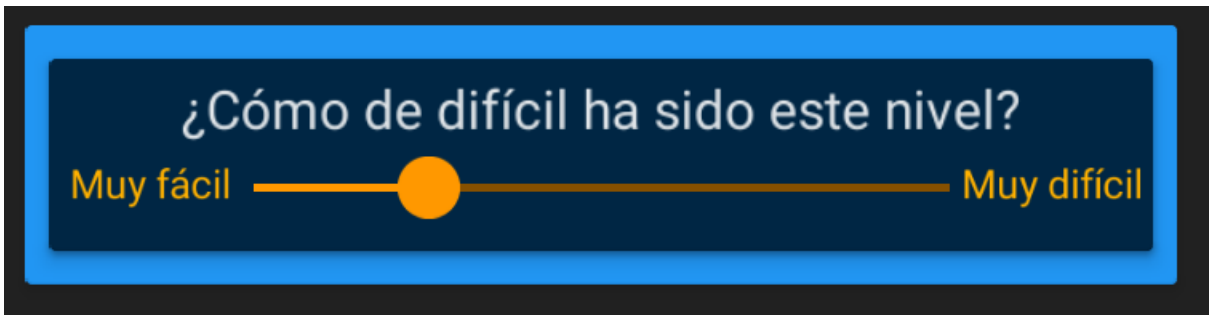

Ilustración 3.20: Cuestionario de la cuarta versión del proyecto

3.5.5 Análisis de resultados

Se recogen datos de un total de 109 partidas, repartidas entre 16 jugadores. Cuando empezamos a tratar los datos y buscar correlaciones entre las columnas nos damos cuenta de que las respuestas al cuestionario no muestran una fuerte correlación con ningún aspecto del juego, probablemente debido al aumento en la complejidad del proyecto.

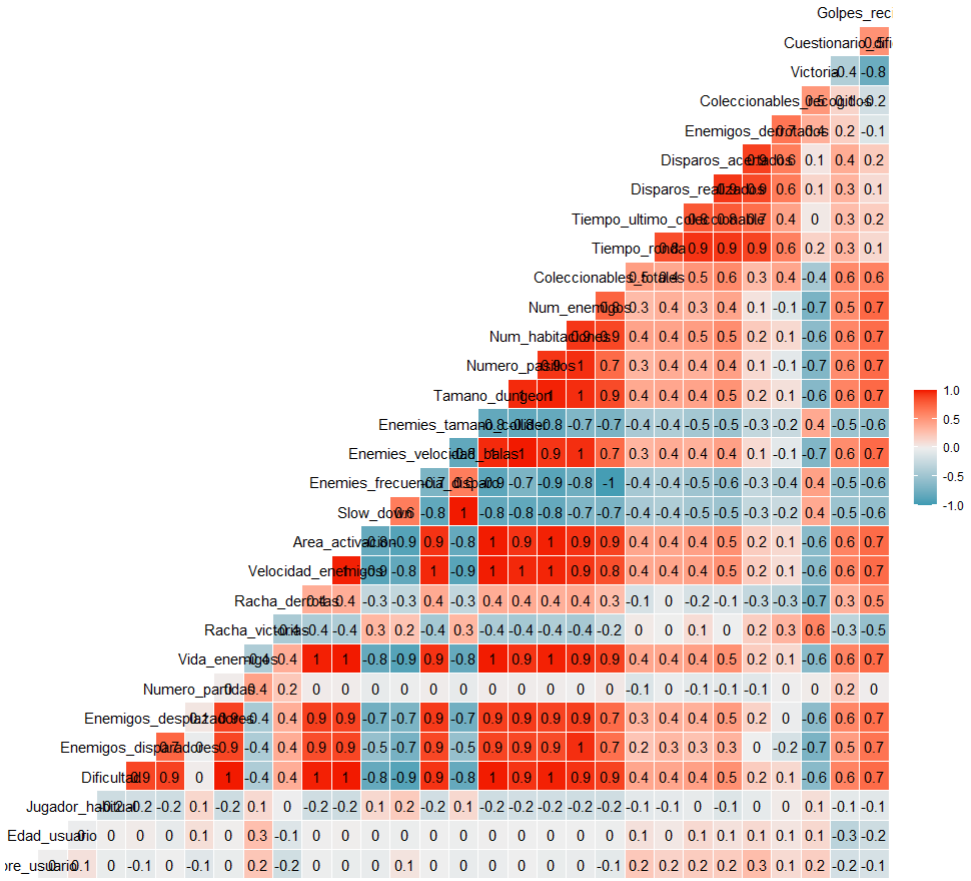

Ilustración 3.21: Gráfica de correlaciones de la cuarta versión del proyecto

En la columna "Cuestionario_dificil" se ven correlaciones con un valor máximo de 0.6, que es insuficiente. Es más complicado evaluar la dificultad ahora que hay 4 posibles configuraciones, menos diferenciadas entre sí. Además, el comportamiento de los enemigos ha cambiado, y con él el del juego y, por lo tanto, la manera en que se percibe la dificultad.

Esto pone de manifiesto que medir la dificultad en videojuegos no es una tarea fácil y cuanto más refinado es el videojuego, más refinada ha de ser la manera en que se evalúa.

Pese a este resultado, se decide entrenar de nuevo una máquina de soporte vectorial con kernel lineal, que había funcionado en el pasado, para modelar la creación de partidas.

La máquina resultante tiene un total de 72 vectores. Esta vez el hiperplano consigue predecir las valoraciones de los jugadores dado un nivel con un 33% de acierto, por lo que esta vez las predicciones de la IA no serán muy precisas.

4. Evaluación y resultados

En el presente capítulo se determinan los resultados obtenidos tras la cuarta iteración del experimento, con un proyecto capaz de realizar predicciones.

Con la IA entrenada se lanza de nuevo el proyecto. De nuevo se divide a los jugadores en dos grupos: los jugadores con dificultad dinámica adaptativa y el grupo de control, con dificultad aleatoria, siendo este último grupo el 33% de los jugadores totales.

Los jugadores del grupo de control tendrán 3 posibles configuraciones de dificultad sustancialmente diferenciadas entre sí, elegidas aleatoriamente.

Esta vez contamos con 16 jugadores que cumplimentaron la encuesta divididos en dos grupos: los usuarios de control (3) y el resto de los usuarios (13).

En la primera pregunta, "¿El juego es muy fácil?", el grupo de control se agrupa alrededor del 2, indicando que no están de acuerdo. Si consultamos la media y la mediana de las diferentes dificultades que se han generado a lo largo de las partidas de este grupo los valores son 1.27 y 1.5 respectivamente.

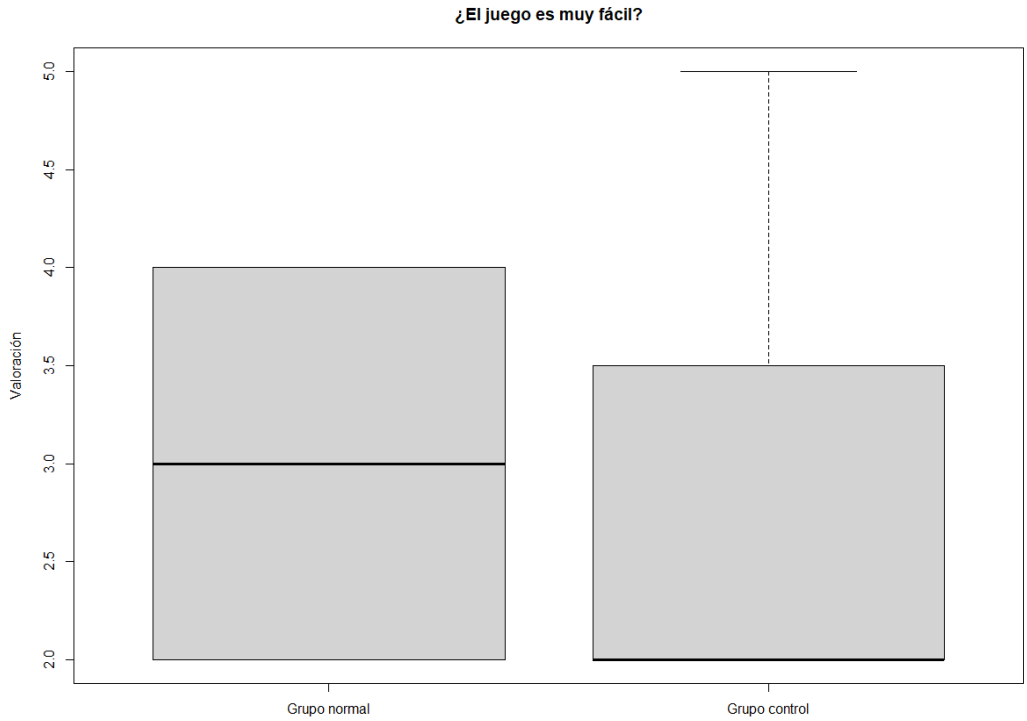

Ilustración 4.1: Diagrama de cajas y bigotes de la pregunta 1 del cuestionario de satisfacción. Cuarta iteración del experimento.

Por otro parte, el grupo de usuarios con ajuste de dificultad dinámico se agrupa alrededor del centro, de lo que podemos deducir que ha sido una experiencia equilibrada debido a las fórmulas que modelan la dificultad.

Para la segunda pregunta, "¿Los cambios de dificultad son muy bruscos?", tenemos el siguiente gráfico:

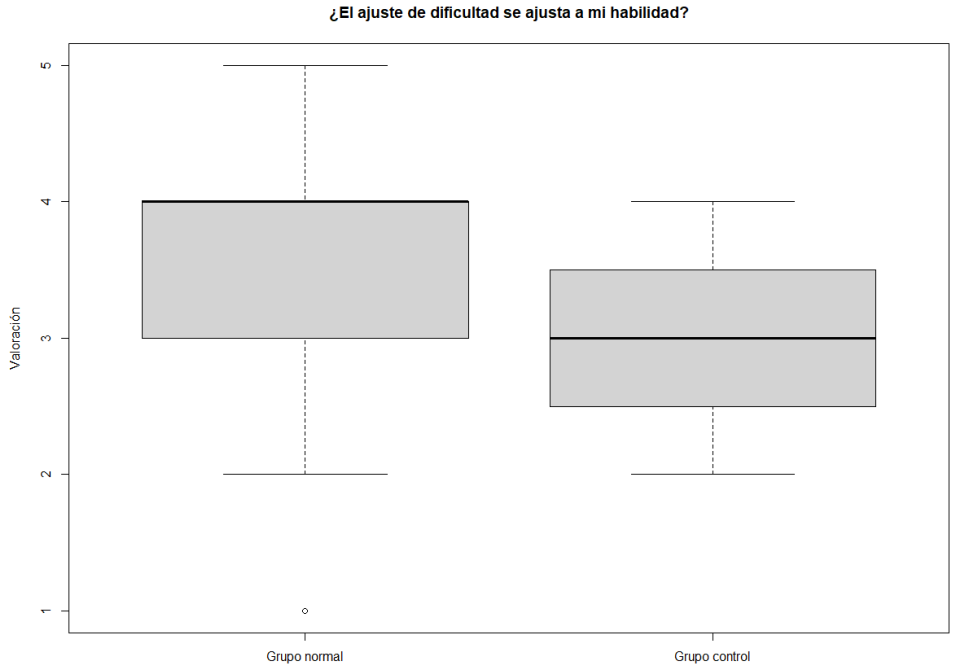

Ilustración 4.2: Diagrama de cajas y bigotes de la pregunta 2 del cuestionario de satisfacción. Cuarta iteración del experimento.

De nuevo el grupo de usuarios con ajuste dinámico de dificultad se agrupa alrededor del 3. El grupo de control tiene una mediana de 4, siendo los cambios de dificultad entre rondas.

La tercera de cuestión, "¿El ajuste de dificultad se ajusta a mi habilidad?", es de gran interés:

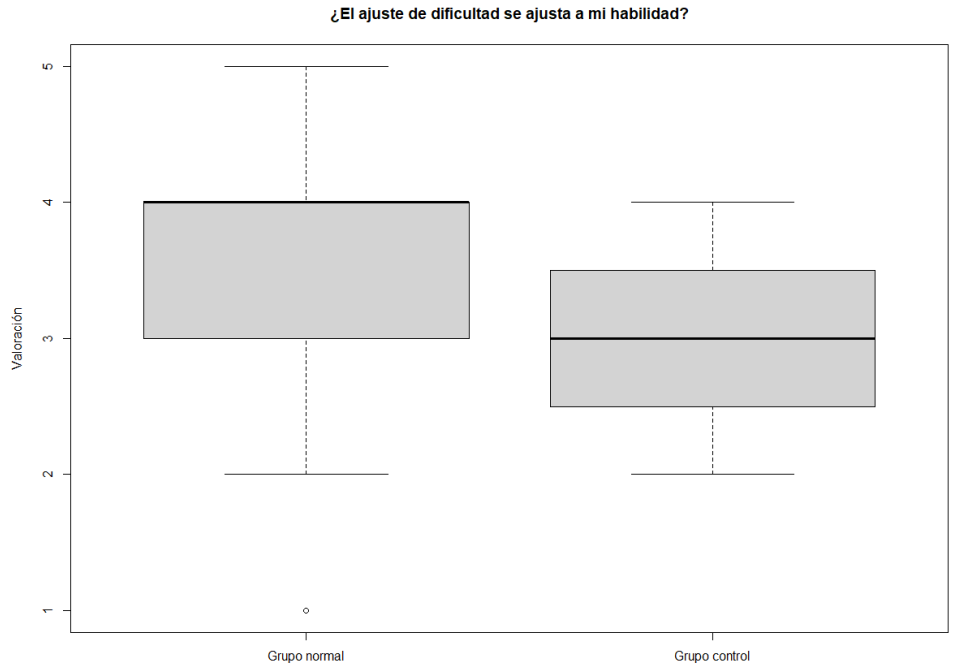

Ilustración 4.3: Diagrama de cajas y bigotes de la pregunta 3 del cuestionario de satisfacción. Cuarta iteración del experimento

La media y mediana del grupo de usuarios con ajuste de dificultad son 3.46 y 4.0, respectivamente. En el grupo de control, tanto la media como la mediana son 3.0. Si comparamos la media de los grupos la mejora es de un 9.2%.

Finalmente, la cuarta pregunta, "¿Se tienen en cuenta mis respuestas al cuestionario?", muestra resultados incoherentes:

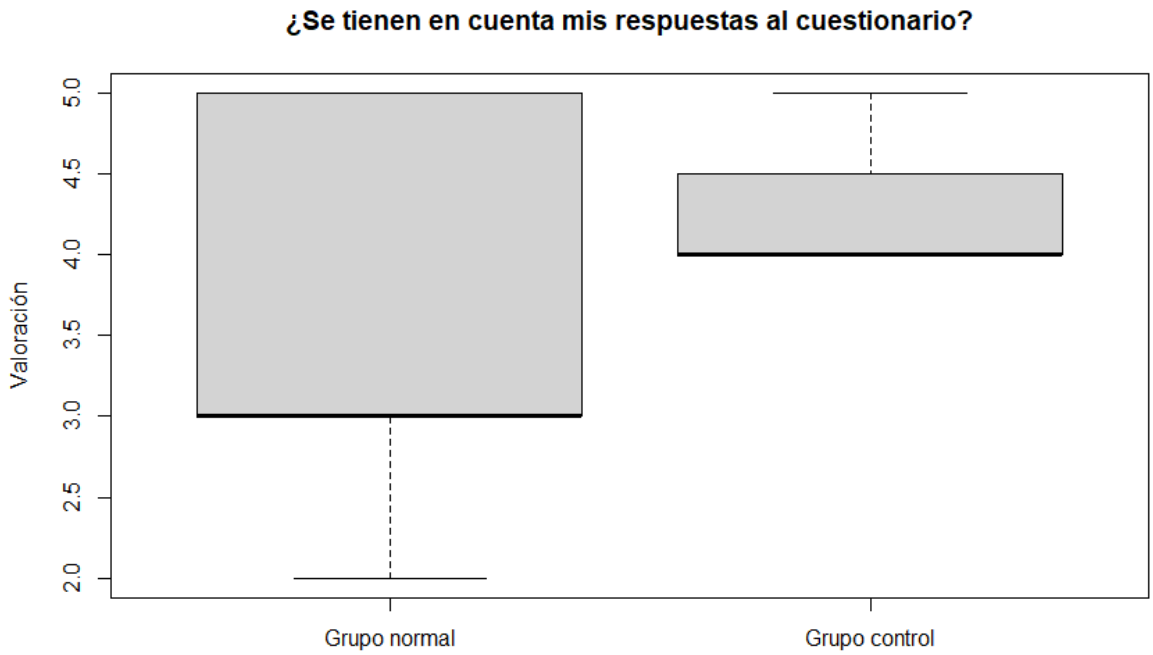

Ilustración 4.4: Diagrama de cajas y bigotes de la pregunta 4 del cuestionario de satisfacción. Cuarta iteración del experimento

El grupo de control tiene una mediana de 4, con una media de 4.33, mientras que el resto de los usuarios tiene una mediana de 3 y una media de 3.5. Al comparar las medias en este caso se produce un empeoramiento respecto a la selección aleatoria de dificultad de un 16.6%. Esto se debe a la falta de precisión por parte del modelo entrenado para predecir las valoraciones de los jugadores. De todos modos, las valoraciones son bastante altas, probablemente por la bondad de los usuarios a la hora de puntuar.

Aunque en la cuarta iteración del experimento el proyecto de un paso atrás, de forma general el proyecto logra alcanzar los siguientes objetivos:

- Obtener un set de datos abundante en la tercera iteración, con un total de 827 partidas jugadas.
- Completar el desarrollo de un videojuego con vista desde arriba funcional, con obstáculos y objetivos.
- Implementación de un sistema que modela la configuración de los niveles en función de la habilidad y retroalimentación del jugador.
- Una respuesta positiva por parte de los usuarios a la adaptación de dificultad.

5. Discusión

En este capítulo se comparan las decisiones de desarrollo del proyecto con las tesis tomadas gracias al trabajo previo.

Rogue sienta la base para los juegos de generación procedimental, hasta el punto de crear su propio género y ser uno de los más populares en la actualidad. Pero 40 años después de su lanzamiento, es lógico esperar que los algoritmos hayan progresado, los videojuegos se hayan refinado, y se obtengan experiencias más complejas y satisfactorias. En (Brenner P. de Castro, 2016) se realiza un análisis de dos títulos recientes, Crypt of the Necrodancer (Brace Yourself Games, 2015) y Shattered Planet (Kitfox Games, 2014), y de los aspectos que hacen exitosa a la generación procedimental. En este artículo se afirma que un juego bien diseñado, con una exploración interesante, favorece la posibilidad de jugar el título de nuevo. También se analizan técnicas seguidas para guiar al jugador por el mapa mediante el diseño y la disposición de elementos, así como sistemas de recompensa.

5.1 Mecánica principal. Movimiento

La utilización de inteligencia artificial para generar mapas o incluso imitar el comportamiento de los jugadores se lleva haciendo desde hace años. El caso más utilizado es el de Super Mario Bros, ya sea por su popularidad o su sencillez. Pese a que las mecánicas de un juego del género de plataformas pueden parecer sencillas: moverse y saltar en el momento adecuado, pueden ser muy complicadas de dominar. La aceleración y la distancia de salto variable contribuyen a hacerlo más difícil.

Siendo el movimiento la base de todo videojuego parece una opción arriesgada escoger uno con tantas posibilidades de obstaculizar al jugador. Por lo que un movimiento suave, constante y sin obstáculos, propio de los juegos de exploración, es una buena decisión. En este género, para modificar la dificultad, se generan mapas con distinto grado de dificultad a la hora de navegar por ellos o memorizarlos, dando como resultado mapas lineales o de estructura laberíntica. A este género pertenecen juegos como *Rogue* (Glenn Wichman, 1980) o *The legend of Zelda* (Shigeru Miyamoto, 1986).

Quedan restantes géneros como el de puzles o lucha, que resultan irrelevantes de estudiar en este ámbito, puesto que el mapa tiene poca importancia en sus partidas.

Los juegos de vista tridimensional se descartan, a diferencia de (Christine Bailey), porque como se muestra en (Buehler, 2019) los juegos que hacen uso de la cámara 3D, ya sea en primera o tercera persona, pueden producir mareos, náuseas e incluso cinetosis en los usuarios que no están acostumbrados a jugar o a controlar la cámara virtual al mismo tiempo que desplazan el avatar. Además, desarrollar juegos en 3D requiere más esfuerzo que los de 2 dimensiones, ya sea por la creación de objetos, como por la complejidad de la lógica.

En (Christine Bailey) implementan una serie de 4 mini-juegos 3D, cuyo número de dimensiones es una mala decisión si se cuenta con usuarios poco experimentados. Pese a que contempla cuatro mini-juegos de distintas características, el sistema no es útil para un

videojuego de puzles, por ejemplo. Queda revelado que un sistema así depende directamente del videojuego y no existe un sistema multidisciplinar.

5.2 Tipo de aplicación y control

La aplicación se desarrolla para ordenador por su inmediatez para realizar test y modificaciones, por su universalidad, popularidad y por la falta de necesidad de publicar el juego en una tienda, como ocurre en el mercado móvil o de consolas.

El control escogido es la combinación de teclado y ratón por su popularidad y fácil acceso, ya que no todo el mundo cuenta con un mando compatible con ordenador. La asignación de teclas es la inmediata y habitual en juegos para ordenador, para no trastornar a los jugadores habituales.

5.3 Estructura de los niveles

Los niveles del juego siguen una estructura base, alterando su grado de complejidad, por lo que todos comparten objetivos y mecánicas. El espacio se divide en habitaciones de distinto tamaño, unidas por 1 o más pasillos entre sí. En cada habitación puede haber o no enemigos, así como coleccionables. El objetivo para superar un nivel es conseguir todos los coleccionables de este.

El proyecto no cuenta con un nivel de tutorial para la explicación de las mecánicas o los controles, pues este nivel forma parte de la curva de aprendizaje y dificultad-reto característica de los videojuegos que este proyecto pretende modelar mediante una inteligencia artificial, por lo que esta tarea de diseño queda excluida al estar desarrollada por humanos.

5.4 Elementos del nivel

Durante las tres primeras iteraciones, el juego cuenta con un tipo de enemigo para favorecer la sencillez en cuanto a la información que aparece por pantalla y el análisis de rendimiento del jugador. El comportamiento del enemigo es el de correr hacia el jugador cuando está cerca para hacer daño por contacto. Su grado de dificultad varía en función de la velocidad a la que se mueve, la velocidad de respuesta, el tiempo de recuperación y el tamaño del área de activación, además de atacar en grupo.

La aplicación cuenta también con coleccionables repartidos por las habitaciones, representados como un rombo dorado. Pueden estar custodiados por grupos de enemigos.

El proyecto es sencillo y escalable, y permite establecer la dificultad de una forma directa. Más enemigos y con mayor agresividad dan como resultado mayor dificultad.

Sin embargo, en la cuarta iteración se decide introducir un nuevo tipo de enemigo. Este tipo permanece inmóvil, sigue al jugador con la mirada y le dispara cuando se encuentra a cierta distancia, con cierta cadencia de disparo. Dispara sin tener en cuenta cosas como su trayectoria o un comportamiento más refinado. Simplemente recoger la posición en la que se encuentra el

jugador en dicho instante y dispara. Al introducir este nuevo tipo de enemigo se altera la forma en la que medimos la dificultad. Más enemigos sigue siendo mayor dificultad, pero hay que identificar qué tipo es más peligroso y para qué tipo de jugador es más engorroso.

Antes de lanzar la cuarta versión del proyecto se tendría que haber evaluado precisamente esto. Se confirma así que, pese a que un juego más complejo aporta experiencias más ricas y satisfactorias, también es mucho más difícil de balancear y escalar.

5.5 Generación de mapas

Uno de los inconvenientes a la hora de trabajar con generación procedimental más pronunciado por artistas o desarrolladores es la falta de control sobre el resultado final. Si bien esto es cierto en casos como la generación mediante autómatas celulares (Brenner P. de Castro, 2016), el control depende de la calidad y optimización del algoritmo. Esto se refleja en casos de éxito como Minecraft, donde la generación de sus mundos es variada, compleja y creíble.

Utilizar algoritmos de tipo árbol para la generación de los mapas facilita la navegación por ellos y el análisis del rendimiento del jugador al estar divididos en habitaciones y pasillos.

5.6 Cuestionario

Se presenta un cuestionario, al igual que en (Rubem Jose Vasconcelos de Medeiros, 2014) para evaluar la dificultad del nivel con una respuesta del 1 al 5. La diferencia respecto al proyecto de Medeiros es que este cuestionario aparece al finalizar la ronda en lugar de cada 30 segundos. Pese a que se recogen datos sobre el daño sufrido durante la partida, no se analiza el momento u o el contexto en el que sucede, como por ejemplo la densidad de enemigos de la habitación en ese preciso instante.

5.7 Recogida de datos

Al igual que en el escenario propuesto por (Christine Bailey), se divide la dificultad en atributos del jugador, de no-jugador y de entorno, aunque en este proyecto las dos últimas comprenden un único grupo.

Pese a compartir esta división, existen atributos que no deberían ser modificados dinámicamente, principalmente los atributos del jugador. Si el jugador quiere llegar a dominar un videojuego con precisión, no podemos variar atributos como su velocidad o capacidad de salto sin avisarle. Se puede, sin embargo, alterar atributos que no tienen relación directa con el control, como el daño que realiza el jugador, o su vitalidad. Un buen ajuste es aquel que es invisible a los ojos de un usuario medio.

Para soportar la acción de sus algoritmos, podrían haber realizado un cuestionario previo al usuario para deducir qué tipo de jugador es, en lugar de hacer partir de cero al algoritmo.

5.8 Algoritmo de Inteligencia artificial

El proyecto destaca por el uso de aprendizaje automático para decidir los parámetros de generación de mapas, adaptándose a la habilidad de los jugadores. Utilizar un modelo de aprendizaje automático parece idóneo cuando se quiere resolver un problema en el que hay involucrados una gran cantidad de parámetros.

La decisión del algoritmo concreto de aprendizaje automático se hizo cuando el prototipo del sistema ya estaba implementado. Se tomó la decisión de usar SVM (*Support Vector Machines*), frente a otras alternativas. Las ventajas de SVM son su flexibilidad y aplicabilidad a un gran número de dominios. Además, las pruebas preliminares con otros algoritmos dieron resultados iguales o peores. El uso de otras técnicas como *Deep Learning* (Aprendizaje Profundo) no es posible en este proyecto debido fundamentalmente a que el volumen de datos que ha podido ser recogido está muy lejos de los mínimos necesarios para estos sistemas.

El problema debe resolverse introduciendo un dato de entrada, la dificultad, y obteniendo un conjunto de valores de salida, los parámetros del nivel. Para esto es necesario clasificar los datos y poder describir el comportamiento que siguen las valoraciones de los usuarios cuando juegan un mapa para definir un hiperplano. Se utiliza una máquina de soporte vectorial, que funciona como una caja negra, obtenemos un modelo entrenado, y realizamos predicciones dado un conjunto de datos.

5.9 Servidor de trazas

Para la recogida de datos, es común utilizar una API REST para comunicar datos con un servidor, que almacene estos en un archivo de información ordenada, cuya extensión puede ser .json o .xml. Esto es muy útil cuando se realizan pruebas no guiadas que no se pueden grabar, como resultado final, se obtiene un documento de texto que resume todo lo ocurrido en la partida sin necesidad de verla en directo. Al igual que (Rubem Jose Vasconcelos de Medeiros, 2014), el juego resultante es una aplicación cliente-servidor. De manera continua se envía información al servidor de lo que ocurre en la partida, y al final de la ronda, un resumen de los datos, junto con las respuestas al cuestionario. En el trabajo de Vasconcelos se recoge la valoración de la dificultad del nivel y de su diversión, además de una lista de momentos en los que el jugador perdió vida, para entenderlo en su contexto.

Hubiera resultado interesante implementar una funcionalidad parecida en la que recoger cómo perdió vida el jugador, o cuándo suceden bugs durante la partida. Finalmente, este trabajo decidió dejar de estimar la diversión, pero siguió evaluando la dificultad, como en (Rubem Jose Vasconcelos de Medeiros, 2014).

6. Conclusiones

Diseñar una IA capaz de ajustar por sí sola la dificultad de un videojuego de manera automática no deja de ser una idea atractiva, pero es una tarea titánica cuyo éxito se ve afectado por muchos factores y no tiene fácil solución. El objetivo del proyecto es utilizar un modelo de aprendizaje automático para lograr interpretar los datos generados por un videojuego y por un conjunto de usuarios, para adaptarse a las necesidades de estos.

El resultado obtenido por el sistema es una leve mejora (16%) en la satisfacción del usuario cuando juega a la versión con dificultad adaptada sin intervención humana. Esto deja claro que es necesario prestar atención a un diseño o un método a seguir a la hora de crear niveles, de posicionar sus elementos y de escoger la cantidad específica de estos. Sin embargo, el sistema no prueba ser más eficiente que un sistema diseñado enteramente por humanos.

Pese a esta mejora en la satisfacción, el proyecto no valora otros aspectos psicológicos como la frustración o el aprendizaje. En un videojuego a veces es necesario perder para aprender.

Dada la magnitud que supone balancear la dificultad general de un videojuego, sería interesante aplicar un sistema como el presente a un aspecto concreto de la dificultad, como por ejemplo la exploración espacial, o el combate, pero centrándose únicamente en un aspecto de estos. También sería interesante dedicar la potencia de un sistema de aprendizaje automático a mejorar o innovar en el campo de la accesibilidad en videojuegos, un proyecto que tiene la misma meta que el de esta memoria: acercar los videojuegos al mayor público posible. Se podría implementar un sistema guía para jugadores invidentes, entrenado con datos del resto de jugadores.

La inteligencia artificial tiene limitaciones como disciplina. Los algoritmos están diseñados para imitar un comportamiento inteligente, por lo que si se les plantea un problema distinto no responden adecuadamente, es decir, un sistema como el propuesto es compatible únicamente con juegos que posean características muy similares al escogido en este experimento.

Incluso obteniendo resultados positivos, este tipo de generación no es eficaz para generar un videojuego completo, pues los mapas generados son siempre del mismo tipo y la mecánica resultante es siempre la misma. Para un videojuego real haría falta un conjunto de sistemas de inteligencia artificial especializados en diseñar distinto tipo de niveles, para añadir variedad a la experiencia de juego, para poder diseñar tutoriales y que el jugador aprenda a jugar, y por último sistemas que aprendan a enlazar los anteriores expuestos.

La dificultad adaptativa amplía el público objetivo, es imposible que este sea universal porque el videojuego sigue sujeto a un género y temática que puede conectar o no con el jugador. Por otro lado, hay juegos tan complejos que no admiten un ajuste dinámico de dificultad si el jugador no cuenta con unas habilidades mínimas.

6.1 Trabajo futuro

En las conclusiones de (Christine Bailey) se evidencia que es necesario un diseño complejo para obtener un ajuste de la dificultad rico y preciso, puesto que permite modificar un mayor número de parámetros. Un juego completo es una experiencia total y adictiva para el jugador, con recompensas que le invitan a seguir jugando. Sin embargo, mini-juegos que funcionan como un banco de pruebas resultan rápidamente repetitivos, aburridos y vacíos en comparación con un juego comercial. Los juegos comerciales pueden tener una duración que va desde las 4 horas hasta 100 o más. Sería interesante poder evaluar a lo largo de esas horas cuáles han sido las reacciones del usuario y cuáles son las mejores y peores escenas.

Un juego completo cuenta con variedad de niveles, escenarios, enemigos, un sistema de recompensas profundo, un sistema de progresión del jugador, control sobre el ritmo de la acción, variedad en las mecánicas y otros elementos.

Un diseño de videojuego más complejo, junto con una recogida de datos más meticulosa, permitiría obtener como resultado porcentajes mayores de satisfacción, y un análisis más rico en relación con los elementos encargados de aportar dificultad, variedad y diversión a un videojuego.

Como se expone en (Brenner P. de Castro, 2016) un buen sistema de generación procedimental no solo se centra en la arquitectura de los mapas, sino en generar buenas ocasiones de exploración e incluir un sistema de recompensas interesante.

Por falta de tiempo, no se pudo implementar un sistema que entrene de manera continua el SVM, almacenando la información del modelo en un servidor y actualizándolo periódicamente. Además, el servicio utilizado para almacenar los recursos necesarios para su funcionamiento, Amazon Web Services, es de pago una vez se supera un umbral de llamadas al servidor. Por estos dos motivos, la versión final utiliza siempre el mismo set de datos y el modelo SVM no aprende nuevas configuraciones a no ser que se le entrene de nuevo de forma manual y sea el usuario el que actualice el ejecutable.

No es descabellado pensar que el resultado del proyecto es mejorable con un grupo de usuarios mayor y más heterogéneo, un juego con un mayor número de capas de complejidad y profundidad, y un sistema de obtención de información más minucioso. Esto por supuesto requiere de más medios y tiempo del disponible. Porque si se quiere ajustar el *flujo* del juego, se debe tener en cuenta la dificultad, pero también las recompensas**.**

En la segunda versión del experimento se abandona la idea de evaluar la diversión para aligerar el proyecto. El concepto de diversión es subjetivo y difícil de explicar en un cuestionario, pero se puede identificar qué grupo de elementos son responsables de su aparición si dividimos el concepto en partes más pequeñas. Por ejemplo, diseccionando un nivel en sus elementos más básicos y preguntando a un grupo de usuarios qué elementos les ha parecido necesarios y divertidos, pudiendo establecer así un top mediante sus respuestas.

Asimismo, un cuestionario más extenso, dirigido por un humano o presencial, hubiera dado más información y de mayor calidad.

Hacer videojuegos es caro, arriesgado y lleva mucho tiempo, por lo que una actividad como la propuesta solo podría ser llevada a cabo de forma exitosa por un estudio grande y con experiencia. Los videojuegos cambian constantemente, por lo que el sistema tendría que ser retroalimentado de manera continua para seguir siendo útil. Y quién sabe, puede que dentro de unos años el debate acerca de la dificultad en videojuegos sea irrelevante.

6. Conclusions

Designing an AI capable of dynamic difficulty adjustment is an attractive idea, but the success of this task depends on a lot of factors and is difficult to achieve. The main goal of this project is to apply machine learning to interpret data generated by a video game and its users to adapt to their necessities.

The result of this experiment is a slight improvement (16%) in the users' satisfaction when playing with the difficulty adjusted to their ability. This establishes the need to pay attention to the design or method to follow when creating levels, positioning elements, and choosing the specific amount of these. However, the project does not show any improvement over a system created entirely by humans.

Despite this improvement in the satisfaction, the project does not measure other psychological aspects such as frustration or learning. Sometimes you must lose in a video game to learn.

Given the magnitude that balancing the general difficulty of a video game requires, it would be interesting to introduce a system like the current one to a concrete aspect of the difficulty, such as space exploration or combat, but focusing only on one aspect of these. It would also be interesting to dedicate the power of an automatic learning system to improve or innovate in the field of accessibility in video games, a project that has the same goal as the one in this memory: get the video games closer to the biggest number of people possible. A guide system could also be implemented for blind people, trained with other players' data.

Artificial intelligence is a limited discipline. Algorithms are designed to emulate intelligent behaviors and they are not capable of changing the task they were programmed for. A project like this is only functional with video games and environments like the one that has been proposed for this experiment.

Despite the positive results, this algorithm is not useful to create a complete video game as its levels are always the same type and structure, resulting in a boring experience. In realistic circumstances, to create a good video game it would be necessary to have a set of systems specialized in the design of different types of levels, to add variety to the gaming experience, and to show tutorials and teach the player how to play. Lastly, it would need systems to connect all the previous systems mentioned.

The dynamic difficulty adjustment enlarges target audiences, but it is impossible to obtain a universal audience because every video game still depends on a genre and a theme that may or not connect to a player. Additionally, some video games are so complicated that they do not admit any adjustment in the difficulty system and require skills previously learned by the player.

6.1 Future work

(Christine Bailey) concludes in his work that it is necessary to have a complete video game to create a rich experience and to adjust multiple parameters of difficulty. A complete video game is a total and engaging experience, with regards that invite him to continue playing. Nevertheless, a testbed formed by mini games feel repetitive. Commercial videogames usually last from 4 hours to 100 or more. It would be interesting to evaluate the reactions of players during game sessions in a long-term video game to find out its strengths and weaknesses.

A complete video game counts with plenty of diverse levels, scenarios, enemies, a deep reward system and a great variety of mechanics.

A more complex design, alongside a more meticulous data collection would acquire higher percentages of satisfaction, variety, and fun.

As (Brenner P. de Castro, 2016) says, a good procedural generation system not only generates good structures, but also opportunities of exploration and interesting rewards.

Due to lack of time, a system responsible for continuous training of the SVM was not possible to complete on time. Besides, the service used for store data online, Amazon Web Services, follows a payment system. Because of this, final version always uses the same dataset and SVM model does not learn from new game sessions.

The project would have improved if the AI system were capable of decide which value of difficulty the player needs. Furthermore, the resulting possible set of maps is limited.

It is not unreasonable to assume that the resulting project is improvable with more subjects and a more complex videogame. This requires facilities and resources and a bigger amount of time. Because the flow of a video game does not rely only on difficulty, it also depends on rewards.

The second version of the project gives up on fun evaluation. The concept of fun is subjective and difficult to explain with just a little poll. It would be effective to divide video game mechanics into little pieces to identify the fun ones. Again, this is only possible with a whole experience and a complex video game, that requires a higher investment.

Likewise, a larger poll or a poll directed by human intervention would have supplied more valuable information.

Video game production is expensive, risky and takes a lot of time. Only a big team with experience and resources would make this project possible. Video games are changing constantly, therefore the resulting system will have to be updated continually to be useful and to keep up. Who knows, maybe in a few years video game’s difficulty debate will be irrelevant.

7. Bibliografía

NINTENDO. KAWASHIMA, R. (2005) *Brain Training del Dr. Kawashima ¿Cuántos años tiene tu cerebro?*. Nintendo DS.

Brown, M. [Game Maker's Toolkit]. (2018, abril 6). How to Keep Players Engaged (Without Being Evil). Recuperado de https://www.youtube.com/watch?v=hbzGO_Qonu0

Brown, M. [Game Maker's Toolkit]. (2015, junio 2). What Capcom Didn't Tell You About Resident Evil 4. Recuperado de https://www.youtube.com/watch?v=zFv6KAdQ5SE

Brown, M. [Game Maker's Toolkit]. (2019, enero 28). Roguelikes, Persistency, and Progression. Recuperado de https://www.youtube.com/watch?v=G9FB5R4wVno&

Brown, M. [Game Maker's Toolkit]. (2018, marzo 14). What Makes a Good Puzzle?. Recuperado de https://youtu.be/zsjC6fa_YBg

Brown, M. [Game Maker's Toolkit]. (2018, marzo 14). How (and Why) Spelunky Makes its Own Levels. Recuperado de https://youtu.be/Uqk5Zf0tw3o

Altozano, J. [DayoScript]. (2014, mayo 11). Nuzlocke y la importancia de la derrota. Recuperado de https://www.youtube.com/watch?v=RxudEpgiHAY

Altozano, J. [DayoScript]. (2019, abril 17). ¿Importa la dificultad en el videojuego?. Recuperado de https://youtu.be/rnglNZZGh-s

O'Dwyer, D. [Noclip Documentaries]. (2017, abril 15). The Making of Spelunky. Recuperado de https://www.youtube.com/watch?v=jv434Xyybqc

Baldino, J. [gameranx]. (2018, abril 30). 10 Best Games That Change Difficulty Based On How Well You Play. Recuperado de https://www.youtube.com/watch?v=PKCMV2m_hls

Dynamic game difficulty balancing. (Sin fecha). En Wikipedia. Recuperado el 19 de noviembre de 2019 de https://en.wikipedia.org/wiki/Dynamic_game_difficulty_balancing

Einhorn, A. (2015, mayo 28). Four-step puzzle design. Lugar de publicación: Gamasutra. https://www.gamasutra.com/blogs/AsherEinhorn/20150528/244577/Fourstep_puzzle_design.php

Anderson, M. (2014, junio 26). Dungeon-Building Algorithm. Lugar de publicación: Roguebasin. http://www.roguebasin.com/index.php?title=Dungeon-Building_Algorithm

Buehler, A. (2019, noviembre 11). Third person camera view in games - a record of the most common problems in modern games, solutions taken from new and retro games. Lugar de publicación: Gamasutra. https://www.gamasutra.com/blogs/AndreasBuehler/20191111/353709/Third_Person_Camera_View_in_Games__a_record_of_the_most_common_problems_in_modern_games_solutions_taken_from_new_and_retro_games.php

León, D. (2013). Roguelike development in Unity: Part 1. Lugar de publicación: Tumblr. http://xdavidleon.tumblr.com/post/51104205836/roguelike-development-in-unity-part-i

Marston, W. The heider-Simmel Illusion. https://youtu.be/8FIEZXMUM2I

Stephenson, J. (2018, noviembre 29). 6 Ways Machine Learning will be used in Game Development. Lugar de publicación: Logikk. https://www.logikk.com/articles/machine-learning-in-game-development/

McAloon, A. (2019, febrero 12). Ubisoft aims to help machine learning find a place in every stage of game dev. Lugar de publicación: Gamasutra. https://www.gamasutra.com/view/news/336478/Ubisoft_aims_to_help_machine_learning_find_a_place_in_every_stage_of_game_dev.php

Santana, C. [DotCSV]. (2018). Aprendiendo Inteligencia Artificial. Recuperado de https://www.youtube.com/playlist?list=PL-Ogd76BhmcDxef4liOGXGXLL-4h65bs4

Woods, D. (2017). R correlation tutorial. Lugar de publicación: datacamp. https://www.datacamp.com/community/blog/r-correlation-tutorial

Chih-Chung Chang y Chih-Jen Lin. (2011). LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology. Recuperado de http://www.csie.ntu.edu.tw/~cjlin/libsvm

Ng, A. CS229 Lecture notes. Part V: Support Vector Machines. Lugar de publicación: Stanford. http://cs229.stanford.edu/notes/cs229-notes3.pdf

Kumar, A. (2018). Classifying data using Support Vector Machines(SVMs) in R. Lugar de publicación: geeksforgeeks. https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-r/

Real Academia Española. (2001). Diccionario de la lengua española (23.a ed.). Madrid, España: Autor. [versión 23.3 en línea]. Consultado en https://www.rae.es

Acid Nerve. (20 de 07 de 2021). Death's Door.

Adam, T. (2014). PROCEDURAL MUSIC GENERATION AND ADAPTATION BASED ON GAME. Obtenido de https://pdfs.semanticscholar.org/4e3a/1aa68638f3ccb3cfb83f08ba0fda547a8aa6.pdf

Adams, E. (14 de mayo de 2008). *Gamedeveloper*. Obtenido de https://www.gamedeveloper.com/design/the-designer-s-notebook-difficulty-modes-and-dynamic-difficulty-adjustment

Adeel Zafar, S. H. (2019). Corpus for Angry Birds Level Generation.

Alexander Baldwin, S. D. (2017). Mixed-initiative procedural generation of dungeons using game design patterns.

Arman Balali Moghadam, M. K. (2017). A genetic approach in procedural content generation for platformer games level creation, X.

Asobo Studio. (13 de septiembre de 2016). ReCore.

Ayora, V. (09 de junio de 2021). *Esports.as.com*. Obtenido de https://esports.as.com/fortnite/Cuantas-personas-juegan-Fortnite-2021_0_1472252762.html

BBVAOpen4u. (23 de marzo de 2016). *bbvaopen4u*. Obtenido de https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

Bethesda. (2008). Fallout 3.

Blizzard Entertaiment. (1996). Diablo.

Blizzard Entertainment. (11 de marzo de 2014). Hearthstone: Heroes of Warcraft. *Hearthstone*.

Blizzard Entertainment. (11 de marzo de 2014). Hearthstone: Heroes of Warcraft.

Brace Yourself Games. (23 de abril de 2015). Crypt of the necrodancer.

Brenner P. de Castro, R. R. (2016). Level Design on Rogue-like Games: An Analysis of Crypt of the.

Brown, M. (25 de abril de 2016). Obtenido de https://www.youtube.com/watch?v=NInNVEHj_G4

Buehler, A. (2019). *Gamasutra*. Obtenido de Gamasutra: https://www.gamasutra.com/blogs/AndreasBuehler/20191111/353709/Third_Person_Camera_View_in_Games__a_record_of_the_most_common_problems_in_modern_games_solutions_taken_from_new_and_retro_games.php

Capcom. (2005). Resident Evil 4.

Casademont, R. S. (17 de agosto de 2021). *Esquire*. Obtenido de https://www.esquire.com/es/tecnologia/a37325079/videojuegos-deberian-tener-modo-facil-dark-souls/

Christine Bailey, M. K. (s.f.). An experimental testbed to enable auto-dynamic difficulty in modern video games.

Clover Studio. (2006). God hand.

Compile. (1986). Zanac.

Csikszentmihalyi, M. (1990). *Fluir: Una psicología de la felicidad.* Kairós.

Dale, L. K. (02 de octubre de 2017). *kotaku*. Obtenido de https://www.kotaku.co.uk/2017/10/02/i-find-cupheads-difficulty-infuriating-not-fun

Daniel Michelon De Carli, F. B. (2011). A Survey of Procedural Content Generation Techniques Suitable to Game Development.

David Braven, I. B. (1984). Elite.

David Stammer, T. G. (2015). Player-adaptive Spelunky level generation.

Diaz-Furlong Hector Adrian, S.-G. C. (2013). An approach to level design using procedural content generation and difficulty curves.

Edmund McMillen, F. H. (28 de septiembre de 2011). Binding of Isaac.

elpublicista. (30 de mayo de 2019). *elpublicista.es*. Obtenido de https://www.elpublicista.es/mundo-online/casi-70-poblacion-espanola-consume-videojuegos-donde-30-son

Epic Games. (21 de julio de 2017). Fortnite. *Fortnite*.

Fernández, I. (23 de Septiembre de 2018). *Xataka.* Obtenido de Xataka: https://www.xataka.com/videojuegos/ponmelo-dificil-como-se-estructura-la-dificultad-en-un-videojuego

Forbes. (s.f.). *forbes.com*. Obtenido de https://www.forbes.com.mx/industria-videojuego-suma-500-millones-de-jugadores/

Frictional Games. (2010). Amnesia: dark descent.

Frictional games. (22 de septiembre de 2015). SOMA.

From Software. (octubre de 2011). Dark Souls.

From Software. (22 de marzo de 2019). Sekiro shadows die twice. Japón.

García, J. (2015). *es.ign.com*. Obtenido de es.ign.com: https://es.ign.com/reportaje/92847/feature/la-curva-de-dificultad-en-videojuegos

Gearbox. (2003). Home world 2.

Gearbox. (2009). Borderlands.

Gervas, P. (7 de julio de 2009). Computational Approaches to Storytelling and Creativity. *AI Magazine*, 30(3), 49.

Glenn Wichman, M. T. (1980). Rogue: Exploring the Dungeons of Doom.

Grub, J. (6 de septiembre de 2017). *venturebeat.com.* Obtenido de https://venturebeat.com/2017/09/06/is-it-ok-if-we-suck-at-games-gamesbeat-decides/

Hee-Seung Moon, J. S. (28 de 06 de 2020). Dynamic Difficulty Adjustment via Fast User Adaptation.

Hello games. (9 de agosto de 2016). No man's sky.

Heras, J. M. (28 de mayo de 2019). *iartificial.net*. Obtenido de https://iartificial.net/maquinas-de-vectores-de-soporte-svm/#Algunas_aplicaciones_de_las_maquinas_de_vectores_de_soporte

Housemarque. (30 de 04 de 2021). Returnal.

Hu, R. (s.f.). *robinforest.net*. Obtenido de robinforest.net: http://robinforest.net/post/cellular-automata/

InvexGames. (s.f.). *github.com/InvexGames/MaterialUI*. Obtenido de https://github.com/InvexGames/MaterialUI/releases

Itrch, S. (30 de Octubre de 2015). *Asset Store*. Obtenido de https://assetstore.unity.com/packages/tools/utilities/dungeon-tools-46732

Jiménez, J. (8 de Enero de 2019). *Xataka*. Obtenido de https://www.xataka.com/otros/de-profesion-cientifico-de-datos

Jiménez, J. (16 de octubre de 2020). *Xataka*. Obtenido de https://www.xataka.com/otros/cientifico-datos-asi-profesion-demandada

Johnson, L. &. (2010). Cellular automata for real-time generation of infinite cave levels.

Kitfox Games. (2014). Shattered planet.

Konami. (2015). Metal gear solid V: The phantom pain.

Lucas Ferreira, C. T. (2014). A Search-based Approach for Generating.

Lucas Ferreira, C. T. (2014). A search-based approach for generating Angry Birds levels.

Matt Thorson, N. B. (25 de enero de 2018). Celeste.

Michael Toy, G. W. (1980). Rogue.

Misaki Kaidan, C. Y. (2015). Procedural generation of angry birds levels that adapt to the player's skills using genetic algorithm.

Miyamoto, S. (1983). Super Mario Bros. Japón.

Motion Twin. (10 de mayo de 2017). Dead Cells.

Munguía, E. (2017). *enrique7mc.com*. Obtenido de http://www.enrique7mc.com/2017/03/crear-una-api-rest-en-30-segundos-con-json-server/

Nakamoto, S. (1983). Bomberman. Japón.

Naughty Dog. (1996). Crash Bandicoot.

Newtonsoft. (s.f.). *newtonsoft.com*. Obtenido de https://www.newtonsoft.com/json

Nintendo. (1992-2019). Mario Kart.

Nishikado, T. (1978). Space Invaders.

Pázhitnov, A. (6 de junio de 1984). Tetris. Moscú.

pharindoko. (s.f.). *github.com/pharindoko/json-serverless*. Obtenido de https://github.com/pharindoko/json-serverless

Playdead. (21 de julio de 2010). Limbo.

Polyphony Digital. (1999). Omega Boost.

Real Academia Española. (2021). *dle.rae.es*. Obtenido de https://dle.rae.es/

Rego, R. L. (2017). *diariodeunjugon*. Obtenido de diariodeunjugon.com: http://www.diariodeunjugon.com/dificultad-videojuegos-analisis/

Remedy. (2001). Max Payne.

Robin Hunicke, V. C. (s.f.). AI for Dynamic Difficulty Adjustment in Games.

Rovio Entertainment. (2009). Angry birds.

r-project.org. (s.f.). Obtenido de https://www.r-project.org/

Rubem Jose Vasconcelos de Medeiros, T. F. (2014). Procedural Level Balancing in Runner Games.

sacredseedstudio.com. (s.f.). Obtenido de https://www.sacredseedstudio.com/tutorials/output-the-unity-console-to-file/

Santana, C. (1 de noviembre de 2017). *Youtube*. Obtenido de https://youtu.be/KytW151dpqU

Sérgio Oliveira, L. M. (s.f.). Adaptive content generation for games.

SHAKER, N. T. (2016). Procedural content generation in games. En N. T. SHAKER, *Procedural content generation in games.* Springer.

Sheena Angra, S. A. (2017). Machine learning and its applications: A review.

Shigeru Miyamoto, T. T. (21 de febrero de 1986). The legend of Zelda. Japón.

Short, T. (12 de diciembre de 2016). *gamasutra.com*. Obtenido de https://www.gamasutra.com/blogs/TanyaXShort/20161216/310387/Procedurally_Generating_Personalities.php

Stålberg, O. (30 de 07 de 2020). Townscaper.

Steve Dahlskog, J. T. (2012). Patterns and Procedural Content Generation.

Steve Dahlskog, J. T. (2013). Patterns as Objectives for Level Generation.

Steve Dahlskog, J. T. (2014). Linear levels through n-grams.

Steve Dahlskog, J. T. (s.f.). A Multi-level Level Generator.

Studio MDHR. (29 de septiembre de 2017). Cuphead. Canadá.

Summerville, A. (2015). Monte Carlo Tree Search to Guide Platformer Level Generation.

Superhys. (25 de febrero de 2018). *igpubcast.com*. Obtenido de https://igpubcast.com/2018/02/25/assist-modes-are-making-games-easier-and-thats-okay/

Team Cherry. (24 de febrero de 2017). Hollow Knight.

typicode. (2013). *github.com/typicode/json-server*. Obtenido de https://github.com/typicode/json-server

Unity Engine. (s.f.). *unity3d.com*. Obtenido de https://unity3d.com

Unity Technologies. (s.f.). *docs.unity3d.com*. Obtenido de https://docs.unity3d.com/Manual/UnityWebRequest-SendingForm.html

Valve. (2008). Left 4 Dead.

Vega, C. S. (18 de junio de 2019). *Youtube*. Obtenido de https://youtu.be/qTNUbPkR2ao

Víctor Manuel Álvarez Pato, D. C. (2013). Dynamic difficulty adjusting strategy for a two-player video game.

Vlambeer. (11 de octubre de 2013). Nuclear Throne.

William M. P. Reis, L. H. (2015). Human computation for procedural content generation in platform games.

Xataka. (26 de agosto de 2020). *Xataka*. Obtenido de https://www.xataka.com/n/nueve-industrias-que-aplican-ciencia-datos-para-solucionar-problemas-reales

Yu, D. (2014). *Spelunky.* Boss fight books.