UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

"Frozen Out", videojuego de aventura gráfica 3D. Diseño y uso de los diálogos

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Vicent Pla Madrid

Tutor: Manuel Agustí Melchor

2019-2020

Resumen

Esta memoria trata el desarrollo de un proyecto de emprendimiento, un videojuego de aventuras y puzles en tres dimensiones, dentro del marco de la universidad, así como la implementación de los sistemas de juego que posibilitaron las funcionalidades del diálogo. El producto se realizó dentro de un equipo multidisciplinar de cinco estudiantes y utilizando el motor de videojuegos Unity.

La propuesta original se analizó y se desarrollaron un conjunto de características para evaluar su viabilidad en dos experimentos frente a público objetivo, los cuales resultaron tener una recepción positiva. Por otro lado, en cuanto al apartado técnico se indagaron e integraron herramientas existentes centradas en el control de conversaciones y se implementación extensiones a este para acomodarse a las necesidades del proyecto en cuestión, resultando en la creación de un gestor de diálogos flexible y personalizable para otros proyectos.

Palabras clave: videojuego, diálogo, emprendimiento, Unity.

Resum

Aquesta memòria tracta el desenvolupament d'un projecte d'emprenedoria, un videojoc d'aventures i puzles en tres dimensions, dins del marc de la universitat, així com la implementació dels sistemes de joc que possibiliten les funcions del diàleg. El projecte es va realitzar dins d'un equip multidisciplinar de cinc estudiants i utilitzant el motor de videojocs Unity.

La proposta original es va analitzar i es varen desenvolupar un conjunt de característiques per a avaluar la seua viabilitat amb dos experiments orientats al públic objectiu, els quals van rebre una recepció positiva. En quant a l'apartat tècnic, es varen indagar e integrar ferramentes existents centrades en el control de conversacions y es varen implementar extensions a aquest per a acomodar-se a les necessitats del videojoc, resultant en un gestor de diàlegs flexible i personalitzable per a altres projectes.

Paraules clau: videojoc, diàleg, emprenedoria, Unity.

Abstract

This paper is about the development of an entrepreneur project, a 3D adventure videogame with puzzles, inside the framework of the university. It will also dive into the implementation of the game systems related to the dialogue functionalities. The project was conducted inside a multidisciplinary team of five students and Unity was used as the game engine.

The original proposal was therefore analysed, and a set of characteristics was picked to be developed and carry out two experiments aimed at the public, where the project received positive feedback. In regard to the technical aspects, a variety of tools designed to control conversations were researched and they were integrated and extended to adjust them to our videogame's need, which resulted in the creation of a flexible dialogue manager that could be personalised for other projects.

Keywords: videogame, dialogue, entrepreneurship, Unity.

Tabla de contenidos

1. Introducción 8
   1.1. Motivación 8
   1.2. Objetivos 9
   1.3. Estructura 9
2. Generación y evaluación de la idea de negocio 11
   2.1. Generación de la idea de negocio 11
   2.2. Estudio de mercado 11
   2.3. Análisis DAFO 19
   2.4. Modelo de negocio y proyección económica 20
   2.5. Lean Canvas 24
   2.6. Conclusiones 26
3. Desarrollo de idea de negocio 27
   3.1. Mapa de características 28
   3.2. Tecnología utilizada 29
   3.3. Primer *Minimum Viable Product* 32
   3.4. Segundo *Minimum Viable Product* 39
   3.5. Mercadotecnia y difusión 47
4. Diálogos 48
   4.1. Estado del arte 48
   4.2. Uso de diálogos realizado en el primer MVP 53
   4.3. Diseño y desarrollo de un Gestor de Diálogos 61
   4.4. Alternativas de Voz 66
   4.5. Implementación de voz 69
   4.6. Estilo de diálogo 72
   4.7. Interacción 78
   4.8. Internacionalización del diálogo 83
   4.9. Conexión con Instagram 84
5. Demos en Unity 88
   5.1. Estilos de texto 89
   5.2. Opciones 91
   5.3. Sistemas secundarios 92
6. Conclusiones 94

7. Trabajo futuro 95
Bibliografía 96
Índice de figuras 98
Índice de tablas 101
Anexo I: Partida guiada de segundo MVP de Frozen Out 102
Anexo II: Diagrama de diálogos completo 107

1. Introducción

Los videojuegos constituyen una industria en constante crecimiento y con unos beneficios que llegan a superar los del cine, solo en 2019 en España la industria del videojuego facturó un total de 1.479 millones de euros [1]. El desarrollo de un videojuego comprende una gran variedad de habilidades y tipos de profesionales debido a la inclusión e integración de diferentes recursos necesarios para llevar a cabo un proyecto de este tipo, de modo que es capaz de juntar personas provenientes de un amplio ámbito, especialmente diseñadores, escritores, informáticos, animadores, así como líderes de proyecto, entre otros. Además, ofrece un entorno real para poner en práctica conocimientos dados durante la titulación, sobre todo en cuestiones de integración, organización, rendimiento y gestión de proyectos. Son también necesarias las habilidades de trabajo en equipo para colaborar con distintas personas, pudiendo también perfeccionar de forma práctica las capacidades de liderazgo y dirección de un proyecto con salida al mercado.

Este trabajo de fin de grado abarca el proceso de creación de un videojuego independiente[1], cubriendo desde la concepción de la idea original, la evaluación de mercado de esta idea, el desarrollo de dos productos mínimos viables[2] (*Minimum Viable Product* o MVP por sus siglas en inglés) y las pruebas reales con público objetivo del resultado de cada MVP. Como parte de un proyecto en equipo también se indagará con más detalle en los aspectos técnicos relativos a la implementación del diálogo, un aspecto relevante debido a la importancia de las conversaciones en el videojuego: se cubrirá su evolución, cambios y extensiones durante el desarrollo del segundo MVP, sobre todo describiendo su diseño interno, las herramientas usadas, las alternativas propuestas, así como las extensiones a estas para acomodarse a los requisitos del juego.

El videojuego en cuestión se titula *Frozen Out*, se desarrolla con el motor de videojuegos Unity [2] y para la plataforma Microsoft Windows. El proyecto se ha desarrollado dentro de un equipo multidisciplinar de cinco alumnos de la universidad. A continuación, se detallan sus nombres y el enfoque de su trabajo de fin de grado; con el fin de obtener un mayor detalle del videojuego se recomienda examinarlos:

— Alejandro Jiménez Carrasco, estudiante del Grado en Diseño y Tecnologías Creativas, centra su memoria en el desarrollo del apartado artístico [3].
— Adrián Reina Sáez, estudiante del Grado en Diseño y Tecnologías Creativas, enfoca su trabajo en la creación de las animaciones e interacción del usuario [4].
— Tomás Ruiz Martín, estudiante del Grado en Ingeniería Informática, describe en su memoria la implementación de las mecánicas de juego [5].
— Pablo López Orrios, estudiante del Grado en Ingeniería Informática, pone énfasis en el desarrollo de la inteligencia artificial de los personajes [6].

1.1. Motivación

La principal motivación es la creación de un videojuego pulido y suficientemente entretenido para conseguir que su público objetivo experimente la mayoría de las funcionalidades y eventos desarrollados. Como jugadores asiduos, y en especial del género de juego desarrollado, siempre había sido de gran interés comprobar y experimentar en primera persona el ambiente y complejidades de un trabajo de esta índole, en definitiva, de responder a la pregunta de cómo se

[1] Comúnmente conocido como *indie*, en el contexto de los videojuegos son juegos creados por un grupo reducido de personas (usualmente menor de 10) y sin financiación de una editora de videojuegos.
[2] Versión de un producto que contiene las funcionalidades y características más relevantes de modo que se pueda evaluar la idea original por un público objetivo en fases tempranas del desarrollo.

hace un videojuego, sobre todo teniendo como referentes otros videojuegos que habían sido fuente de inspiración. Era también de especial interés participar en un proyecto de este tipo dada la oportunidad de centrarse en secciones concretas del juego, ya que se trataba de un equipo pequeño. Dado que se trataba de un equipo multidisciplinar también estuvo presente la esperanza de aportar y recibir ideas de ámbitos distintos a los que se han cursado y también de experiencias jugables de otros referentes, enriqueciendo así el producto final.

Más allá de la ilusión personal de crear un videojuego con salida al mercado, son también relevantes sus efectos colaterales, en especial el control y la planificación de un proyecto de emprendimiento con la posibilidad de fundar una empresa. Como jugadores y participes pasivos de la industria del videojuego, la meta de acabar formando una empresa y poder vivir de los ingresos de nuestras creaciones era una motivación muy presente.

Por otro lado, el diálogo en videojuegos es una funcionalidad presente que el público da por sentado, pero detrás de ella hay dificultades y peculiaridades que hacen su uso agradable y poco intrusivo, sobre todo en novelas visuales o en juegos donde, generalmente, existe un extenso guion que el jugador recibe a base de pequeñas dosis en forma de conversaciones o cinemáticas de juego. Por ello era interesante centrarse en las tareas relacionadas con el diálogo y experimentar con las diferentes implementaciones existentes en el mercado, así como construir a su vez sobre ellas tanto para asemejarse a estilos que habíamos visto en otros juegos como para incluir funcionalidad nueva con la finalidad de dar vida a nuestras ideas.

1.2. Objetivos

Los objetivos generales de este trabajo de fin de grado son los siguientes:

- Realizar un segundo producto mínimo viable del videojuego *Frozen Out*.
- Expandir el sistema de diálogos existente del primer producto mínimo viable.

A partir de estos, se pueden discernir los objetivos específicos:

- Obtener retroalimentación del segundo experimento por parte de su público objetivo con el fin de construir un mejor producto.
- Analizar la viabilidad de la idea original para reevaluar el seguimiento del proyecto.
- Ampliar los estilos y opciones de texto soportados para mostrar el diálogo en pantalla.
- Crear perfiles de personajes, de modo que se pueda configurar su estilo de diálogo.
- Expresar emociones y el tono del discurso a través de efectos aplicados en tiempo de ejecución.
- Definir mecanismos para posibilitar la internacionalización de los diálogos.
- Introducir opciones de diálogo abiertas accesibles mediante la interacción del usuario.
- Integrar herramientas que permitan leer el texto en pantalla y añadir opciones de configuración.
- Conectar el diálogo de personajes con redes sociales para favorecer la mercadotecnia del juego.

1.3. Estructura

La estructura de esta memoria se divide en dos bloques: el bloque de emprendimiento y el bloque de diálogos. El primer bloque, que comprende el segundo y tercer capítulo, se centra en el comentario de los aspectos propios de un proyecto de emprendimiento. El segundo capítulo habla de la generación de la idea de negocio, explicando el contexto y particularidades en las que se

creó la idea original; y la evaluación de la idea, donde se aplicarán técnicas de estudio de mercado para valorar la situación del equipo y de la idea frente a su competencia. En el tercer capítulo se describirá el desarrollo de la idea, entrando en detalle en las características consideradas inicialmente y el proceso de elaboración de estas funcionalidades en dos productos mínimos viables, así como su evaluación por su público objetivo y mecanismos para difundir el producto.

El segundo bloque, contenido en el cuarto y quinto capítulo, indagará en los aspectos técnicos relativos al desarrollo de las funcionalidades relacionadas con el diálogo. El cuarto capítulo habla sobre las extensiones que recibió, durante el segundo MVP, el sistema de diálogos inicial para convertirse en un gestor de diálogos que cumpliera con los objetivos del proyecto. En el quinto capítulo se mostrarán los diferentes subsistemas dentro de contextos reales del videojuego para comprobar su usabilidad.

Finalmente, en el sexto capítulo, se introducirán las conclusiones derivadas del trabajo en equipo e individual presentado y, en el séptimo capítulo, se describirá el trabajo futuro que pueda emerger a partir del desarrollo descrito.

2. Generación y evaluación de la idea de negocio

En esta sección se describirá la idea de negocio, así como aplicar diferentes técnicas para evaluar en profundidad la viabilidad de esta idea, como el análisis DAFO o el Lean Canvas. Tras el análisis de su resultado, se elegirá un modelo de negocio que se ajuste al marco en el que nos movemos debido a la naturaleza de nuestro producto: un videojuego.

2.1. Generación de la idea de negocio

La idea del proyecto se originó durante el transcurso de la asignatura Introducción a la Programación de Videojuegos, durante la cual se formaron grupos y se introdujeron estudiantes de la asignatura Desarrollo de Videojuegos (del grado de Diseño y Tecnologías Creativas en la facultad de Bellas Artes). Al principio de la asignatura se pidió a cada alumno, de ambas facultades, la propuesta de una idea para videojuego en formato *pitch doc*[3]. De todas las propuestas los profesores escogieron siete y formaron grupos de seis a siete alumnos, en base a nuestras preferencias, con el objetivo de desarrollar la idea propuesta, la cual sería liderada por el alumno que originó la idea.

En nuestro caso, la idea original consistía en un juego de puzles y aventuras en tercera persona con un parte de exploración [7], donde uno de los atractivos sería la historia: ambientada en una distopía[4] polar protagonizada por polos y helados con una clara temática medioambiental. La metáfora de juego deseada es la representación de los problemas de la crisis medioambiental y los vicios del modelo productivo que la han perpetrado. La idea estaba pensada para todos los públicos. Los referentes a la hora de desarrollar la idea fueron *Zack and Wiki*[5], un videojuego 3D en tercera persona publicado en 2008 para la plataforma Nintendo Wii[6], donde se siguen las aventuras del personaje Zack en una isla perdida, la cual sirve como entorno de juego.

2.2. Estudio de mercado

Sobre el ejercicio del año 2019, la facturación digital de videojuegos en España ha crecido un 6,6 %, mientras que la facturación física y la total ha decaído un 11,2 % y un 3,3 % [1], respectivamente. Estas cifras justifican la indiferencia en torno a la creación de una edición física, centrando los recursos y el esfuerzo en la edición digital. Así pues, los estudios independientes de videojuegos suelen optar por esta opción debido al bajo coste de producción. En cuanto al público objetivo el género aventura es el cuarto más popular, con 1.064.891 de unidades vendidas. El producto por desarrollar no pretende encontrar un vacío en el mercado, sino explotar un mercado existente y sobreponerse en cuestiones de historia y jugabilidad sobre sus competidores directos.

Atendiendo a datos del resto del mundo, España está situado en el décimo puesto dentro de los países en los que la industria del videojuego más factura, con unas cifras de 2.656 millones de dólares [8]. Mientras que China se encuentra en el primer puesto con unas cifras de 40.854 millones de dólares, debido principalmente al auge de los juegos para dispositivos móviles, los

[3] Documento corto empleado para la presentación de una idea ante futuros inversores o interesados.
[4] Representación ficticia de una sociedad futura de características negativas causantes de la alienación humana. Extraído de <https://dle.rae.es/distop%C3%ADa>
[5] Más información en <https://en.wikipedia.org/w/index.php?title=Zack_%26_Wiki:_Quest_for_Barbaros%27_Treasure&oldid=937787682>
[6] Más información en <https://es.wikipedia.org/w/index.php?title=Wii&oldid=128666528>

cuales son gratuitos y presentan un modelo de negocio basados en microtransacciones[7] para desbloquear cosméticos o mejoras dentro del juego. Cabe también destacar las restricciones aplicadas por el gobierno chino [9], razón por la que no se podría confiar en este mercado para el lanzamiento de nuestro juego, ya que debería pasar por una serie de inspecciones para poder ser publicado en el país. Por otro lado, en EE. UU. se facturaron 36.921 millones de dólares, y, pese a tener grandes beneficios en el sector móvil, los juegos en las demás plataformas siguen teniendo un papel principal en la industria. Por tanto, se diferencian por tener una mayor proporción de jugadores para consola y computadoras, y son más susceptibles a formar parte del público objetivo.

Visto el panorama del mercado móvil y las grandes campañas de *marketing* elaboradas por las editoras de videojuegos, la opción más viable para un estudio de desarrollo pequeño e independiente para crear un juego sin cualidades de servicio ni recurrencia es optar por un sistema de microfinanciación como Kickstarter[8], donde se realizan campañas de corta duración (entre 1 y 60 días) para apoyar a un producto en desarrollo. Claros ejemplos de este modelo son *Moonlighter*[9] (2018) y *Summer in Mara*[10] (2020), los cuales tuvieron gran éxito en su campaña de Kickstarter. En concreto, *Moonlighter* obtuvo 134.276 $ de los 40.000 $ que pedían en la propuesta[11], y para el caso de *Summer in Mara* se superó en un 1000% el objetivo base de la propuesta[12], con 233.919 €. A continuación, se presentan siete videojuegos que entran en el género del producto a desarrollar y serían considerados competidores directos.

2.2.1. *A Short Hike*[13]

A Short Hike es un videojuego 3D en tercera persona publicado en 2019 para Windows, Linux, macOS y Nintendo Switch, desarrollado por una sola persona, Adam Robinson-Yu. Se trata de un juego del género aventura y plataformas para un jugador donde controlas a un personaje pájaro a través de una isla. Este personaje empieza una aventura por la isla para llegar al punto más alto de una montaña con el objetivo de conseguir señal de telefonía, durante esta travesía conoce a otros personajes y explora la naturaleza del entorno. Como se puede comprobar en la Figura 1, cuenta con una estética de baja resolución. El juego ha ganado premios en certámenes independientes, como el premio de la audiencia en el *Independent Games Festival*[14] de 2019, además de ser nominado en otros eventos como *D.I.C.E.*[15] o *Golden Joystick Awards*[16]. Además de mover al personaje en el entorno, el juego también ofrece otras mecánicas como saltar, escalar y planear, así como el uso de ítems esparcidos por el juego que permiten expandir sus acciones. Por otro lado, las conversaciones con otros personajes son usualmente de carácter humorístico y alegre.

[7] Modelo de negocio donde los usuarios pueden comprar objetos virtuales mediante el uso de pagos dentro de la aplicación.
[8] Más información en <https://es.wikipedia.org/w/index.php?title=Kickstarter&oldid=127304922>
[9] Más información en <https://es.wikipedia.org/w/index.php?title=Moonlighter&oldid=122617059>
[10] Más información en <https://www.mobygames.com/game/summer-in-mara>
[11] Disponible en <https://www.kickstarter.com/projects/digitalsun/moonlighter>
[12] Disponible en <https://www.kickstarter.com/projects/chibig/summer-in-mara-an-adventure-set-in-a-tropical-ocea>
[13] Más información en <https://en.wikipedia.org/w/index.php?title=A_Short_Hike&oldid=975096031>
[14] Más información en <https://es.wikipedia.org/w/index.php?title=Independent_Games_Festival&oldid=119050744>
[15] Más información en <https://en.wikipedia.org/w/index.php?title=D.I.C.E._Awards&oldid=972060055>
[16] Más información en <https://es.wikipedia.org/w/index.php?title=Premios_Golden_Joystick&oldid=128294657>

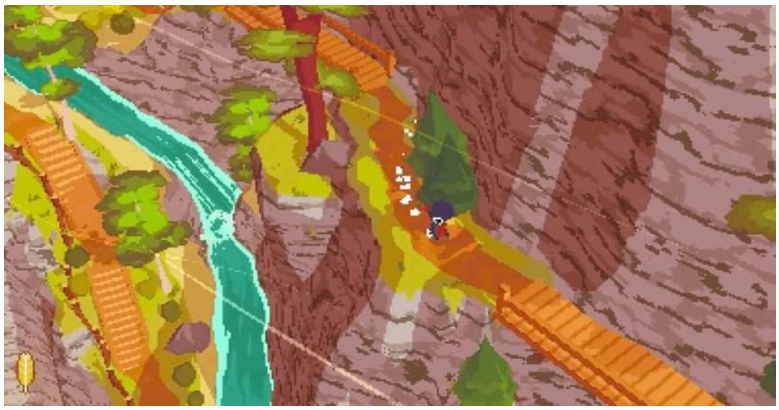

Figura 1. Imagen promocional de A Short Hike. A Short Hike (2019).

2.2.2. *Summer in Mara*

Summer in Mara es un videojuego de aventura y gestión de recursos para un solo jugador, desarrollado por Chibig[17] y publicado en 2020 para las plataformas Windows, PS4, Xbox One y Nintendo Switch. Los creadores realizaron una campaña de Kickstarter con éxito, con un total de 9523 patrocinadores y 233.919 € recaudados. El videojuego trata de una niña que vive en una isla, la cual deberá cuidar y hacer recados para los lugareños para así avanzar en la historia. Para ello, la protagonista es capaz de moverse libremente por el entorno para recoger recursos y crear nuevos objetos útiles a partir de estos. Es un juego en 3D (Figura 2) en el que explorar el entorno es una parte vital para encontrar aquello que necesitas.

Figura 2. Imagen promocional de Summer in Mara. Summer in Mara (2020).

2.2.3. *A Hat in Time*[18]

A Hat in Time es un videojuego de aventuras y plataformas desarrollado por Gears for Breackfast[19] y publicado en 2017 para las plataformas Windows, macOS, PS4, Xbox One y Nintendo Switch. Al igual que el anterior, también se desarrolló gracias a una campaña de Kickstarter[20], con 9169 patrocinadores y 296.360 $ recaudados. Cabe destacar que el equipo de desarrollo consistía en un equipo fijo y de voluntarios que se iban sumando al proyecto para sacarlo adelante. En el videojuego, el cual se puede ver en la Figura 3, el jugador controla el personaje principal a través de diferentes mundos de juego para recoger coleccionables que le permitirán visitar más niveles. Gracias a ellos tendrá acceso a diferentes movimientos y habilidades para moverse por las distintas plataformas.

[17] Más información en <https://chibig.com/>
[18] Más información en <https://es.wikipedia.org/w/index.php?title=A_Hat_in_Time&oldid=126612097>
[19] Más información en < https://gearsforbreakfast.com/>
[20] Disponible en <https://www.kickstarter.com/projects/jonaskaerlev/a-hat-in-time-3d-collect-a-thon-platformer>

Figura 3. Imagen promocional de A Hat in Time. A Hat in Time (2017).

2.2.4. *Spyro Reignited Trilogy*[21]

Spyro Reignited Trilogy es un videojuego 3D en tercera persona para un jugador publicado en 2019 para Windows, PS4, Xbox One y Nintendo Switch, desarrollado por Toys for Bob[22]. Se trata además de una remasterización[23] de una serie de juegos publicados entre los años 1998 y 2000. Se encuentra dentro del género de aventuras, puzles y plataformas, y sigue las andanzas de un dragón para restablecer la paz en una gran variedad de entornos y mundos. El juego permite controlar libremente al personaje principal, así como usar su habilidad especial para lanzar fuego como ataque para enemigos, tal como se ve en la Figura 4. El objetivo del juego es el de liberar otros dragones, pero también ofrece la posibilidad de recoger coleccionables esparcidos por los diferentes mundos que están conectados entre ellos por un mapa abierto. La serie de juegos originales gozó de popularidad en su momento, en gran parte al carisma de sus personajes y la estética de dibujo animado, además no estaban disponibles en hardware actual, de modo que era una revisión demandada dentro de la comunidad.

Figura 4. Imagen promocional de Spyro Reignited Trilogy. Spyro Reignited Trilogy (2019).

2.2.5. *Crash Bandicoot 4: It's About Time*[24]

Crash Bandicoot 4 es un videojuego 3D en tercera persona pendiente de publicarse en 2020 para PS4 y Xbox One, y desarrollado por Toys for Bob. Se trata de un juego de aventuras y

[21] Más información en <https://es.wikipedia.org/w/index.php?title=Spyro_Reignited_Trilogy&oldid=119196281>
[22] Más información en <https://www.toysforbob.com/>
[23] Recreación de un videojuego publicado con anterioridad para actualizarlo a nuevo hardware y gráficos modernos con el fin de hacerlo accesible a un público actual.
[24] Más información en <https://en.wikipedia.org/w/index.php?title=Crash_Bandicoot_4:_It%27s_About_Time&oldid=974546032>

plataformas, donde se controla a un animal humanoide a través de una multitud de entornos. El objetivo es derrotar a un malvado científico, superando y desbloqueando niveles que se seleccionan en un mapa. El juego permite controlar el personaje en diferentes estilos, intercalando secciones en 3D donde el movimiento es hacia dentro de la pantalla o hacia fuera (Figura 5) y secciones en 2D como un juego de plataformas clásico. Las tres primeras entregas, publicadas originalmente entre 1996 y 1998, recibieron una remasterización en 2017, recuperando así la base de usuarios y captando a nuevos interesados actuales.

Figura 5. Imagen promocional de Crash Bandicoot 4. Crash Bandicoot 4: It's About Time (2020).

2.2.6. *Yooka-Laylee*[25]

Yooka-Laylee es un videojuego de plataformas de mundo abierto desarrollado por Playtonic Games[26] y publicado en 2017 por Team17 en las plataformas Windows, Linux, macOS, PS4, Xbox One y Nintendo Switch. Los creadores realizaron una campaña de Kickstarter para financiar el juego[27], en esta participaron 73.206 personas las cuales contribuyeron 2.090.104 £. El jugador controla a Yooka en un mundo 3D para explorar una variedad de niveles (Figura 6) y resolver puzles en un mundo abierto. Esta exploración se realiza a través de las habilidades especiales que se desbloquean durante el progreso de la partida, los cuales permitirán avanzar y terminar la historia. Además, se hace énfasis en los coleccionables.

Figura 6. Imagen promocional de Yooka-Laylee. Yooka-Laylee (2017).

[25] Más información en <https://es.wikipedia.org/w/index.php?title=Yooka-Laylee&oldid=127889060>
[26] Más información en <https://www.playtonicgames.com/>
[27] Disponible en <https://www.kickstarter.com/projects/playtonic/yooka-laylee-a-3d-platformer-rare-vival>

2.2.7. *Night in the Woods*[28]

Night in the Woods es un videojuego de aventuras desarrollado por Infinite Fall, un estudio formado por tres personas y publicado en 2017 para las plataformas Windows, Linux, macOS y PS4, aunque más adelante también fue publicado para las plataformas Xbox One y Nintendo Switch. Los creadores realizaron una campaña de Kickstarter para financiar el juego[29], en esta participaron 7372 personas las cuales contribuyeron 209.375 $. El juego trata de una joven que vuelve a su pueblo tras irse a la universidad. Este intenta reunirse con sus amigos y recuperar su antigua vida, pero todo parece distinto debido al paso del tiempo. Centrado en la exploración, historia y personajes, se hace énfasis en los diálogos y las interacciones con los personajes, visible en la Figura 7, en un mundo en 2D con personajes y entornos originales.

Figura 7. Imagen promocional de Night in the Woods. Night in the Woods (2017).

[28] Más información en <https://es.wikipedia.org/w/index.php?title=Night_in_the_Woods&oldid=128563295>
[29] Disponible en <https://www.kickstarter.com/projects/1307515311/night-in-the-woods>

2.2.8. Resumen

Los videojuegos descritos sirven como análisis de mercado para comprobar con qué tipo de juegos puede llegar a competir nuestra propuesta. Por otro lado, también es útil para comprobar qué características son populares o esperadas por los usuarios de este género de videojuegos. A continuación, en la Tabla 1, se muestran las funcionalidades más relevantes de estos juegos para una comprensión más fácil y rápida. Además, adjuntamos nuestra propuesta a modo de comparativa crítica de las funcionalidades deseables.

Característica	*A Short Hike*	*Summer in Mara*	*A Hat in Time*	*Spyro Reignited Trilogy*	*Crash Bandicoot 4: It's About time*	*Yooka-Laylee*	*Night in the Woods*	*Frozen Out*
Plataformas	Windows Linux macOS	Windows Linux macOS PS4 Switch	Windows macOS PS4 XBO Switch	Windows PS4 XBO Switch	PS4 XBO	Windows Linux macOS PS4 XBO Switch	Windows Linux macOS PS4 XBO Switch	Windows Linux macOS
Número de jugadores	1	1	1-2	1	1	1-2	1	1
Precio de salida	6,59 €	21,99 €	27,99 €	39,99 €	-	39,99 €	19,99 €	19,99 €
Mecánicas jugables (pocas y funcionales)	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Duración aprox.	3 h	25 h	10 h	15 h	-	15 h	8 h	8 h
Ambientación original	Sí	No	Sí	Sí	No	No	Sí	Sí
Diálogo entretenido	Sí	No	Sí	No	-	Sí	Sí	Sí
Estilo artístico simple	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Crítica medioambiental	No	No	No	No	-	No	No	Sí
Simplicidad de niveles	No	Sí	No	Sí	-	No	Sí	Sí
Centrado en exploración del entorno	Sí	Sí	Sí	Sí	No	Sí	No	Sí
Uso de ítems	Sí	Sí	No	No	No	No	No	Sí
Coleccionables	Sí	No	Sí	Sí	Sí	Sí	No	No

Tabla 1. Tabla comparativa de juegos competidores con la propuesta a desarrollar. Elaboración propia.

Examinando la tabla se puede extraer que un juego de plataformas y aventuras debe tener una ambientación original, usualmente dentro de un mundo fantástico, con personajes carismáticos y

con un estilo de dibujo animado o con diseños simples, donde gran parte del encanto viene de la interacción con ellos a través de los diálogos, dejando también sitio para la crítica en estos. En cuestión de mecánicas, aun siendo simples deben permitir al jugador poder explorar el entorno de forma progresiva. Al igual que en nuestro proyecto, los esfuerzos en la mayoría de los casos se centran en crear una aventura para un jugador. En cuanto a las plataformas donde se encuentran disponibles, visto la presencia de consolas es una buena idea considerar en un futuro estas plataformas, ya que usualmente estos juegos consideraron las consolas una vez tuvieron una base de jugadores en computadoras. También suele ser habitual el añadido de coleccionables para incentivar una segunda partida, sin embargo, en nuestro caso tendrían el papel de ampliar el elenco de habilidades o mejorarlas, como en el caso de *A Short Hike* y *Yooka-Laylee*.

2.3. Análisis DAFO

El análisis DAFO permite a cualquier proyecto conocer sus puntos fuertes y débiles, de modo que el equipo de desarrollo puede centrar sus esfuerzos en aquellos aspectos que harán su producto destacar, además de conocer de antemano en qué secciones se va a ver perjudicado por su competencia y debería minimizarse su impacto. A continuación, se ofrece en la Tabla 2 la matriz realizada para este proyecto usando esta técnica:

Debilidades	**Amenazas**
● Escaso contenido al principio. ● Falta de financiación. ● Equipo inexperto. ● Poca presencia online. ● Primer producto en el mercado. ● No presente para plataformas móviles ni consolas.	● Exceso de lanzamientos de juegos independientes en plataformas digitales. ● Competencia asentada. ● Mayor duración de los juegos. ● Auge de la popularidad de juegos como servicio.
Fortalezas	**Oportunidades**
● Equipo multidisciplinar. ● Entorno estudiantil. ● Soporte de profesorado experto. ● Participación en ferias universitarias. ● Idea innovadora. ● Personajes y entorno del juego originales. ● Disponibilidad de espacio de trabajo.	● Jugadores en busca de contenido diferente. ● Jugadores con poco tiempo en busca de experiencias cortas. ● Comunidad centrada en juegos independientes.

Tabla 2. Matriz DAFO para el proyecto Frozen Out y sus componentes. Elaboración propia.

El proyecto se va a realizar bajo el marco de la universidad, de modo que va a existir una amplia cantidad de recursos disponibles, además de la ayuda directa o indirecta de profesorado experto a través de seminarios y la asistencia a ferias creadas por la comunidad universitaria. Cabe destacar también la disponibilidad de espacios de trabajo por parte de la universidad para casos como el nuestro, en el que un grupo de estudiantes decide crear un proyecto de emprendimiento. Además, la variedad de personalidades y especialidades en los componentes del grupo de trabajo permite crear un entorno donde contrastar diferentes ideas y puntos de vista. Creemos que se trata también de una idea innovadora, dado que no hemos encontrado ejemplos de juegos protagonizados por helados y polos, y con una crítica medioambiental, razón por la cual esta será una de las características que reforzaremos en la campaña de mercadotecnia.

Por otra parte, el hecho de ser la primera toma de contacto del equipo con la industria conlleva una serie de problemas: falta de presupuesto, poca presencia destacable en redes y baja experiencia en las mismas. Además, entre tanta cantidad de lanzamientos de juegos, sobre todo videojuegos independientes, es bastante común que productos prometedores se vean enterrados entre tantos productos disponibles. En caso de no ser así, también habría que tener en cuenta la cantidad de tiempo empleado por los jugadores en juegos como servicio, hecho que limita su disponibilidad para probar otros juegos.

2.4. Modelo de negocio y proyección económica

A la hora de crear una nueva empresa, al ser el objetivo de un proyecto de emprendimiento, es de alta relevancia analizar la proyección económica con tal de revisar la viabilidad del proyecto durante su desarrollo y comprobar de manera aproximada los gastos e ingresos que pueda generar el proyecto. En cuanto al modelo de negocio propuesto para nuestro videojuego, se va a optar por un precio de licencia único de 19,99 €, ya que en los juegos con una historia cerrada los consumidores prefieren un precio final para despreocuparse de ítems y contenido adicional que pueda crear experiencias distintas. La razón de este precio es por la relación de contenido y duración con los ejemplos mostrados durante el estudio de mercado, concretamente con *A Hat in Time* y *Yooka-Laylee*. Aun así, no se descarta la creación de expansiones que exploren otras historias dentro del entorno creado.

Las plataformas de venta elegidas para su distribución van a ser primariamente plataformas para computadoras, ya que la publicación de videojuegos para estos sistemas requiere de una inversión nula o más reducida en comparación con las plataformas de consola, donde existe una única plataforma digital de venta y es controlada por las compañías que desarrollan las consolas. Esto también se debe a la libertad de desarrollo para sistemas de computadoras, ya que la creación del ejecutable para consolas requiere de un permiso de desarrollador de las compañías propietarias de dichas consolas. Además, se descarta la publicación física debido, por una parte, a la decaída de las ventas físicas durante los últimos años [1], y, por otra parte, a la necesidad de contratar los servicios de una editora de videojuegos. De este modo, las plataformas consideradas son: Steam[30], la cual cuenta con un alto porcentaje de adopción en los consumidores de computadora [10]; y itch.io[31], una plataforma dirigida a pequeños estudios y desarrolladores independientes.

A continuación, se muestra la Tabla 3 con la proyección económica a tres años vista dividida en trimestres, así como su representación gráfica (Figura 8). En ella se muestra el balance durante este periodo y cómo se encontrarían las cuentas del equipo a raíz del modelo de negocio elegido, las decisiones tomadas, las plataformas en las que se comercializará y el número de gente que se encargará de terminar con su desarrollo tras el segundo MVP.

Es importante remarcar la fórmula utilizada para calcular los ingresos por licencia: por una parte, están divididos en las principales plataformas de venta consideradas, valorando que en Steam se realizarán el 85 % de las ventas totales, y, el resto, por consiguiente, en itch.io; por otra parte, se debe tener en cuenta las comisiones de venta que se lleva cada plataforma, siendo de un 30 % por cada venta en Steam [11] y del 10 % en itch.io, aunque esta plataforma permite la asignación libre de esta comisión [12], pudiendo incluso ser nula. Por tanto, la fórmula para cada plataforma tiene en cuenta las licencias estimadas a vender en ese trimestre multiplicado por el precio del producto (19,99 €), la comisión de venta y el porcentaje de ventas dirigido a esa plataforma.

[30] Más información en <https://es.wikipedia.org/w/index.php?title=Steam&oldid=126892293>
[31] Más información en <https://es.wikipedia.org/w/index.php?title=Itch.io&oldid=119199832>

	Trim. 1 año 1	**Trim. 2 año 1**	**Trim. 3 año 1**	**Trim. 4 año 1**	**Trim. 1 año 2**	**Trim. 2 año 2**	**Trim. 3 año 2**	**Trim. 4 año 2**	**Trim. 1 año 3**	**Trim. 2 año 3**	**Trim. 3 año 3**	**Trim. 4 año 3**
Licencias	0	0	0	0	2500	500	10000	3000	200	200	100	50
Ingresos trimestrales												
Licencias Steam	0,00 €	0,00 €	0,00 €	0,00 €	29.735,13 €	5.947,03 €	118.940,50 €	35.682,15 €	2.378,81 €	2.378,81 €	1.189,41 €	594,70 €
Licencias itch.io	0,00 €	0,00 €	0,00 €	0,00 €	6.746,63 €	1.349,33 €	26.986,50 €	8.095,95 €	539,73 €	539,73 €	269,87 €	134,93 €
Total ingresos	0,00 €	0,00 €	0,00 €	0,00 €	36.481,75 €	7.296,35 €	145.927,00 €	43.778,10 €	2.918,54 €	2.918,54 €	1.459,27 €	729,64 €
Gastos trimestrales												
Gastos GitHub	13,23 €	13,23 €	13,23 €	13,23 €	13,23 €	13,23 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Licencia Steam Direct	0,00 €	0,00 €	0,00 €	0,00 €	88,20 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Mercadotecnia	0,00 €	0,00 €	0,00 €	0,00 €	1.000,00 €	500,00 €	100,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Empleado	18.000,00 €	18.000,00 €	18.000,00 €	18.000,00 €	18.000,00 €	18.000,00 €	18.000,00 €	18.000,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Total gastos	18.013,23 €	18.013,23 €	18.013,23 €	18.013,23 €	19.101,43 €	18.513,23 €	18.100,00 €	18.000,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Personal contratado	3	3	3	3	3	3	3	3	0	0	0	0
Resultado trimestral	-18.013,23 €	-18.013,23 €	-18.013,23 €	-18.013,23 €	17.380,32 €	-11.216,88 €	127.827,00 €	25.778,10 €	2.918,54 €	2.918,54 €	1.459,27 €	729,64 €
Resultado trimestral acumulado	-18.013,23 €	-36.026,46 €	-54.039,69 €	-72.052,92 €	-54.672,60 €	-65.889,48 €	61.937,52 €	87.715,62 €	90.634,16 €	93.552,70 €	95.011,97 €	95.741,60 €

Tabla 3. Proyección económica del proyecto a tres años. Elaboración propia.

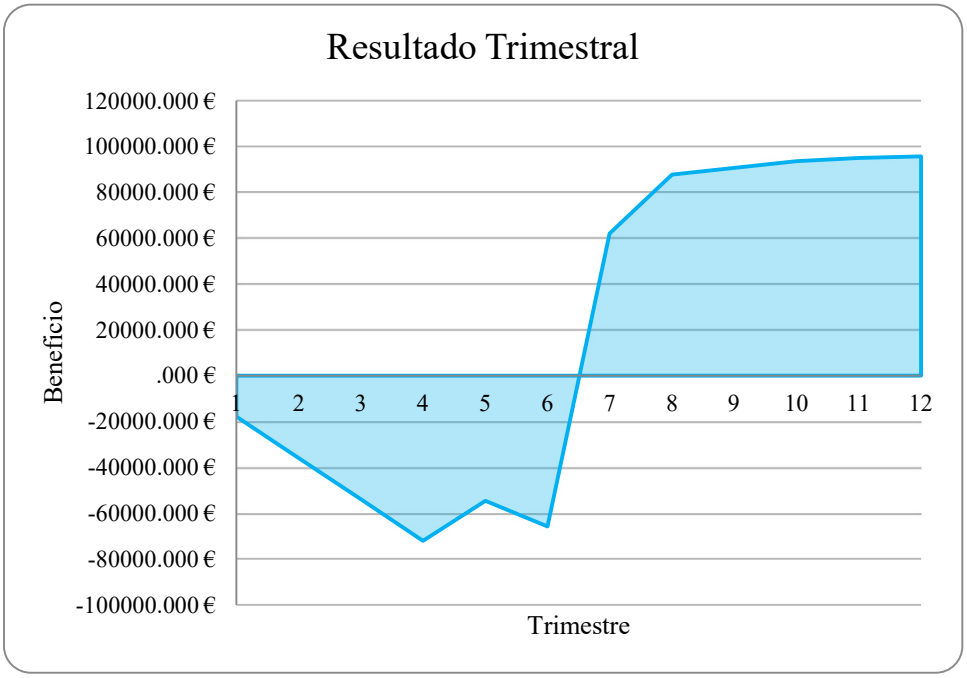

Figura 8. Gráfica sobre el resultado dividido en trimestres. Elaboración propia.

Para financiar el proyecto, se plantea el uso de la plataforma Kickstarter. En nuestro caso, y como se puede comprobar en la Figura 8, nos planteamos una campaña de 30 días con una meta de 75.000 €. Esta elección se debe a que, con un tiempo de desarrollo estimado de un año y nueve meses, el total acumulado tendría un máximo de 72.052,92 €. La cifra impuesta se justifica con el pago de los sueldos de los trabajadores, la necesidad de pagar la licencia necesaria para publicar un juego en la plataforma digital de Steam, las tasas a pagar a Kickstarter por haber percibido ingresos a través de su plataforma y tener así un margen para posibles imprevistos.

El desarrollo de los primeros MVP supondrá la inclusión de los aspectos clave del proyecto, conocido comúnmente como *vertical slice*[32], en el cual se desarrollan por completo las funcionalidades necesarias para un nivel, tal que se pueda jugar en su totalidad como si formara parte del producto final. Esta técnica fuerza al equipo de desarrollo a describir e implementar los sistemas base, así como un marco operativo y funcional donde conectar dichos sistemas, de modo que, al finalizar este nivel, la creación de los subsecuentes niveles se va a ver beneficiada de este trabajo existente. Por estas razones se estima que el tiempo de desarrollo para terminar el producto, al finalizar el segundo MVP, va a ser de año y medio, y además durante este periodo el equipo de desarrollo va a ser más reducido, de tres personas. Al finalizar esta etapa, el desarrollo se puede dar como finalizado, momento en que los trabajadores podrán ser recolocados en otros proyectos (que se encontrarán en la fase de preproducción o en proceso de salir de esta), tal como se puede apreciar a partir del tercer trimestre del segundo año.

Examinando con más detalle el desglose de gastos e ingresos mostrados, se puede comprobar que, durante el primer año, el trabajo se centra en el desarrollo del producto, de modo que los gastos son derivados de las remuneraciones de los empleados, en concreto de pagar tres empleados un sueldo de 2000 €, y del coste de herramientas para control de versiones, con un coste de 5 $ mensuales[33], con el objetivo de poder almacenar el proyecto y los archivos binarios de gran

[32] Más información en <https://en.wikipedia.org/w/index.php?title=Vertical_slice&oldid=962437839>
[33] Más información en <https://docs.github.com/es/github/setting-up-and-managing-billing-and-payments-on-github/about-billing-for-git-large-file-storage>

tamaño en el servicio online de GitHub. De este modo, al finalizar el primer año se tendría un balance negativo de 72.052,92 € sin la inversión inicial de Kickstarter, suponiendo el pico de perdidas.

Durante el segundo año se empezarán a realizar los preparativos para llevar el juego a las plataformas de juego y tiendas digitales, en concreto se debería abonar la tasa única de entrada en la tienda de Steam de 100 $ [13] (en el momento de redacción la conversión es a 88,20 €). Con esta entrada temprana, se podría abrir un periodo de preventa en la que se pueden conseguir ingresos adicionales de consumidores interesados que no pudieron participar en la campaña de Kickstarter. Con este método se estima obtener 3000 licencias adicionales, vendidas al precio habitual de 19,99 €. Además, la campaña de venta y preventa se verán ayudadas por la visibilidad producida gracias a la campaña de mercadotecnia, a la que se dedicará un total de 400 € repartidos sobre todo en los primeros trimestres: creación de videos promocionales y cumplimiento con las recompensas de los patrocinadores de la campaña de Kickstarter. A partir del tercer trimestre, el juego habría salido formalmente a la venta, momento en el que se pronostica el mayor pico de ventas durante las primeras semanas, de unas 10.000 licencias. Cabe señalar que se mantienen los tres empleados hasta final del año para cubrir el mantenimiento y resolución de los mayores problemas durante los primeros meses de lanzamiento. Adicionalmente, durante el cuarto trimestre se prevé la venta de unas 3000 licencias más.

Durante el tercer año, los trabajadores ya habrían sido reubicados a otros proyectos y el producto existente en el mercado habría alcanzado un nivel de madurez y estabilidad aceptable para poder dejar de lado su mantenimiento. Por tanto, en este periodo solo habría ventas, ayudadas sobre todo por temporadas de rebajas y ofertas presentes en las plataformas de venta, aunque cabe destacar que las ventas trimestrales serían de menor cantidad y decrecientes debido a la alta competencia y a la pérdida de las cualidades novedosas del producto.

Como conclusión de la proyección económica deducimos que, dado la naturaleza del producto, solo generaría beneficios una vez sea lanzado al mercado, decreciendo con el tiempo los ingresos que este pueda producir, a diferencia de los juegos como servicio que generan ingresos a lo largo de su vida, pero dedicando el equipo recursos en sacar nuevo contenido de forma periódica. En nuestro caso, el lanzamiento de este producto, aunque no aporte un gran beneficio, supone la entrada del equipo al mercado y, por consiguiente, abre la posibilidad a la creación de otros productos de tamaño similar, que en conjunto sí podrían ofrecer una fuente de ingresos estable. Esta estrategia también serviría, en vistas a futuro y en base a sus ventas, para financiar nuevos proyectos de mayor envergadura que requieran de más tiempo y dedicación, dejando de depender de técnicas de microfinanciación colectiva como el mencionado Kickstarter.

2.5. Lean Canvas

El Lean Canvas es una herramienta que permite analizar el modelo de negocio escogido para aumentar las probabilidades de tener éxito. Esta técnica consiste en enumerar los valores de un producto o de una idea de negocio, tal que queden claros en una tabla que aspectos del producto son diferenciadores y únicos en el mercado en cuestión, así como las diferentes formas en la que se puede dar a conocer el producto. Además, permite distinguir el público objetivo y el tipo de consumidor que pueden servir para probar el producto en sus fases más tempranas, llamados *early adopters*. La sección de costes e ingresos se puede observar como un resumen de los factores más relevantes destacados en el apartado de proyección económica. En definitiva, ofrece, de forma compacta, las características clave del producto con el objetivo de compartir y difundir la idea de negocio a posibles interesados. A continuación, en la Tabla 4, se puede apreciar el Lean Canvas elaborado para este proyecto.

2 Problema

Se encontró en el mercado escasos juegos con temáticas éticas, así como universos que apuesten por una narrativa trabajada en este aspecto o con protagonistas no humanoides.

Cada vez hay más tendencia en la industria a hacer los juegos más largos y con más contenido, suponiendo un gran esfuerzo para el jugador terminar la historia.

Tendencia a realizar juegos como servicio a los que hay que dedicar mucho tiempo para progresar.

4 Solución

Historia ambientada en un mundo distópico, donde tu misión es paliar las consecuencias de la crisis climática.

Contenido cerrado y de corta duración, que se pueda finalizar de una sentada.

8 Métricas

- Valoraciones de la página del producto en sus plataformas de venta (Steam y itch.io).
- Formularios y cuestionarios realizados en ferias.
- Impacto social en redes sociales.

3 Proposición de valor

Videojuego narrativo con temática del cambio climático.

La historia busca concienciar y ridiculizar a partes iguales los vicios de nuestra sociedad acerca de la crisis climática, así como presentarnos una ciudad distópica habitada por helados.

Estilo artístico *low-poly* [34] y desenfadado.

Inclusión de mecánicas simples, pero con gran variedad de uso.

9 Ventaja competitiva

Entorno de juego y propuesta original, ya que se ambienta en un mundo habitado por helados.

5 Canales

- Portales de distribución de juegos (itch.io, Steam).
- Ferias y eventos (Feria de proyectos de estudiantes).
- Redes sociales (Instagram, Twitter).

1 Clientes

Público objetivo:

Se busca un perfil casual, es decir, gente interesada en videojuegos o en entrar en el medio, y por tanto no necesariamente muy habilidosos con los mismos.
Interesados en la problemática social del cambio climático.
Interesados en experiencias dirigidas a un solo jugador.

Early adopters:

Familiares, amigos, patrocinadores relevantes y activos de la campaña de Kickstarter.

7 Costes

Sueldos de trabajadores contratados para finalizar el desarrollo y costes de publicidad para mejorar la visibilidad y aumentar la cantidad de potenciales compradores.
Licencias de herramientas de desarrollo como GitHub LFS.
Tasas de publicación de juegos en plataformas de venta, concretamente Steam Direct.

6 Ingresos

Ventas, tanto provenientes de campañas de preventa como ya en su salida al mercado. Ingresos derivados de la campaña de microfinanciación en Kickstarter.
Donaciones provenientes de la campaña de Kickstarter o posterior a ella.
Financiación o aportación adicional de los componentes del grupo.

Tabla 4. Lean Canvas realizado para el proyecto. Elaboración propia.

[34] Estilo o técnica de modelado basada en un bajo uso de polígonos a la hora de construir los modelados. Más información en <https://en.wikipedia.org/w/index.php?title=Low_poly&oldid=973057569>

2.6. Conclusiones

Tras finalizar la evaluación de la idea original a través de diferentes técnicas, el equipo obtuvo las siguientes consideraciones: el equipo es consciente de los puntos fuertes del videojuego en cuestión, especialmente su historia y entorno únicos, de modo que serían aspectos por destacar en el juego y, sobre todo, en las campañas de publicidad; la baja experiencia en el mercado del equipo es uno de los aspectos que más preocupan, es por ello que se debería apostar por crear una experiencia corta y satisfactoria; similarmente, se debería centrar el desarrollo para su salida en computadoras y, una vez comprobado su recibimiento, considerar su salida en otras plataformas, ya que requerirían un periodo adicional de aprendizaje, registro en plataformas de desarrollo de consolas y adaptación de las mecánicas para otras formas de juego.

Por otro lado, el género del juego a desarrollar no está entre los más rentables y la competencia para cualquier tipo de juego en la actualidad no permite tampoco a un estudio recién creado depender de su primer producto, es por esto que se deberá recurrir a la ayuda de campañas de financiación colectiva para poder cubrir los gastos necesarios para finalizar el producto. Además, será esencial, una vez sacado al mercado, la planificación de otros proyectos de similar envergadura con el objetivo de conseguir un grupo de productos que permitan: por una parte, sostener al equipo y poder contratar nuevo personal que ayude a realizar los siguientes trabajos; y, por otro lado, subvencionar los subsecuentes proyectos, disminuyendo así la dependencia en las plataformas de financiación colectiva. En cualquier caso, el aumento de juegos realizados conseguirá que el estudio consolide un lugar en el mercado, hecho que permitirá el crecimiento de la empresa, pudiendo incluso llegar a realizar desarrollos más largos y atraer a editoras con el porfolio de los productos creados, las cuales posibilitarían la comercialización de ejemplares físicos gracias su apoyo.

3. Desarrollo de idea de negocio

El desarrollo de este proyecto ha tenido una duración de nueve meses, periodo de tiempo en el que también se incluyeron las distintas pruebas con usuarios. La asignatura de Introducción a la Programación de Videojuegos sirvió de marco inicial para este desarrollo; donde se nos proporcionó la oportunidad de juntarnos con alumnos del grado de Diseño y Tecnologías Creativas. Tras la sesión en la que se presentaron las distintas ideas propuestas por los alumnos de ambas asignaturas, se creó, a mediados de octubre, el equipo en base a nuestras preferencias, con el objetivo de no solo ser evaluado en la asignatura sino también presentar el resultado en la «Feria de proyectos de estudiantes» de la ETSINF que se celebraría en diciembre de 2019, momento en el que tendríamos nuestra primera toma de contacto con posibles usuarios fuera del aula.

Tras reunirnos, el equipo comenzó a idear que podría hacer el protagonista, un polo: se comparó con distintos juegos para ver dónde se podía sacar inspiración y además desarrollar elementos que diferenciaran este proyecto del resto para hacerlo atractivo. Después de varias reuniones se asentaron algunas mecánicas básicas, la estructura del entorno y qué características tendría. Durante el mes de octubre se elaboró un tablero Kanban[35] usando la herramienta colaborativa Trello[36] con las diferentes características a implementar, de esa forma se podría realizar un seguimiento de las tareas, así como asignar responsabilidades a cada integrante. El primer MVP tuvo lugar entre los meses de octubre y diciembre de 2019, coincidiendo con la asignación de los equipos y la celebración de la feria, respectivamente.

Para la muestra en la feria de diciembre se elaboró una encuesta que pudiera proporcionar datos significativos acerca de la experiencia de los usuarios con la demo desarrollada. Estos hicieron al equipo reflexionar acerca de la experiencia y los resultados, obteniendo como resultado la necesidad de replantear la disposición del mapa a mostrar al usuario y cómo este interactuaría con el entorno. Tras esto, se dejó un tiempo para reflexionar al equipo y decidir si seguir con el proyecto.

Después del periodo de reflexión, se decidió continuar con el proyecto, de modo que nos volvimos a reunir, con un integrante menos, para decidir qué cambios aplicábamos de cara a un segundo MVP. Tras una serie de reuniones, empezamos su desarrollo, sin embargo, debido a la pandemia del COVID-19[37], tuvimos que cambiar nuestro modo de trabajo: por un lado, una coordinación desde nuestras casas y reuniones telemáticas, realizando sesiones de trabajo conjuntas a través de servicios en línea; y, por otro lado, una preparación de los mecanismos de difusión digitales para evitar el contacto físico y recoger las experiencias de los usuarios de forma digital. Este segundo periodo de desarrollo ocurrió entre los meses de febrero y junio de 2020.

[35] Más información en <https://www.atlassian.com/agile/kanban/boards>
[36] Más información en <https://es.wikipedia.org/w/index.php?title=Trello&oldid=121097153>
[37] Más información en <https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019>

3.1. Mapa de características

Al inicio del proyecto, el equipo de trabajo discutió las características que iban a conformar el videojuego, con este propósito se elaboró una lista de unidades de trabajo[38], de ahora en adelante UT, tal que pudieran formar parte de la bolsa de trabajo[39] del proyecto, la cual se puede ver en la Figura 9. Basándonos en la jugabilidad deseada para el juego, se incluyeron varias UTs relacionadas con el movimiento y las mecánicas de juego (salto, sigilo, forma de sorbete y cambio de palos para conseguir nuevas habilidades). También se incluyeron tareas relativas a los sistemas con el que el jugador podría interactuar dentro del entorno de juego (diálogos, inventario, misiones, cámara, patrullas y seguimiento de enemigos, zonas de calor y muerte), así como los menús que posibilitarían navegar hacia los niveles y configurar el juego (menú de pausa, menú principal, menús de configuraciones, guardado y cargado de partidas). Con el objetivo de hacer el juego más accesible y ayudar a nuevos jugadores se incluyeron diversas tareas para introducir: soporte para idiomas castellano e inglés, cinemáticas de introducción, cinemáticas de explicación, minimapa[40] y una flecha para indicar el siguiente objetivo.

El desarrollo de los niveles que compondrán el juego en un futuro se representó con las tareas de niveles exteriores y niveles interiores (entendidos en referencia a la nevera, un punto central en el juego donde el jugador podrá entrar), sin embargo, el desarrollo del primer nivel (exterior) ya se ha colocado como UT propia al ser el más relevante para la evaluación de la idea inicial. Esta tarea se consideró demasiado genérica, de modo que se desglosó en tres: creación del mapa nivel uno, eventos nivel uno y personajes nivel uno. Para tener en cuenta la creación de componentes artísticos y modelados, se incluyeron UTs para objetos que aparecerían en la mayoría de los niveles (objetos del entorno y objetos clave) y paralelamente, tareas para interactuar con los componentes del entorno de juego (interacción con personajes). Adicionalmente, se incluyeron tareas de componentes que establecerían el tono del juego y que contribuirían a su apartado artístico (postprocesado, animaciones, efectos del entorno, pisadas en la nieve, estilos de texto y voces de diálogo).

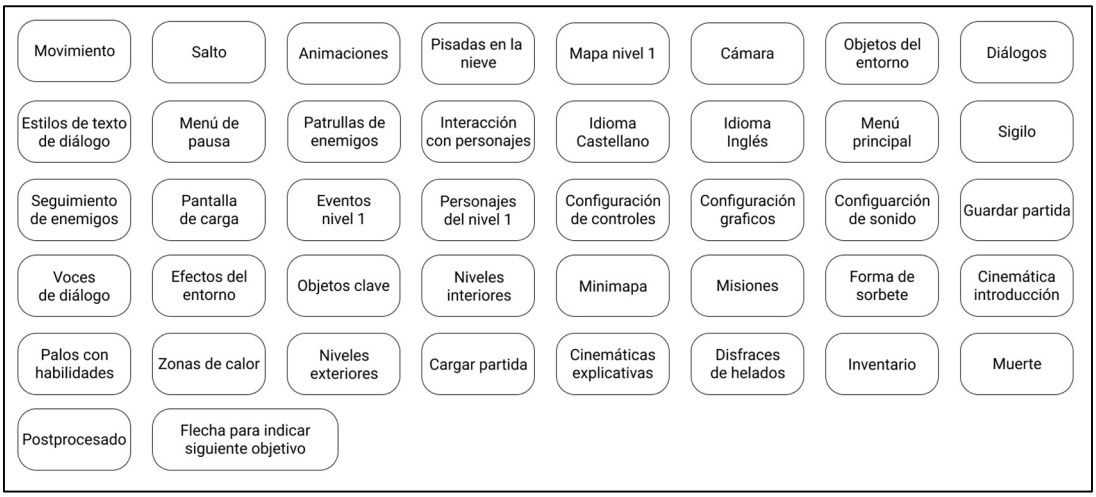

Figura 9. Mapa de características. Elaboración propia.

[38] Entendido como una parte del producto o servicio requerido. Característica de un producto cuya finalización aporta valor al usuario.
[39] Conjunto de tareas que se pretende realizar durante el desarrollo de un proyecto.
[40] Mapa de pequeño tamaño empleado en videojuegos para situar al jugador en un espacio.

3.2. Tecnología utilizada

Durante el desarrollo se hizo uso de diversas herramientas y tecnologías para llevar a cabo la creación del videojuego y, respecto a la especialidad de esta memoria, las funcionalidades relacionadas con el diálogo. En la Figura 10, se pueden comprobar a simple vista los logotipos de las tecnologías utilizadas y, a continuación, se ofrece una breve descripción de cada una de ellas y su utilidad para el desarrollo del proyecto.

Figura 10. Logotipos de las tecnologías utilizadas. De izquierda a derecha y de arriba a abajo: Unity, Visual Studio Community (2019), Git, GitHub, Trello, Yarn Spinner y RestSharp.

Unity

Unity es un motor de videojuegos 2D y 3D disponible para Windows, Linux y macOS, aunque es capaz de compilar proyectos para más de 25 plataformas, entre ellas Windows, Linux, macOS y las consolas de videojuegos de actual generación[41] a través de una misma base de código. Cabe destacar que el desarrollo para estas consolas está regido por el programa de desarrollador de cada compañía, la cual distribuye un módulo de soporte de Unity para sus consolas[42] y controla su uso. Unity es una de las soluciones más populares para la creación de videojuegos gracias a su versión *Personal*, la cual se encuentra de forma gratuita para proyectos que no superen los 100.000 $ en ingresos[43], de modo que se posiciona como una herramienta idónea para el desarrollo independiente y amateur. Unity presenta una interfaz de desarrollo moderna donde se puede añadir funcionalidad a través de escenas, permitiendo extender su funcionamiento e interacciones a través de archivos de código (también conocidos como *scripts*) de C# pudiendo también ver los cambios realizados en tiempo real. Además, cuenta con una tienda de recursos, llamada Asset Store[44], donde se pueden encontrar herramientas externas relacionadas, principalmente, con audio, gráficos, modelados, cámara y sistemas de juego. Para este proyecto en cuestión se utilizó la versión 2019.3.0.

Visual Studio Community

Visual Studio Community es un entorno de desarrollo integrado desarrollado por Microsoft y disponible para Windows y macOS. Se eligió este entorno de desarrollo debido a las extensiones y facilidades para el desarrollo en Unity disponibles desde su instalador nativo, los cuales incluyen opciones de depuración para proyectos de Unity. Además, las herramientas disponibles para el

[41] Octava generación de consolas, donde se incluyen principalmente Microsoft Xbox One, Sony PlayStation 4 y Nintendo Switch. Más información en <https://es.wikipedia.org/w/index.php?title=Videoconsolas_de_octava_generaci%C3%B3n&oldid=127962989>
[42] Más información en <https://unity3d.com/platform-installation>
[43] Extraído de <https://unity3d.com/legal/terms-of-service/software>
[44] Disponible en <https://assetstore.unity.com/>

desarrollo de proyectos en C# y .NET también se pueden reaprovechar, donde se incluye el análisis en tiempo real de código con recomendaciones funcionales y autocompletado de código basado en el contexto actual, así como la integración de las librerías de Unity para facilitar su acceso. Para este desarrollo se utilizó su versión de 2019.

Git

Git es una herramienta de control de versiones descentralizada disponible para Windows, macOS y Linux. A través de esta herramienta se posibilitó la colaboración conjunta en el proyecto de los integrantes del equipo. Dado que el proyecto en cuestión también consta de archivos binarios de gran tamaño (principalmente para modelados y texturas) se recurrió a uso de Git Large File Storage[45] (Git LFS), una extensión para Git que gestiona estos tipos de archivo de forma más eficiente para mejorar su acceso y ahorrar espacio de almacenamiento en el repositorio.

GitHub

GitHub es un servicio web que permite el guardado de proyectos bajo el control de versiones de Git, de modo que facilita la creación y mantenimiento de un repositorio remoto, simplificando el acceso al proyecto a sus colaboradores. Al tratarse de un servicio web, puede ser usado para almacenar y compartir el estado del proyecto con las partes interesadas, sin embargo, al estar basado en actualizaciones del código es preferible una herramienta externa más centrada en las características de alto nivel, como Jira[46] o Trello.

Trello

Trello es una aplicación web orientada a la gestión de proyectos. A través de esta herramienta es posible especificar las características deseadas para el proyecto en forma de tarjetas y planificar con facilidad las siguientes características que se van a desarrollar añadiéndolas a listas, así como asignarse características para dar a conocer en qué está trabajando cada componente del equipo. Esta estructura se puede adaptar en caso de hacer uso de una metodología ágil, como es nuestro caso, de modo que cada tarjeta representa una historia de usuario y se crean listas para simular un tablero Kanban para proyectos ágiles: una lista *backlog*, para las características por desarrollar; una lista en espera, para las características especificadas que esperan ser desarrolladas; una lista en desarrollo, para las características elegidas para el MVP actual; una lista en revisión, para las características en fase de pruebas y revisión; y finalmente una lista terminada, para las características que ya están dentro del producto. Esta herramienta permite compartir el estado del proyecto desde una perspectiva más abstracta y basada en características útiles para el usuario final.

Yarn Spinner

Yarn Spinner [14] es una herramienta que existe como extensión para Unity. Esta herramienta ofrece mecanismos para la proporción de líneas de texto de forma individual, tal que se puedan presentar durante el juego simulando una conversación. Las líneas de texto son almacenadas en un archivo de texto independiente siguiendo un formato específico. Durante el capítulo 4.1.1 se indagará en el porqué de su elección y cómo se integró en el proyecto. Cabe destacar que también se utilizó Yarn Spinner Editor, un aplicación web que facilita el control y creación de los archivos para almacenar el guion.

[45] Más información en <https://git-lfs.github.com/>
[46] Aplicación web diseñada para la administración de proyectos y seguimiento de su estado. Más información en <https://es.wikipedia.org/w/index.php?title=Jira&oldid=124628999>

RestSharp

RestSharp es una librería para .NET que permite realizar llamadas web y, sobre todo, es útil como cliente para APIs REST. Además, permite la serialización y mapeo a objetos de los datos XML y JSON recibidos para un manejo más intuitivo de los datos recibidos. Para su uso en Unity se recurrió a un *fork* [47] pensado para su integración en el motor de videojuegos, llamado RestSharp.Unity[48]. Esta herramienta se utilizó para realizar la conexión con Instagram, descrita en el capítulo 4.9, necesaria para la creación de una escena orientada a la mercadotecnia del juego.

[47] Traducido como bifurcación, en proyectos de código libre es el hecho de crear un proyecto en base a otro para añadir o modificar su funcionalidad, de modo que se pueden discernir dos ramas de desarrollo. Más información en <https://es.wikipedia.org/w/index.php?title=Bifurcaci%C3%B3n_(desarrollo_de_software)&oldid=128243521>

[48] Disponible en <https://github.com/eamonwoortman/RestSharp.Unity>

3.3. Primer *Minimum Viable Product*

Las características seleccionadas a desarrollar para el primer MVP fueron las mostradas en la Figura 11. Como se puede comprobar, aquí se incluyeron las mecánicas básicas, como andar, saltar y agacharse, así como la posibilidad de diálogo con los distintos personajes que aparecían en el entorno y el comportamiento de los enemigos. Por otro lado, también se desarrolló una primera versión del menú principal y de pausa, para que los usuarios pudieran hacer ajustes y acomodar los controles a sus necesidades, o pausar el juego. Para evaluar la viabilidad de las características novedosas se incluyeron la creación de los personajes para el primer nivel, el estilo de texto y la creación de objetos del entorno, con el objetivo de empezar a establecer el carisma del juego y comprobar el atractivo de este desde su inicio. También se incluyeron factores de riesgo como el efecto de pisadas en la nieve, el cual también contribuiría a la estética del juego, pero, por experiencia anterior de los componentes del equipo, podría suponer un problema en su implementación.

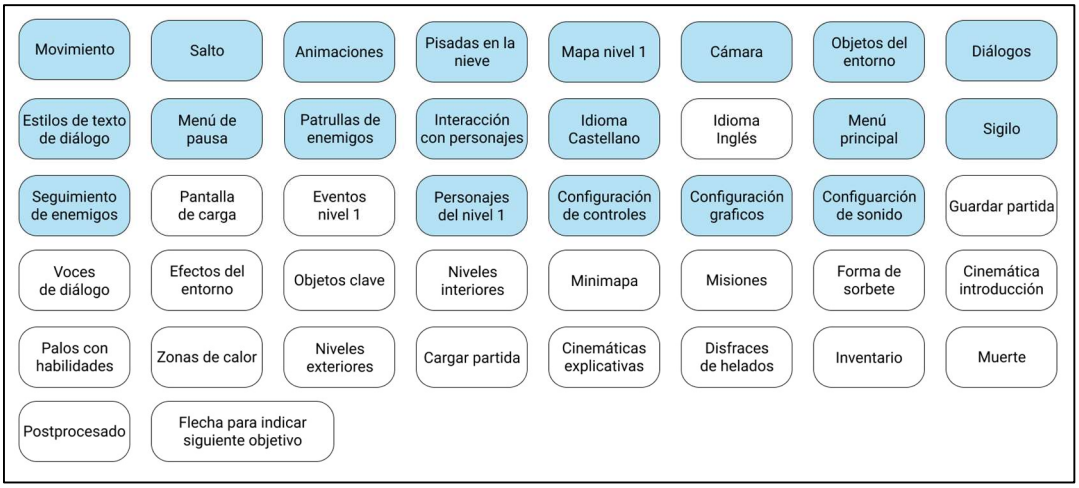

Figura 11. Mapa de características seleccionadas para el primer MVP (marcadas en azul). Elaboración propia.

3.3.1. Desarrollo del MVP

El nivel pensado para este periodo de desarrollo se iba a situar en un entorno exterior, tomando como punto de referencia la nevera, ya que es donde el inicio de la historia iba a tener lugar. En la Figura 12 se puede comprobar el aspecto que tendría el nivel, introduciendo la nevera, la ciudad exterior, así como un vistazo a la ciudad abandonada, el que se consideraría el mapa para el segundo nivel. Este mapa también se reutilizaría para crear un corto tutorial para la feria.

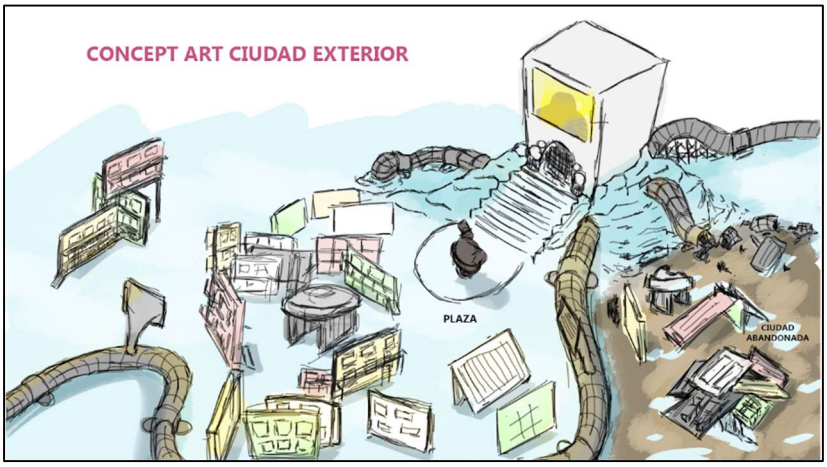

Figura 12. Arte conceptual para el nivel exterior. Elaboración propia.

En cuanto a la jugabilidad, se discutió cómo debería ser el movimiento del personaje y la libertad que tendría para moverse por el entorno, qué animaciones serían necesarias para acompañarlo y cómo sería la cámara del juego. Nos decantamos por una cámara libre en tercera persona que siguiera al jugador, y se resolvió el problema de que la cámara atravesase objetos: acercándola al jugador según la distancia del choque. Por otro lado, pese a que el usuario tiene total libertad de movimiento, la zona se delimitó con tuberías y muros que dan forma al entorno, y, por otra parte, tal como se puede ver en la Figura 13, ciertos personajes no jugables (PNJ) marcados con STOP, permanecerían bloqueando caminos de forma indefinida.

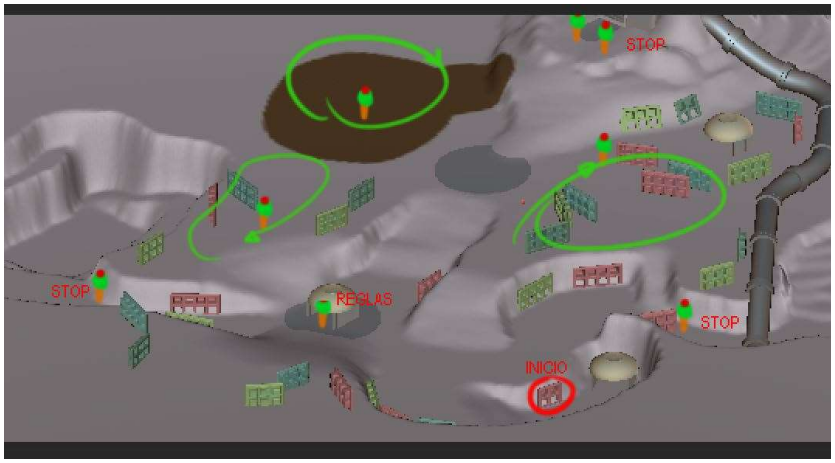

Figura 13. Mockup del mapa para el primer MVP con la distribución de enemigos, patrullas y puntos clave para el jugador. Elaboración propia.

Como se ha descrito, el efecto de la nieve era clave para transmitir una sensación de frio y, en general, marcar la estética del entorno. Para ello, se recurrió al uso de un sombreador (*shader*)[49] para aplicar el efecto de una capa de nieve que se acumula en el suelo[50], así como pisadas en el suelo disminuyendo esta capa y creando un rastro por donde pasan los personajes. Este efecto era eficaz en terrenos pequeños, sin embargo, para el mapa de este nivel el efecto era pobre e imperceptible, de modo que la demo compilada no tuvo este efecto y esta característica se mantuvo para el trabajo del siguiente MVP con el objetivo de finalizar su implementación.

En el apartado de los diálogos, la idea original para la presentación del texto se puede comprobar en la Figura 14, donde el texto aparece como bocadillos flotantes sobre el personaje. Además, también se puede ver una idea inicial del indicador de diálogos, colocando una imagen sobre ellos para señalar al jugador que un personaje tiene una conversación disponible. Estas ideas se refinaron en un segundo mockup (Figura 15), donde el texto se presentaría en una clásica caja de diálogos, a modo de centralizar la atención del jugador.

[49] Código ejecutado en la unidad gráfica de una computadora que modifica la imagen procesada según se especifique en este. Más información en <https://es.wikipedia.org/w/index.php?title=Sombreador&oldid=122810336>
[50] El tutorial seguido por el equipo para introducirse en este concepto fue el disponible en <https://www.youtube.com/playlist?list=PL3POsQzaCw53KM74XVRXv2xyesLjngfbG>

Figura 14. Primer mockup de indicadores de diálogo y presentación de diálogo con bocadillos. Elaboración propia.

Figura 15. Segundo mockup de indicadores de diálogo y presentación de diálogo en una caja de texto. Elaboración propia.

Por otra parte, para el menú principal (Figura 16), al ser una primera versión de prueba a enseñar al público, se optó por un estilo básico, situando de forma simple las funcionalidades que tenía: empezar la partida, acceder a las opciones (como ajustes de sonido, video, controles y juego) y salir de la partida. De la misma forma, el menú de pausa (Figura 17) solamente contenía las opciones de reanudar partida, guardar y cargar partida (aunque estas no estaban desarrolladas), la opción de repetir el nivel y la de volver al menú principal.

Figura 16. Menú principal durante el primer MVP. Elaboración propia.

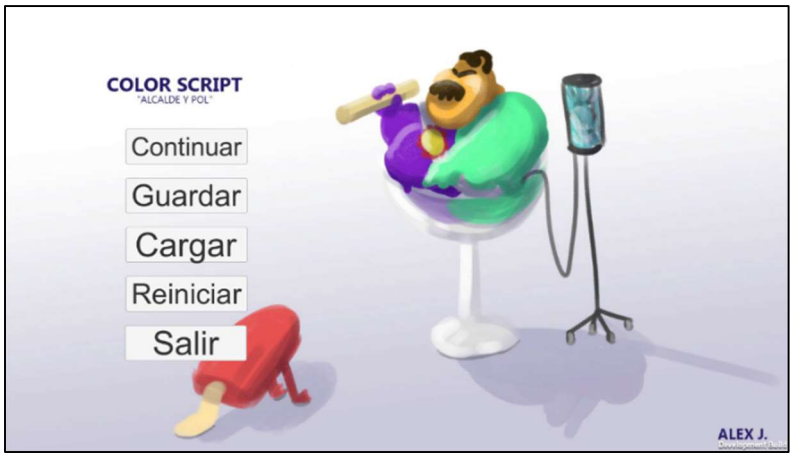

Figura 17. Menú de pausa del juego durante el primer MVP. Elaboración propia.

3.3.2. Experimento

Durante la «Feria de proyectos de estudiantes» se compartió una encuesta realizada con Google Forms[51] con ocho preguntas y tres sugerencias a los que probaron el juego. Estas preguntas estaban dirigidas a resolver dudas sobre los aspectos que creíamos eran los más atractivos (estética, temática, historia) y a poner énfasis en aspectos que podían haber resultado problemáticos. Las sugerencias planteadas permitían a los usuarios escribir de forma libre aspectos relacionados con aquello que más y menos había gustado. A continuación, se pueden ver los resultados de la encuesta, la cual fue contestada por ocho personas.

Pregunta 1.- ¿Crees que te ha quedado clara la idea del juego?

Dado que la temática del juego iba vinculada con un mensaje relacionado con el cambio climático, se elaboró esta pregunta para comprobar si los jugadores la habían percibido, tal que se debiera realizar algún cambio en el diseño del nivel y cómo la idea era representada. Como se puede observar en la Figura 18, el 62,5 % comprendió la idea, sin embargo, quedaba claro que aun podía mejorar dado que un 37,5 % de los encuestados habían seleccionado la opción «no del todo».

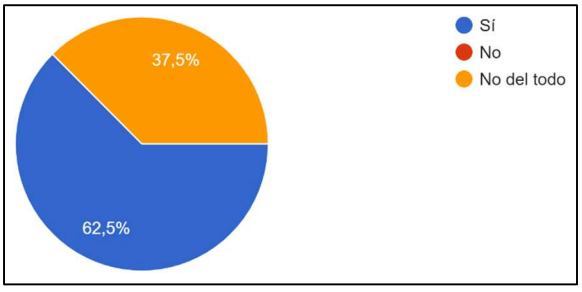

Figura 18. Encuesta primer MVP, resultados pregunta 1. Elaboración propia.

Pregunta 2.- ¿Te ha resultado fácil comenzar a jugar o cambiar las opciones de juego?

La segunda pregunta, relacionada con el menú principal y el de opciones, fue elaborada para comprobar si los usuarios podían navegar correctamente por estos, comprobando si era intuitivo su uso. Como se puede observar en la Figura 19, en este caso se obtienen resultados muy positivos, pues al 87,5 % de los usuarios les resultó fácil, hecho que indicaba que estos cumplían su función.

[51] Disponible en <https://www.google.com/intl/es/forms/about/>

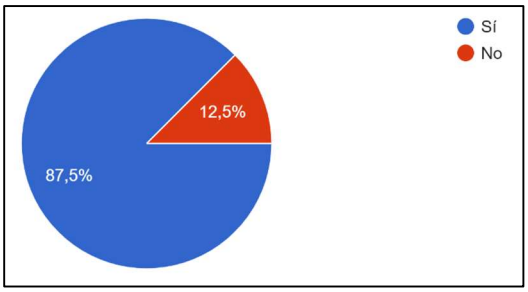

Figura 19. Encuesta primer MVP, resultados pregunta 2. Elaboración propia.

Pregunta 3.- ¿Te has quedado con ganas de jugar al resto de niveles del juego?

La tercera pregunta estaba relacionada con el interés general del producto, comprobando si los encuestados quisieran jugar al resto de niveles del juego. Así podríamos comprobar si la idea podía mantener su frescura y mantener al jugador interesado. Como se puede observar en la Figura 20, todos los encuestados quedaron con ganas de probar más niveles del juego, resultado que el equipo recibió como un éxito.

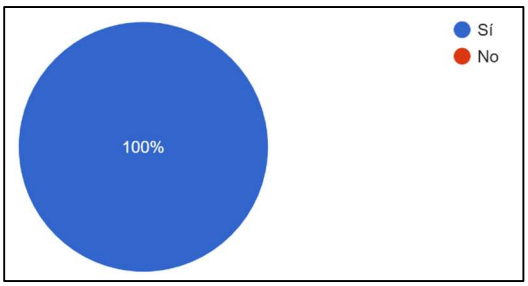

Figura 20. Encuesta primer MVP, resultados pregunta 3. Elaboración propia.

Pregunta 4.- ¿Cuántas horas estarías dispuesto a invertir en un posible juego final?

La cuarta pregunta estaba relacionada con la duración del juego, preguntando cuantas horas estarían dispuestas a jugar a la versión final del producto. Como se puede observar en la Figura 21, esta pregunta fue más diversa, ya que se ofrecían diez opciones. La mayoría de los encuestados preferirían jugar entre una y cuatro horas, mientras que una minoría eligieron una duración más larga (ocho y diez horas, respectivamente). Con esto podemos comprobar que el juego tendría éxito como una experiencia corta, aunque puede que la gente acostumbrada a jugar a estos juegos prefiriera una mayor duración.

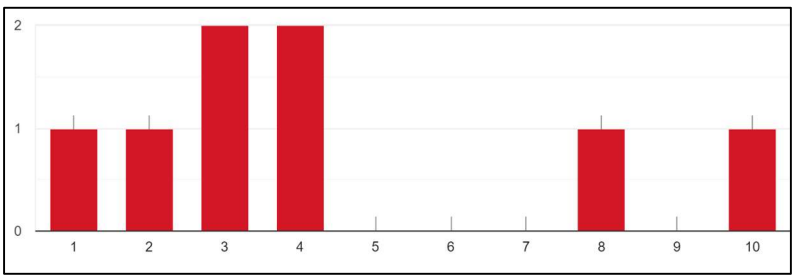

Figura 21. Encuesta primer MVP, resultados pregunta 4. Elaboración propia.

Pregunta 5.- ¿Crees que se puede hacer repetitivo/aburrido al poco tiempo?

Esta pregunta fue realizada para comprobar si el conjunto de las mecánicas elegidas para el juego junto con los diálogos, eran suficientemente atractivas para que el jugador siguiera jugando durante un intervalo prolongado de tiempo. Como se puede observar en la Figura 22, esta pregunta presenta diversidad en sus respuestas, la mayoría de los encuestados perdonarían la repetitividad en caso de ser un juego corto, de modo que, en caso de aumentar el número de zonas, se debería

tener en cuenta que no tengan una gran extensión que alarguen más de la cuenta la duración del juego.

Figura 22. Encuesta primer MVP, resultados pregunta 5. Elaboración propia.

Pregunta 6.- ¿Estarías dispuesto a terminar el juego solo por continuar con la historia (conspiración de la nevera, enfrentamiento con el alcalde, …)?

La sexta pregunta estaba relacionada con la historia. Dado que la demo disponible solo mostraba una pequeña porción de la historia en comparación con los planes previstos, se tuvo que explicar a los participantes los antecedentes del personaje y que tipo de historia era esperable en los siguientes niveles. Se preguntó, por tanto, si estarían dispuestos a terminar el juego para continuar la historia, con tal de comprobar su atractivo. Como se puede observar en la Figura 23, en este caso se obtuvieron resultados muy positivos, dado que el 87,5 % de encuestados indicaron que sí estarían dispuestos a seguir y terminar el relato, reforzando el encanto y originalidad de la historia.

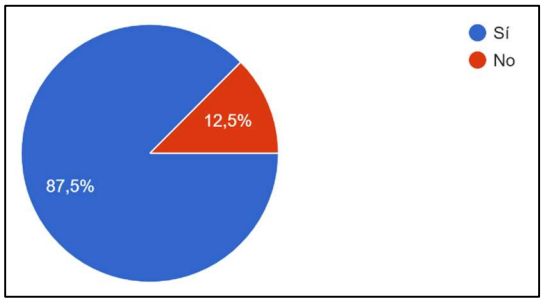

Figura 23. Encuesta primer MVP, resultados pregunta 6. Elaboración propia.

Pregunta 7.- ¿Recomendarías a otra gente jugar a este juego?

La séptima pregunta era sobre si recomendarían el juego a otra gente, poniendo a prueba el factor de la difusión del juego, el cual mejoraría la base de posibles interesados sin invertir recursos adicionales. Como se puede observar en la Figura 24, el 62,5 % de los encuestados recomendaría el juego, aunque el 25 % solo lo recomendaría a fans del género, de modo que, en general, se demuestra que el producto podría gustar a bastante gente y ser fácilmente comunicado a otros interesados, aunque algunos consideran que solo interesaría a gente acostumbrada a juegos similares.

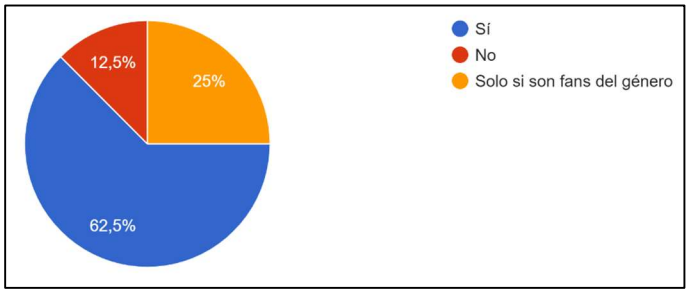

Figura 24. Encuesta primer MVP, resultados pregunta 7. Elaboración propia.

Pregunta 8.- ¿Has jugado a otros juegos como este?
Uno de nuestros puntos fuertes es la originalidad del producto y su estética, sin embargo, las mecánicas son conocidas por jugadores del género de puzle y aventuras, de modo que con esta pregunta quisimos conocer como de similar podían ver los usuarios nuestro producto con otros juegos del mismo género para así atraerlos a nuestro producto. Como se puede observar en la Figura 25, el 75 % de los encuestados han jugado a juegos similares o relacionados, esto nos muestra que podríamos atraer al público existente e interesado en este género.

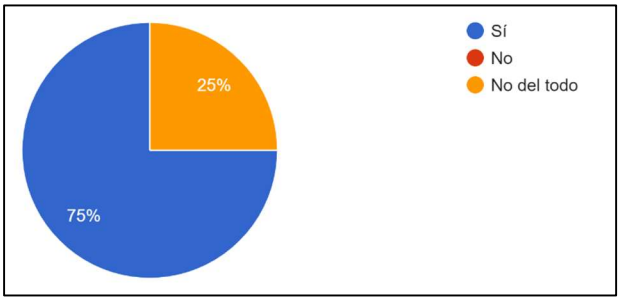

Figura 25. Encuesta primer MVP, resultados pregunta 8. Elaboración propia.

Sugerencia 1.- ¿Qué aspectos te han gustado del juego?
En la primera de las sugerencias, queríamos conocer qué aspectos eran los que más habían gustado tras haber probado la primera versión del juego. Así pues, se obtuvo como resultado que los aspectos más destacados fueron la originalidad de la historia y la estética del juego («La ambientación e historia» o «El diseño y los controles»).

Sugerencia 2.- ¿Qué aspectos no te han gustado y cómo crees que se pueden mejorar?
La segunda sugerencia, en contraposición, hacía referencia a los aspectos negativos. El mapa obtuvo críticas por su extensión y la falta de contenido en él («El mapa es demasiado grande o confuso» o «Falta contenido, mapa muy grande»). La jugabilidad también estuvo presente con comentarios como «Los controles pueden dar fallos» o «lento de andar». De modo que se tuvo en cuenta para la siguiente iteración de desarrollo una revisión a la jugabilidad y una recreación del mapa para que fuera más reducido y sus componentes más fácilmente accesibles.

Sugerencia 3.- Sugerencias y otras consideraciones.
Para cerrar la sección de sugerencias, se pidió a los usuarios que dieran su libre opinión acerca del juego sin ninguna pregunta en específico, para ver así qué otros aspectos no incluidos en el juego echaban a faltar o que les sobraba. Como respuesta, se obtuvieron distintas sugerencias, principalmente destacando que la apuesta era interesante, pero que hacía falta más contenido y que aspectos de la interfaz relacionados con el texto en ocasiones no se ajustaba al cuadro de diálogos («El *layout* de los diálogos es confuso»).

3.4. Segundo *Minimum Viable Product*

Las características seleccionadas a desarrollar para este segundo MVP fueron las mostradas en la Figura 26. Por un lado, con el objetivo de mejorar los aspectos problemáticos durante el primer experimento se planificó modificar aspectos como el movimiento, el salto y el sigilo, que formarían parte de una reestructuración del movimiento. Similarmente, para atajar los problemas relacionados con la falta de dirección y magnitud del mapa, se planeó el rediseño del mapa, así como adaptar la cámara al nuevo entorno y añadir nuevos personajes al nivel. Se incluyeron también retoques al efecto de pisadas en la nieve, el cual no pudo ser finalizado y debería ser revisado para este periodo. Con el objetivo de reforzar la estética del juego, un aspecto apreciado en el anterior MVP, se consideró la inclusión de más efectos de entorno, como las partículas y el postprocesado, y actualizar el menú principal y el de pausa.

Por otro lado, para este periodo se planeó la inclusión de voces para los diálogos, la traducción del juego al inglés, la introducción de los diversos eventos en el mapa, la creación de la zona de calor para delimitar la zona jugable. Se consideró también una nueva tarea de mejoras de rendimiento, que se incluyó al desarrollo de este MVP, para resolver los problemas relacionados con el uso de memoria y diseño interno observados durante el anterior periodo. En cuestión de mecánicas se añadiría un inventario al jugador, así como los objetos clave, la forma de sorbete del jugador y la posibilidad de morir para reiniciar el nivel. Para mejorar la comprensión también se seleccionaron la creación de una pantalla de carga y la inclusión de diferentes cinemáticas, tanto explicativas como relacionadas a eventos.

Adicionalmente, y para futuros desarrollos, las tareas de cargado y guardado se remplazaron por una tarea de sistema de puntos de control[52] para reflejar mejor el estilo de juego, liberando al jugador de las tareas de control de guardados.

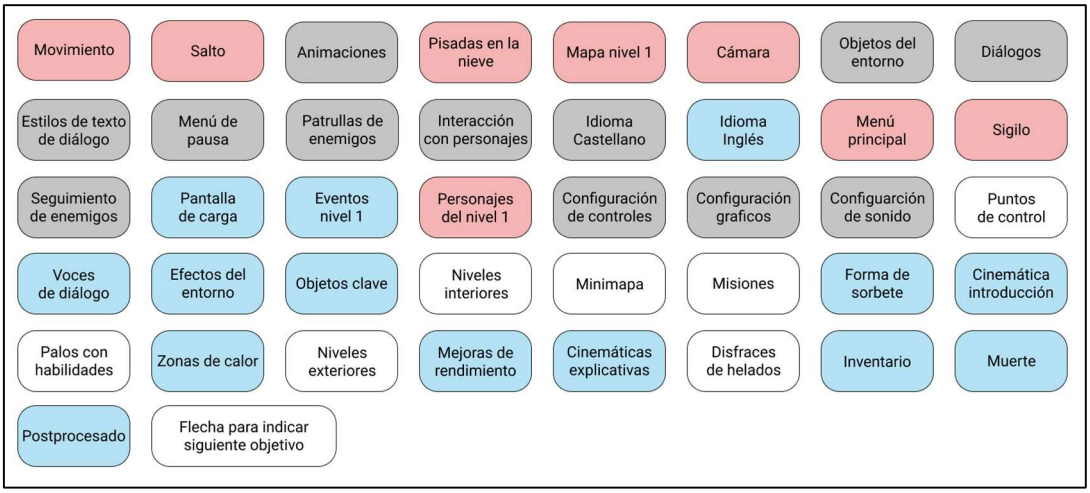

Figura 26. Mapa de características seleccionadas para el segundo MVP (en azul las nuevas características; en rojo las características a modificar o arreglar en base a los comentarios recibidos; en gris las características finalizadas en el MVP anterior). Elaboración propia.

3.4.1. Desarrollo del MVP

Tras la feria de diciembre y los exámenes del mes de enero, no fue hasta casi mitad de febrero que se empezó a planear la remodelación del nivel que presentamos en la feria, así como el movimiento, que es el sistema que había dado más problemas. Junto a esto, se pensó como se

[52] Zonas en las que se guardará automáticamente la situación del jugador la primera vez que esta es alcanzada, sirviendo como punto de aparición en caso de cerrar el juego o perder la partida.

podrían adaptar los sistemas que ya se habían implementado en el juego y como se podrían incluir nuevos, añadiendo el inventario que, junto a los diálogos, serían el núcleo de la jugabilidad.

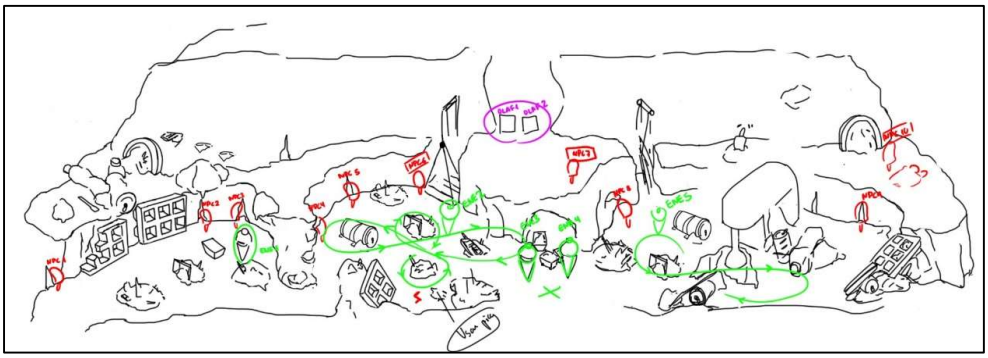

Figura 27. Mockup de la versión del mapa para el segundo MVP. Elaboración propia.

Como se pude ver en la Figura 27, se planeó un mapa con una extensión más reducida, teniendo lugar en una mina donde sus componentes serían más fácilmente accesibles. Tal y como se puede apreciar, el lado contrario del mapa quedaría vacío, visible en la zona marrón vacía de la Figura 28. Para el control de la cámara en este nuevo entorno se decidió usar Cinemachine[53], una paquete opcional incluido en Unity que extiende las funciones para la cámara. Con esta herramienta se pudo crear un sistema basado en raíles por los que interpolaría la cámara, creando un movimiento más suave.

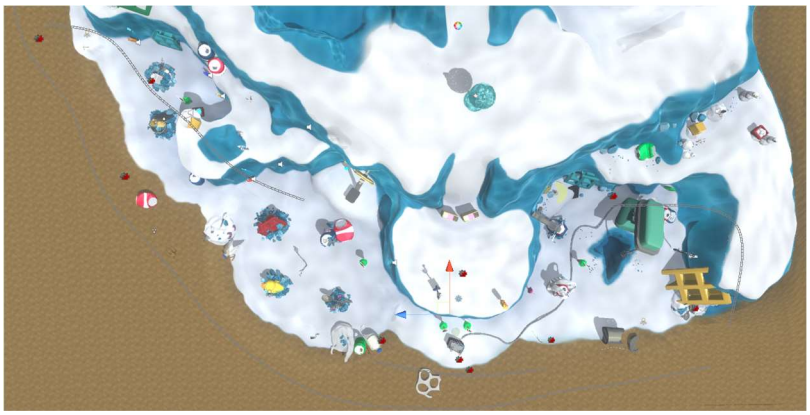

Figura 28. Zonas del mapa del segundo MVP vistas desde arriba, incluyendo el páramo (zona marrón). Elaboración propia.

En cuanto al movimiento, dado que la primera versión estaba desarrollada a partir de un componente que no tenía en cuenta las físicas y no se tuvo en mente la habilidad de cambio de forma del jugador, nos vimos obligados a cambiarlo por completo de forma interna, separándolo además en diversos *scripts* y empleando un componente diferente que si las empleara. Además, se modificaron animaciones y se añadieron sonidos a algunas de estas.

Sobre la inteligencia artificial, para este segundo MVP se mejoró la respuesta de las patrullas, resolviendo algunos fallos relacionados con las animaciones; se preparó un sistema de detección completo similar al de juegos de sigilo tradicionales con detección y distintos campos de visión. Junto a esto, también se desarrolló un editor propio para su manejo y configuración.

[53] Disponible en <https://unity.com/es/unity/features/editor/art-and-design/cinemachine>

Uno de los aspectos que iba a mejorar la interacción con el nivel y favorecer la exploración era la adición de ítems y un inventario del personaje. Tal como se puede ver en la Figura 29, este inventario sería presentado de una forma sencilla, con cada ítem representado por un icono. Además, a través de este menú se podrían equipar los ítems que el personaje pudiera llevar (para este nivel el pico y la cuchara). Solo un ítem podría estar equipado a la vez, y en el menú del inventario sería marcado con un color azul.

Figura 29. Mockup de la interfaz para el inventario. Elaboración propia.

Con relación a las mejoras de rendimiento, por una parte, se disminuyó en la medida de lo posible el número de polígonos de los elementos que conformaban el mapa, eliminando además las colisiones no necesarias o sustituyendo las que se podían por unas más simples. Por otro lado, se reestructuró el código para que la comunicación entre componentes fuera más eficiente y también hiciera más fácil la creación de nuevos niveles desde cero. Para acomodarse a este nuevo diseño interno, se migraron las funcionalidades de los diálogos, y se retocaron sus componentes de interfaz para tener un estilo consistente con el resto del juego. El soporte para otros idiomas, concretamente el inglés, se llevó a cabo a través de una librería desarrollada por Unity (Unity Localization[54]) que se encontraba en estado beta. Esta librería permitía actualizar el texto mostrado en pantalla en base al idioma seleccionado. Para localización del diálogo, sin embargo, se usaron las opciones internas de Yarn Spinner, de modo que se focalizara la edición del guion. Para acomodarse al estilo del videojuego, las voces de los personajes fueron sintéticas; durante el capítulo 4.4 se indagará en las alternativas y su implementación.

En cuanto al menú principal, este fue sustituido completamente por uno nuevo (Figura 30) en el que se mostraron las opciones principales y los ajustes, ambos integrados en una nevera y con efectos de distorsión de imagen. En cuanto al efecto de las pisadas en la nieve, se aplicaron las mejoras para soportar terrenos grandes, de modo que el efecto fuese visible en el mapa actual. Cabe destacar, sin embargo, que este efecto no está presente en la demo compilada de este MVP debido a problemas técnicos con la exportación de Unity.

[54] Disponible en beta como paquete de Unity <https://docs.unity3d.com/Packages/com.unity.localization@0.6/manual/index.html>

Figura 30. Menú principal de Frozen Out tras su rediseño en el segundo MVP. Elaboración propia.

3.4.2. Experimento

Al finalizar este MVP se realizó una prueba digital compartiendo un ejecutable del juego a distintos jugadores. Decidimos también incluir un enlace a una guía para que supieran qué hacer si se encontraban perdidos. Para evaluar su experiencia, se les adjuntó nuevamente una encuesta de Google Forms de diez preguntas y tres sugerencias. Están preguntas estaban dirigidas a resolver dudas sobre los aspectos que habían sido revisados, como la reducción del mapa e historia más concisa, así como el movimiento y la cámara. También se planeaba evaluar los aspectos que habían tenido éxito (humor, estética) para saber si permanecían interesantes. En cuanto a las sugerencias, similarmente a la encuesta anterior se permitió compartir libremente qué aspectos habían sido de su agrado y cuáles no. A continuación, se describen los resultados de la encuesta, la cual fue contestada por trece personas.

Pregunta 1.- ¿Has podido saber cuál era el objetivo por cumplir en cada momento?

Debido al rediseño del mapa, se preguntó a los encuestados si, gracias a los diálogos con los diferentes PNJs y las diferentes cinemáticas, les resultaba intuitivo conocer cuál debía ser su siguiente objetivo conforme los iban completando. Como se puede ver en la Figura 31, un 53,8 % de los encuestados tuvieron dificultades para conocer el objetivo a seguir. Esto significa que deberemos hacer énfasis en este aspecto y reforzarlo para mejorar la experiencia.

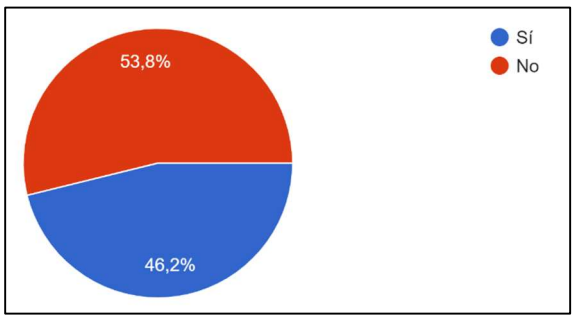

Figura 31. Encuesta segundo MVP, resultados pregunta 1. Elaboración propia.

Pregunta 2.- ¿Crees que ha sabido transmitir correctamente su idea y critica medioambiental?

La segunda pregunta se hizo para comprobar si la crítica medioambiental había quedado clara durante el juego, la cual había sido insinuada en conversaciones de personajes y el entorno. Como podemos observar en la Figura 32, más de la mitad de los encuestados considera que no se transmite del todo su idea y critica ambiental, de modo que el equipo decidió mejorar la representación de estos aspectos en más mecánicas, como cinemáticas o efectos.

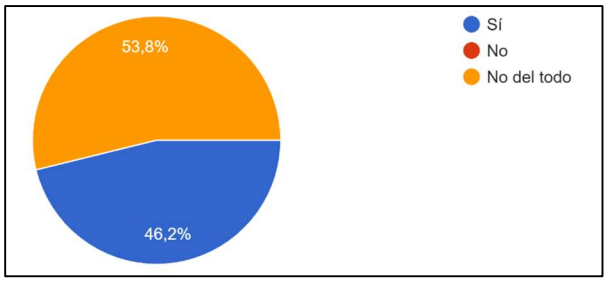

Figura 32. Encuesta segundo MVP, resultados pregunta 2. Elaboración propia.

Pregunta 3.- ¿Cómo de satisfactoria te ha resultado la partida?

Debido al cambio en el mapa y los nuevos objetivos incluidos, se pidió a los encuestados que puntuarán del 1 al 10 su satisfacción tras haber completado todos los objetivos. Como se puede ver en la Figura 33, había diferencia de opiniones, obteniendo una media de 6. De modo que, pese a que sabemos que no es una nota muy destacable, por comentarios de las sugerencias, conocemos que el principal problema era debido a la ausencia de puntos de control durante la demo.

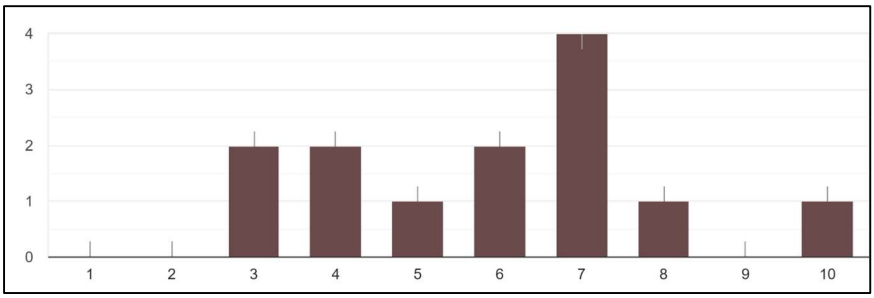

Figura 33. Encuesta segundo MVP, resultados pregunta 3. Elaboración propia.

Pregunta 4.- ¿Estarías dispuesto a terminar el juego solo por continuar la historia?

La cuarta pregunta tenía el objetivo de conocer si se mantenía el atractivo de la historia y sus personajes. Como podemos observar en la Figura 34, el 86,4 % de la gente estaría dispuesta a terminar el juego por continuar la historia, de modo que se reafirma el interés del público en el entorno y narrativa presentados, demostrando que el esfuerzo en estos aspectos no debe ser mermado.

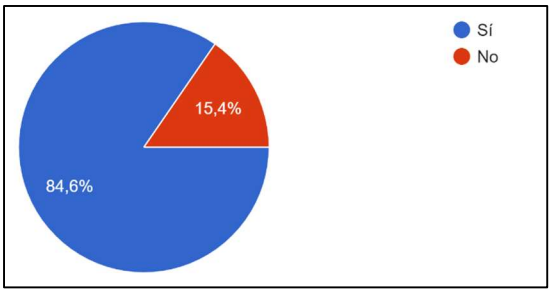

Figura 34. Encuesta segundo MVP, resultados pregunta 4. Elaboración propia.

Pregunta 5.- Si probaste el juego en la feria de diciembre, ¿crees que se han resuelto los problemas que encontraste?

Con el objetivo de intentar que alguna de las personas que participaron en el primer experimento probaran la nueva versión del producto, realizamos esta pregunta para comprobar si desde su punto de vista, los problemas que percibieron se habían solucionado, obteniendo una respuesta afirmativa de todos los encuestados que participaron en el experimento anterior, tal y como se aprecia en la Figura 35.

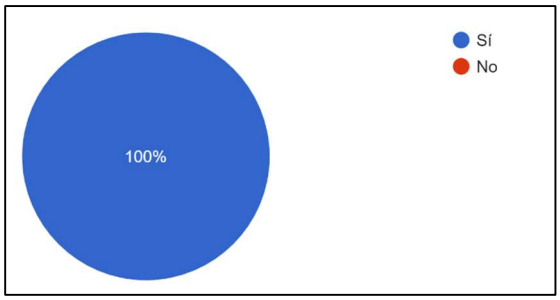

Figura 35. Encuesta segundo MVP, resultados pregunta 5. Elaboración propia.

Pregunta 6.- ¿Te ha parecido claro y funcional el diseño de los menús?

La sexta pregunta se realizó para asegurarse que los menús eran claros y sencillos de utilizar, ya que se habían realizado cambios a los mismos y se habían agregado menús para el control del inventario. Como se puede observar en la Figura 36, en este caso las respuestas fueron positivas, con un 92,3 % de los encuestados que han considerado el diseño de los menús claro y funcional. Solo un 7,7 % ha indicado que tuvo una mala experiencia.

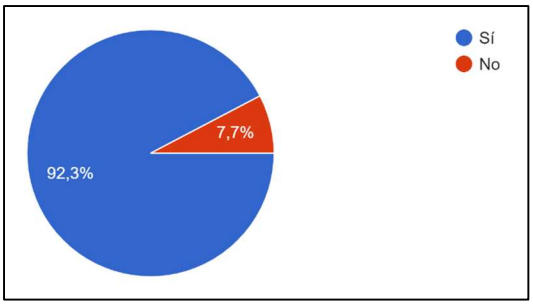

Figura 36. Encuesta segundo MVP, resultados pregunta 6. Elaboración propia.

Pregunta 7.- ¿Has tenido algún problema al interactuar con el inventario?

Debido a la inclusión del inventario en el juego, decidimos preguntar más específicamente acerca de su interfaz y uso, con el objetivo de comprobar si el diseño y funcionamiento elegido era del agrado de los usuarios. Como podemos observar en la Figura 37, el 69,2 % de los encuestados no ha tenido problemas para interactuar con el inventario, con un 30,8 % teniendo dificultades, indicando que este sistema es funcional, pero debería ser ligeramente revisado.

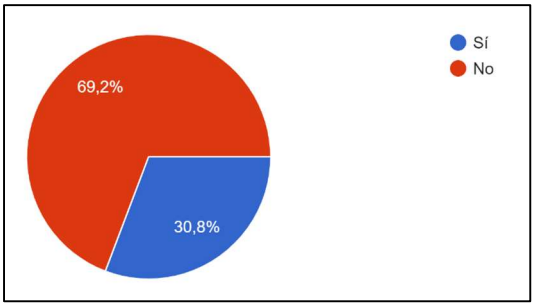

Figura 37. Encuesta segundo MVP, resultados pregunta 7. Elaboración propia.

Pregunta 8.- ¿Has encontrado el texto fácil de leer?

Como pregunta acerca de la accesibilidad a la hora de leer los diálogos del juego, preguntamos acerca de su facilidad de lectura, ya que optamos por un color para la caja de diálogos y la letra de un tono similar y con un bordeado negro. Según las respuestas de la encuesta (Figura 38), un

7,7 % de los encuestados tuvo problemas ocasionales al momento de leer alguno de los textos, frente a la amplia mayoría del 92,3 % que no encontró problemas para ello.

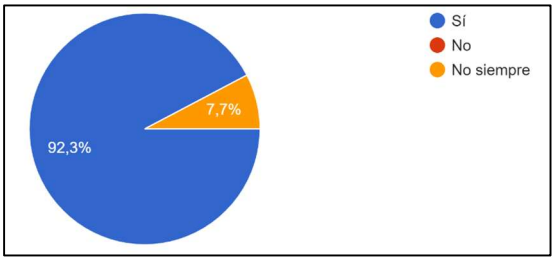

Figura 38. Encuesta segundo MVP, resultados pregunta 8. Elaboración propia.

Pregunta 9.- ¿Si se abriera una campaña de Kickstarter (micromecenazgo), estarías dispuesto a contribuir para terminar el desarrollo?

La novena pregunta hacía referencia al futuro del proyecto y la disposición de los usuarios a ayudar en la finalización del proyecto, en concreto a su posible participación con la hipotética campaña de microfinanciación en Kickstarter. Como se puede observar en la Figura 39, un 69,2 % de los encuestados estrían dispuestos a participar en esta campaña de Kickstarter para contribuir al desarrollo, poniendo de manifiesto una vez más que la idea tiene potencial.

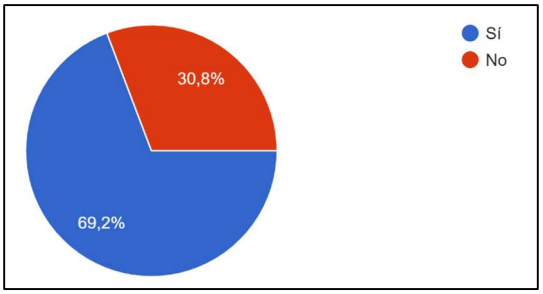

Figura 39. Encuesta segundo MVP, resultados pregunta 9. Elaboración propia.

Pregunta 10.- En caso afirmativo, ¿cuánto estarías dispuesto a contribuir (en €)?

Como continuación de la pregunta anterior, se preguntó sobre la cantidad de dinero con la que estarían dispuestos a contribuir en caso de abrirse la campaña. Cabe destacar que esta pregunta se dejó como obligatoria de contestar por error, hecho que puede justificar los valores atípicos como 0,01. Pese a esto, debido al reducido número de encuestados, los resultados (Figura 40) son bastante prometedores, ya que de los trece encuestados, ocho afirmaron estar dispuestos a contribuir con 5 € o más, de modo que, a una escala considerablemente mayor, se podría obtener beneficio.

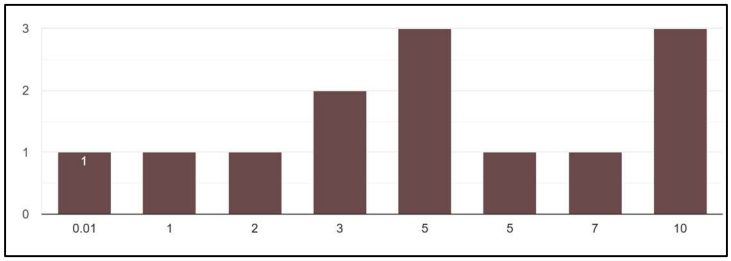

Figura 40. Encuesta segundo MVP, resultados pregunta 10. Elaboración propia.

Sugerencia 1.- ¿Cuáles han sido los aspectos que más te han gustado?

La primera sugerencia estaba relacionada con los puntos positivos del proyecto, preguntando qué aspectos habían gustado más. En la mayoría de las respuestas se menciona la estética y diseño del

entorno, resaltando su originalidad («El diseño de los personajes y el mundo tienen mucha personalidad.») y detalles («Las animaciones de los helados y el movimiento» o «La adaptación de elementos del mundo real al mundo de los polos»). También se hace hincapié en su aspecto narrativo, destacando las diferentes conversaciones con los personajes («Los diálogos eran graciosos, y la historia interesante»).

Sugerencia 2.- ¿Cuáles han sido los aspectos que no te han gustado o podrían mejorarse?
Dado el mayor alcance conseguido con este experimento, era de esperar nuevos puntos de vista a la hora de probar el producto. Así pues, los aspectos que menos gustaron fueron tener que repetir todo el nivel tras morir, debido a la ausencia de puntos de control («Me gustaría haber tenido algún tipo de *checkpoint* o guardado de partida, aunque fuera manual»); el comportamiento de la cámara, que en ocasiones no era el correcto («La cámara es liosa») y debido a no poder controlarla generaba situaciones en las que dificultaba el juego («La cámara no ayudaba: esto era especialmente problemático en algunos saltos, ya que era difícil saber dónde ibas a caer»); y la interacción con el inventario, cuyos controles no quedaron claros a algunos usuarios («La interfaz de los menús es clara excepto el menú de objetos. A lo mejor indicar las teclas de cómo utilizarlo»). Con estas sugerencias, está claro que nos falta mucho por mejorar en general y mucho que pulir en cuanto al diseño y la implementación de mecánicas y sistemas.

Sugerencia 3.- Sugerencias y otras consideraciones.
La tercera sugerencia serviría para incluir conclusiones que no habían tenido cabida en los otros apartados. En este caso nos indicaron posibles mejoras para conocer el objetivo o la misión actual en todo momento, ya que muchos de ellos habían usado la guía para avanzar en el juego («Estaría bien que de alguna forma se indicara lo que debes de hacer» o «No se sabe que hay que hacer […] sin la guía»). También hay mejoras para la cámara, dónde se sugirió el uso de una «[cámara] estática un poco desde arriba» para aliviar los problemas de visibilidad. Finalmente, como consideración para los menús se recomienda ofrecer soporte para «navegar [los menús] usando el teclado».

3.5. Mercadotecnia y difusión

Dado que este trabajo se realizó dentro de un marco universitario, existieron actividades de control por parte del profesorado de las asignaturas para controlar y evaluar el desarrollo de cada grupo de trabajo. Estas evaluaciones se realizaban en un estilo de presentación, donde cada grupo de trabajo debía defender delante del resto de alumnos las adiciones que se habían realizado a su proyecto. De este modo, se podía aprovechar para mostrar el juego y sus atractivos al resto de compañeros, los cuales podrían difundir el proyecto dentro de la universidad en preparación para futuros eventos estudiantiles.

Como tarea final para la asignatura, se nos presentó la oportunidad de mostrar el proyecto realizado en la «Feria de proyectos de estudiantes» de la escuela (Figura 41), en la que pudimos tener un contacto más cercano con gente ajena al proyecto y mostrarles las características principales de nuestro proyecto, qué idea quería representar y en qué destacaba. Durante esta puesta en escena, la gente interesada pudo probar el juego y realizar al final la encuesta que nos sirvió como retroalimentación al equipo.

Figura 41. Equipo inicial de desarrollo de Frozen Out durante la feria de diciembre. De izquierda a derecha: Vicent Pla Madrid, Adrián Reina Sáez, Alejandro Jiménez Carrasco, Alejandro Gómez Noe, Tomás Ruiz Martín y Pablo López Orrios.

En preparación para la salida de la segunda demo al público (relacionada con el desarrollo del segundo MVP) se abrió una cuenta de Instagram para compartir imágenes y novedades sobre el juego. Se eligió esta red social debido a la importancia que tienen las imágenes en sus publicaciones, una característica ideal para la difusión del apartado artístico, uno de los atractivos del juego.

La difusión de la segunda demo y la prueba por parte de sus usuarios estuvo mermada por la evolución de la COVID-19, de modo que se centraron los esfuerzos en proveer de mecanismos para permitir una difusión totalmente digital. Es por ello que se preparó la página de GitHub del proyecto para albergar un espacio donde informar los usuarios de los enlaces para descargar la demo y para responder la encuesta, así como información adicional sobre los controles y una pequeña guía para poder terminar la demo en caso de confusión.

4. Diálogos

Antes de indagar en el problema, se debe tener presente el uso de los diálogos a través de su historia en el videojuego. Debido a la capacidad del medio para contar historias ficticias, el diálogo se ha convertido en un elemento fundamental.

Aquí exploraremos su uso en el contexto de los videojuegos, para diseñar y desarrollar diferentes aproximaciones posibles que amplíen la interacción entre los participantes del juego, el cual nos sirve de base a este proyecto en grupo. Este capítulo constituye el reflejo más propio y personal de mi aportación, con su consecuente necesidad de estudiar el uso en productos existentes, las técnicas y herramientas que hay disponibles, así como plantear casos de uso relacionados con la etimología de diálogo, entendido como conversación entre dos o más interlocutores.

4.1. Estado del arte

En sus inicios, cuando los juegos solían carecer de historia y las interacciones eran simples (*Pong*[55] (1972), *Space Invaders*[56] (1978)), el diálogo tenía muy pocos usos, limitándose a instrucciones directas al jugador, sin embargo, conforme la complejidad de las historias y el avance tecnológico lo permitió se ampliaron sus usos para acomodarse a las nuevas capacidades: doblaje, opciones y ramificación narrativa más pronunciada, entre otros. Aun así, en la actualidad, se recurre al diálogo escrito en multitud de proyectos de bajo coste debido a la sencillez de implantación y a su eficacia.

Medios más longevos como el cine y los libros han dedicado gran parte de sus esfuerzos en la narración, de modo que el corpus dedicado a contar y transmitir historias se puede trasladar con facilidad al medio del videojuego. Además, debido a su limitada interactividad con su audiencia, los esfuerzos de sus creativos se han volcado casi en exclusiva a esta finalidad. Esta es también una de las razones por las que Smith [15] sostiene que hay pocos estudios que hablan de las funciones del diálogo en el videojuego, argumentando, de nuevo, que ya existe una colección probada y establecida de estudios de dicha índole en el cine.

Texto

Atendiendo a las ocurrencias de diálogo en videojuegos a lo largo de su historia se puede elaborar un seguimiento de su evolución, tanto en la presentación como en sus usos. Cabe destacar que, en los primeros juegos conocidos, como *Tennis for Two*[57] (1958) o *Spacewar!*[58] (1962), no existía texto en pantalla, prefiriendo dedicar todos sus recursos en mostrar las imágenes para favorecer el novedoso aspecto de la interactividad. Las primeras instancias de diálogo en videojuegos son de tipo lúdico, consistiendo en texto estático para informar al jugador del estado de su partida, claros ejemplos son juegos de recreativa como *Pong* (1972) o *Space Invaders* (1978), donde se informa al jugador de su puntuación y de la validez de su partida («*Insert coin to play*»). Más tarde, en el espacio de las computadoras personales, fueron muy relevantes las aventuras textuales [16], donde el juego consistía únicamente de texto relatando una historia y las acciones que tomaba el jugador se elegían mediante la escritura en el teclado. *Colossal Cave Adventure*[59] (1976) y *Zork*[60] (1977) son claros ejemplos. Para la elección de opciones y la ramificación del

[55] Más información en <https://es.wikipedia.org/w/index.php?title=Pong&oldid=128326450>
[56] Más información en <https://es.wikipedia.org/w/index.php?title=Space_Invaders&oldid=127407638>
[57] Más información en <https://es.wikipedia.org/w/index.php?title=Tennis_for_Two&oldid=124558078>
[58] Más información en <https://es.wikipedia.org/w/index.php?title=Spacewar!&oldid=125095953>
[59] Más información en <https://es.wikipedia.org/w/index.php?title=Colossal_Cave_Adventure&oldid=127684919>
[60] Más información en <https://es.wikipedia.org/w/index.php?title=Zork&oldid=124090792>

diálogo, se usaba un analizador de texto que reconocía comandos escritos a través del teclado, de modo que podía reconocer palabras y frases para decidir acciones a tomar en el juego, usualmente frases compuestas por un verbo y un sustantivo (ej.: «coger cuerda», «tirar palanca»). Cuando las capacidades tecnológicas permitieron la inclusión de gráficos para acompañar al texto, la necesidad de describir textualmente los entornos y objetos se relegó, permitiendo así centrar el uso del texto en diálogos, dando comienzo a las aventuras gráficas (*Mystery House*[61] (1980) o *King's Quest I*[62] (1984)).

A mediados de los años 80, la popularización del ratón como dispositivo de entrada introdujo nuevos sistemas para la elección de opciones que pusieron en desuso los analizadores de texto. Uno de los ejemplos más relevantes es el sistema SCUMM, presentado con la salida de *Maniac Mansion*[63] (1987), con este sistema la creación de comandos se producía a través de la elección de palabras (verbos) e ítems (sustantivos) con el ratón. La simplicidad de interacción con este sistema junto con los avances gráficos facilitó la popularización de este género de juegos y la consecuente publicación de múltiples aventuras gráficas. A finales de la década de los 90, y con la popularización del juego en tres dimensiones, este sistema empezó a entrar en desuso.

Paralelamente a los desarrollos acaecidos en las aventuras gráficas, los videojuegos de rol también resultaron ser otro género que depende del diálogo de forma similar. Este género de videojuegos se nutre principalmente de los juegos de rol de mesa, donde el jugador toma el control de un personaje y sigue sus aventuras a través de un mundo fantástico junto con otros jugadores. Son característicos en este tipo de juegos la abundante exposición y desarrollo de sus personajes, con el objetivo de hacer más creíble los eventos acontecidos durante la narración de la historia. Es en este tipo de videojuegos donde se populariza la caja de texto y donde se puede rastrear el origen de muchas de las características inherentes del diálogo en videojuegos. Uno de los ejemplos que perfeccionó estas cualidades fue *Final Fantasy VII*[64] (1997). En él se pueden enumerar las siguientes características [15]: controlar un personaje permite conocer distintos puntos de vista simplemente habitando su avatar y siguiendo sus vivencias; dado que conocer la historia no es el único interés del jugador (gracias a la interactividad y juego añadido) los diálogos se pueden espaciar más que en otros medios; el jugador decide si hablar con los PNJs, de modo que el juego puede tener en cuenta la falta de información o, similarmente, se puede beneficiar de esconder ciertos PNJs. Otra de las características de este género es la inclusión de los eventos o secuencias programadas (comúnmente conocidos por su homólogo inglés *scripted events*), donde personajes entablan conversación entre ellos sin intervención del jugador, con el objetivo de crear la sensación de estar en un mundo vivo, que no necesita del jugador para existir.

Voz

El diálogo en forma de texto en videojuegos ha ido acompañado de efectos de sonido para simular la lectura o el habla de sus personajes y, en la actualidad, se puede hacer uso de actuaciones de voz para asemejarse a una conversación natural. En los inicios, los juegos tenían un almacenamiento limitado, de modo que la inclusión de actuaciones de voz no era factible, recurriendo a efectos de sonido para transmitir la sensación de voz durante una conversación. Un efecto que, combinado con la presentación del texto de forma incremental, ayuda a simular una conversación natural, donde el usuario procesa las palabras conforme las lee y escucha, ya que

[61] Más información en <https://es.wikipedia.org/w/index.php?title=Mystery_House&oldid=117419471>
[62] Más información en <https://es.wikipedia.org/w/index.php?title=King%27s_Quest_I:_Quest_for_the_Crown&oldid=118768646>
[63] Más información en <https://es.wikipedia.org/w/index.php?title=Maniac_Mansion&oldid=128032544>
[64] Más información en <https://es.wikipedia.org/w/index.php?title=Final_Fantasy_VII&oldid=128372645>

son reveladas y acompañadas por sonido. El sonido que acompaña al texto mostrado en pantalla siempre ha ido ligado a los avances en los chips de sonido[65] presentes en máquinas recreativas y, eventualmente, en computadoras personales y consolas de juego.

El objetivo de los desarrolladores de juegos era conseguir que el texto de los personajes pudiera ser hablado, por ello las primeras iteraciones de los chips de sonido tenían la capacidad de sintetizar texto, es decir, crear voz artificial a partir de texto para simular la voz humana. De esta forma se podía crear audio en tiempo real, sin necesidad de almacenar el archivo de audio completo junto con el juego. Estos chips se comercializaban como extensiones a productos existentes, claros ejemplos son el cartucho *Intellivoice Voice Synthesis Module*[66], para la consola de juegos *Mattel Intellivision*[67], sacada al mercado en 1982; la extensión para síntesis de voz para la consola de juegos *Philips Videopac G7000*[68] (también conocida como *Magnavox Odyssey 2)* de 1982; o una extensión para las computadoras personales *Atari 400* y *Atari 800*[69] llamada *Voice Box*[70] de 1982. Pese a la disponibilidad de esta tecnología, la complejidad de estos chips y el conocimiento necesario para producir voz natural ocasionaron que los desarrolladores optaran por un uso más simple de ellos para crear efectos de sonidos sin aparente sentido para representar la voz, como en uno de los primeros ejemplos de uso de esta técnica: *Q*bert*[71] (1983).

La técnica de crear sonidos aparentemente sin sentido para substituir las posibles instancias de diálogo ganó popularidad debido a su facilidad de implementación y bajo espacio de almacenamiento, tal como adoptaron juegos como *The Legend of Zelda*[72] (1986). Además, otro de los factores positivos está relacionado con la localización de estos, ya que no era necesario crear y distribuir un nueva colección de sonidos de diálogo para otros idiomas o regiones. El avance en la capacidad de los chips de sonido, también permitieron asignar un sonido diferente a cada personaje, un ejemplo de esto fue *Star Fox*[73] (1993), donde se juega con el tono y el timbre para caracterizar el sonido de cada personaje.

Junto con los avances en chips de sonido y almacenamiento de datos, surgieron los primeros juegos con actuación de voz, siendo uno de los pioneros *Resident Evil*[74] (1996) y *Metal Gear Solid*[75] (1998). Pese a las capacidades técnicas disponibles, el alto coste de producción derivado de añadir actuaciones de voz permitió a la técnica tradicional de sonidos aleatorios seguir evolucionando de forma paralela. Por una parte, *Banjo-Kazooie*[76] (1998), aun haciendo uso de tecnología moderna para almacenar voz, optó por replicar el efecto de los sonidos ininteligibles modificando y remezclando actuaciones de voz. De manera que se crea el contraste de personas reales actuando para simular un estilo de voz sintética.

---

[65] Circuito integrado que posibilita la generación de sonido. Más información en <https://es.wikipedia.org/w/index.php?title=Chip_de_sonido&oldid=125829942>
[66] Más información en <https://en.wikipedia.org/w/index.php?title=Intellivoice&oldid=937333570>
[67] Consola de juegos elaborada por la empresa Mattel y lanzada en 1979. Más información en <https://en.wikipedia.org/w/index.php?title=Intellivision&oldid=968225862>
[68] Consola de juegos elaborada por Magnavox (subsidiaria de Philips) lanzada en 1978. Más información en <https://en.wikipedia.org/w/index.php?title=Magnavox_Odyssey_2&oldid=964965762>
[69] Computadoras personales elaboradas por Atari en 1979. Más información en <https://en.wikipedia.org/w/index.php?title=Atari_8-bit_family&oldid=969022045>
[70] Más información en <https://atari8bitads.blogspot.com/2016/11/atari-says-its-first-word.html>
[71] Más información en <https://es.wikipedia.org/w/index.php?title=Q*bert&oldid=124058662>
[72] Más información en <https://es.wikipedia.org/w/index.php?title=The_Legend_of_Zelda_(videojuego)&oldid=128320172>
[73] Más información en <https://es.wikipedia.org/w/index.php?title=Star_Fox&oldid=127996743>
[74] Más información en <https://es.wikipedia.org/w/index.php?title=Resident_Evil&oldid=128319310>
[75] Más información en <https://es.wikipedia.org/w/index.php?title=Metal_Gear_Solid&oldid=127694317>
[76] Más información en <https://es.wikipedia.org/w/index.php?title=Banjo-Kazooie&oldid=126994283>

Por otra parte, hubo juegos que, descartando las actuaciones de voz, tomaron la técnica de los efectos de sonido para crear su propio idioma fonético, es decir, crear una correspondencia ente letras y fonemas en el texto con un sonido específico. De esta forma, se mantiene la conexión entre el texto y el habla, ya que el sonido va a depender del diálogo y este va a mantener una prosodia[77] familiar, pero los sonidos se mantienen ininteligibles ya que el oyente no es capaz de discernir las palabras. En este campo se encuentra el idioma «simlish» proveniente de la saga *The Sims*[78] (2000), donde cada idea o emoción se corresponde con un sonido, el cual consta de remezclas de actuaciones de voz reales tal que el resultado final es inteligible pero tiene semejanza con el idioma ingles original; y el idioma «animalese» procedente de la saga *Animal Crossing*[79] (2001), donde la técnica usada es la pronunciación estricta de cada fonema presente en el texto (ej. la palabra «Hola» se pronunciaría «H»-«o»-«ele»-«a»). Ambos idiomas consiguen mantener la familiaridad del jugador al mantener el ritmo y pausas de la pronunciación, si bien los sonidos y el resultado total no se puede comprender ni se corresponde con un idioma conocido.

4.1.1. Conclusiones

A partir del análisis del estado del arte y las ocurrencias de diálogo se pueden extraer consideraciones y características que afectan el desarrollo de los diálogos en videojuegos.

Niveles de comunicación

En Domsch [17] se especifican tres niveles de comunicación presentes en el videojuego, a través del cual se pueden clasificar las diferentes instancias de diálogo, estas son: la comunicación lúdica, la comunicación diegética y la comunicación mixta, vista esta como un tercer nivel derivado de la mezcla de las dos anteriores de forma simultánea. Además de estos niveles de comunicación, se distinguen dos subtipos atendiendo a la dirección de la comunicación, los cuales permiten un mayor desglose de los diferentes diálogos en cada nivel. De modo que se tiene en cuenta la comunicación: *game-to-player*, de juego a jugador, donde elementos relativos al juego (interfaz, personajes, ...) se dirigen al jugador, y viceversa; *player-to-player*, de jugador a jugador, donde el jugador se comunica con otros jugadores a través de herramientas proporcionadas por el videojuego. Cabe destacar que se ignora la comunicación *game-to-game*, ya que, además de que no incluye al jugador, hace referencia a mecanismos y sistemas internos de comunicación de datos y estados entre componentes del juego. Profundizando en los niveles mencionados, el primero de ellos, el nivel lúdico, se refiere al que se dirige directamente al jugador y lo trata como persona real (reconociéndola como aquella que está interactuando con el videojuego). Atendiendo a la dirección de la comunicación, la correspondiente de juego a jugador tiene en cuenta los mensajes explicativos o que retroalimentan el jugador sobre acciones que está tomando, de modo que suelen hacer referencia a actividades propias del acto de jugar; por otro lado, los mensajes de jugador a jugador serían los provenientes de un canal de comunicación interno, ya sea de texto o de voz, donde los jugadores pueden entablar conversaciones libres entre ellos, es decir, sin estar necesariamente centradas en el juego.

El nivel diegético habla de aquel que transcurre dentro de las limitaciones ficticias del videojuego y hace referencia a conceptos de ese mundo ficticio. Mientras que el nivel lúdico tenía como función principal comunicar las reglas de juego al jugador e informarle sobre sus acciones, el nivel diegético tiene la función de exponer la historia al jugador y hacerle participe de los

[77] Estudio fonético y fonológico de los elementos que se refieren a unidades superiores al fonema. Extraído de <https://dle.rae.es/prosodia>.
[78] Más información en <https://es.wikipedia.org/w/index.php?title=Los_Sims&oldid=128342005>
[79] Más información en <https://es.wikipedia.org/w/index.php?title=Animal_Crossing_(videojuego)&oldid=126951209>

acontecimientos que transcurren en ese mundo que habita su avatar. Considerando de nuevo la dirección de la comunicación, cuando esta es de juego a jugador se basa en la transmisión de mensajes de aspecto narrativo a través de elementos del mundo ficticio, aquí pueden entrar tanto conversaciones con otros personajes como la lectura de información en un libro que habita en ese mundo. Este tipo de comunicación es el que más se nutre de lecciones y prácticas de otros medios como el cine o los libros. Hablando ahora de la comunicación de jugador a jugador, como se ha comentado, existen videojuegos que permiten la comunicación libre entre jugadores, la cual suele derivar en conversaciones lúdicas sobre temas externos al juego en cuestión. No obstante, la opción más común para mantener la conversación sobre conceptos del mundo del juego es limitar la comunicación y sus opciones. Usualmente esto significa integrar la comunicación entre jugadores dentro de mecánicas de juego, incluso omitiendo el hecho de que la comunicación se está produciendo entre jugadores reales, de modo que los jugadores solo perciben el mensaje a través del avatar del otro jugador o de otros elementos del mundo ficticio (para más información refiérase a casos como *Journey*[80] (2012) o *Dark Souls*[81] (2011)).

El tercer y último nivel, el mixto, se usa para clasificar aquellas interacciones que combinan los niveles anteriormente descritos, es el caso en el que el mensaje se dirige tanto al jugador (lúdico) como a su avatar (diegético). Aquí entran situaciones en el que los personajes toman conciencia de su propio papel y se dirigen al jugador directamente (acto comúnmente conocido como romper la cuarta pared[82]). Por otro lado, también entran casos en los que mensajes de juego e información usualmente relegada a menús se integran dentro de conversaciones de personajes del mundo ficticio. Esta técnica favorece la inmersión del jugador ya que reduce la consulta de interfaces de juego y mantiene el enfoque en el mundo ficticio, es decir, reduce la presencia explicita de comunicación lúdica.

Cabe señalar también en este apartado que, aunque las conversaciones de los personajes tienen como receptor al avatar del jugador, la mayoría de la información comunicada va a ser recibida y procesada por el jugador, ya que es el receptor final que interactúa a través de su avatar. De modo que la comunicación diegética siempre va a necesitar del jugador para ser completa.

Relación con el estilo

Tal como se describe en Stoeber [18], con la salida al mercado de *Banjo-Kazooie* y su coexistencia con otros videojuegos que hacían uso de actores de voz, se empezó a percibir cómo la elección de actuaciones de voz frente a los efectos ininteligibles de sonido afectaba al estilo y tono que los creadores querían transmitir sobre su juego. De este modo, se relacionó la actuación de voz para contar una historia más seria y aplicable al mundo real (como en los casos de *Resident Evil* con un drama postapocalíptico y *Metal Gear Solid* con la problemática de la guerra), mientras que el diálogo usando síntesis de voz o efectos de sonido fue más preferible para una historia con un tono más alegre y humorístico (como en el caso de *Banjo-Kazooie* junto con su estética de dibujo animado e historia fantástica).

En la actualidad, de hecho, se mantienen estos estereotipos, siendo este un punto más que sopesar en la generación del videojuego. Esto ha llevado a la coexistencia de productos de alto

[80] Más información en <https://es.wikipedia.org/w/index.php?title=Journey_(videojuego)&oldid=125684793>
[81] Más información en <https://es.wikipedia.org/w/index.php?title=Dark_Souls&oldid=127857124>
[82] Acto común en el medio del cine y la literatura en el que personajes de la narración se dirigen al lector o audiencia, reconociendo su presencia como espectador. Más información en <https://es.wikipedia.org/w/index.php?title=Cuarta_pared&oldid=126438524>

presupuesto (*The Last of Us Part II*[83] (2020) y *Ghost of Tsushima*[84] (2020)), donde se utiliza el soporte de voz para transmitir seriedad y favorecer al realismo de los personajes y su historia, junto con títulos en los que se utiliza sonido (aparentemente aleatorio) para favorecer una historia más ligera y con estética desenfada (*Animal Crossing: New Horizons*[85] (2020) o *A Short Hike*, (2019)). Cabe destacar que está última opción sigue siendo preferible por estudios independientes debido a su bajo coste de implementación y su comprobada eficacia.

4.2. Uso de diálogos realizado en el primer MVP

Previamente a ahondar en la reestructuración y extensiones realizadas durante el segundo MVP, periodo de desarrollo donde se van a centrar las explicaciones técnicas de esta memoria, cabe destacar el trabajo realizado en cuanto a diálogos para el primer MVP, con el objetivo de establecer la base de desarrollo sobre la que se trabajó posteriormente.

4.2.1. Alternativas de Texto

Al principio del desarrollo se puso gran énfasis en encontrar una solución eficaz y flexible para manejar los diálogos y, en consecuencia, la forma de contar la historia, dado que se trata de un juego donde las conversaciones y la trama es de alta relevancia. Dado que el inicio del desarrollo estuvo dirigido por las clases prácticas, y el material docente práctico estaba centrado en el desarrollo a través de la herramienta Unity, se optó por la búsqueda e integración de una herramienta externa (*plugin*). Esta herramienta, además, tendría la ventaja de ser una solución existente y probada por otros desarrolladores y contaría con ejemplos prácticos en proyectos reales. De este modo se investigaron diferentes opciones y se consideraron las tres más populares: Twine[86], Ink[87] y Yarn Spinner.

Previamente a la elección, sin embargo, se elaboró una lista de características óptimas que este sistema debía cumplir: guardar del guion de texto de forma separada del código, tal que se pudiera ver y editar sin necesitar conocimiento de programación; permitir editar el estilo del texto, en especial el tamaño, y efectos como el color, la cursiva y el grosor; ofrecer opciones para crear narrativas ramificadas, es decir, controlar la sucesión y flujo del texto en base a condiciones de juego. La simplicidad de estas opciones debía ser clave, ya que facilitaría el trabajo de los compañeros de diseño, los cuales estaban a cargo de la elaboración del guion. A continuación, se detallan las alternativas consideradas durante el proceso de búsqueda y el porqué de la elección final.

Twine

Twine es una herramienta de creación de historias interactivas creada por Twinery. En esta herramienta los guiones se conocen como historias, dentro los cuales se incluyen los diálogos, conocidos como pasajes, y las conexiones entre ellos (a través de opciones y condiciones). Twine soporta el guardado de variables y su uso para condicionar las conversaciones. Sin embargo, su soporte se limita a su aplicación web, donde las historias creadas pueden cobrar vida en forma de páginas webs interactivas. Cuenta con un editor donde se visualizan los pasajes (Figura 42) para abstraer parte del formato del archivo fuente, no obstante, el formato de cada conversación (condicionalidad, acceso a variables, entre otros) debe ser escrito de forma explícita.

[83] Más información en <https://es.wikipedia.org/w/index.php?title=The_Last_of_Us_Part_II&oldid=128330374>
[84] Más información en <https://es.wikipedia.org/w/index.php?title=Ghost_of_Tsushima&oldid=128138003>
[85] Más información en <https://es.wikipedia.org/w/index.php?title=Animal_Crossing:_New_Horizons&oldid=127808362>
[86] Disponible en <https://twinery.org/>
[87] Disponible en <https://www.inklestudios.com/ink/>

Para posibilitar la integración de estas historias dentro de Unity, existe la herramienta Cradle[88], la cual toma el archivo fuente generado por Twine y proporciona la funciones para acceder a cada pasaje y extraer sus líneas de texto. De este modo, se ofrece una separación clara entre los escritores que pueden centrar sus esfuerzos en la edición a través del editor, y los desarrolladores los cuales trabajan con Cradle para implementar las cuestiones técnicas derivadas de mostrar y controlar el diálogo. Cradle también elimina la necesidad de establecer un pasaje inicial, ya que en un videojuego es necesario acceder a conversaciones sin un orden concreto.

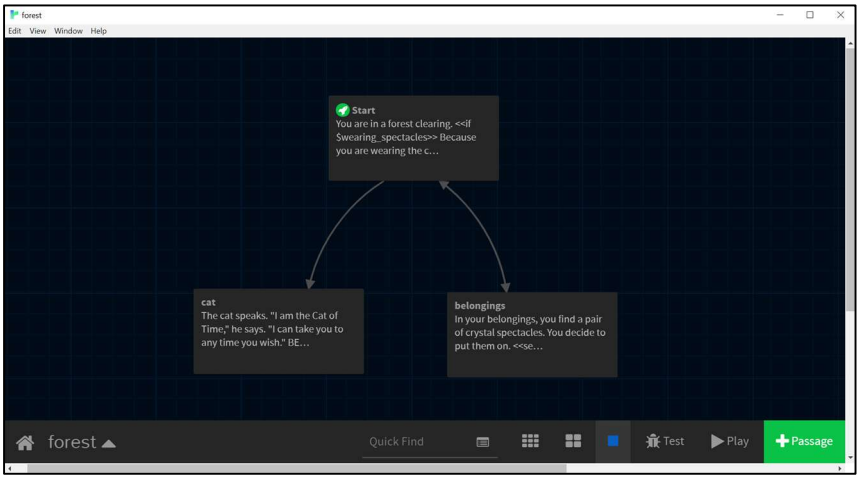

Figura 42. Captura de pantalla de la aplicación Twine con un archivo de muestra abierto. Elaboración propia.

Yarn Spinner

Yarn Spinner [14] es una herramienta y lenguaje de creación de diálogos desarrollado por el estudio thesecretlab. Este sistema de diálogos fue creado originalmente para el videojuego *Night in the Woods* (2017), el cual posteriormente desacoplaron y publicaron como herramienta independiente. Sus desarrolladores la describen como un analizador sintáctico, el cual lee el guion contenido en un archivo de texto y suministra las líneas de texto según son pedidas por el usuario. Este archivo de texto debe seguir un formato establecido (llamado comúnmente *scripting language*), del mismo modo que Twine, donde se estructura en nodos, entendidos como un conjunto de líneas entre personajes (similar a los pasajes de Twine).

Yarn Spinner también ofrece soporte para expresiones condicionales, así como la configuración de variables de diálogo y su uso en estas expresiones. La integración de Yarn Spinner dentro de Unity es prácticamente nativa al estar desarrollada originalmente para ella, sin embargo, este solo se encarga de proveer la línea de diálogo deseada, la presentación en pantalla y gestión de variables no se proporciona y debe implementarse para cada proyecto. De este modo también se pueden diferenciar, por una parte, la escritura del guion y, por otra parte, la integración de la herramienta en Unity. Cabe destacar que la edición del guion se puede elaborar en su editor web (Figura 43), el cual visualiza las conversaciones (nodos) como cajas, abstrayendo el formato para separar los nodos, pero para las conversaciones de cada nodo es necesario saber el formato de condiciones y acceso de variables.

[88] Disponible en <https://github.com/daterre/Cradle>

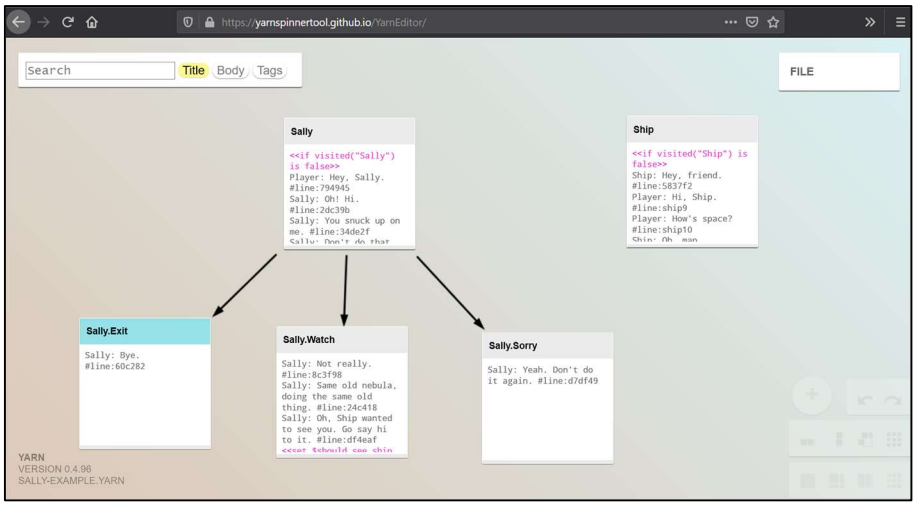

Figura 43. Captura de pantalla de la aplicación web Yarn Spinner Editor con un archivo de muestra abierto. Elaboración propia.

Ink

Ink es un lenguaje para la creación de diálogos desarrollado por Inkle Studios. Este lenguaje de diálogos permite el control del flujo del diálogo a través de expresiones condicionales y uso de variables. El guion está estructurado en historias, las cuales contienen un conjunto de líneas formando una conversación independiente. Cada conversación, junto con sus opciones y ramificaciones, constituye una historia y, por ende, un archivo distinto.

Para facilitar la creación y edición de los archivos fuente, el estudio ofrece también la aplicación Inky, la cual consta de un visor de historias para comprobar en tiempo real el flujo de diálogo del guion abierto, tal como se puede ver en la Figura 44. Sin embargo, este editor no abstrae el lenguaje de Ink, de modo que sería necesario conocer el formato para la creación de diálogos. Para la integración con Unity existe Ink-Unity integration[89], el cual permite la conversión del archivo fuente a un formato JSON y posibilitar la ejecución de las historias deseadas. La gestión y presentación de estas líneas al jugador se deja en manos de cada equipo de desarrollo, similar a las otras herramientas discutidas.

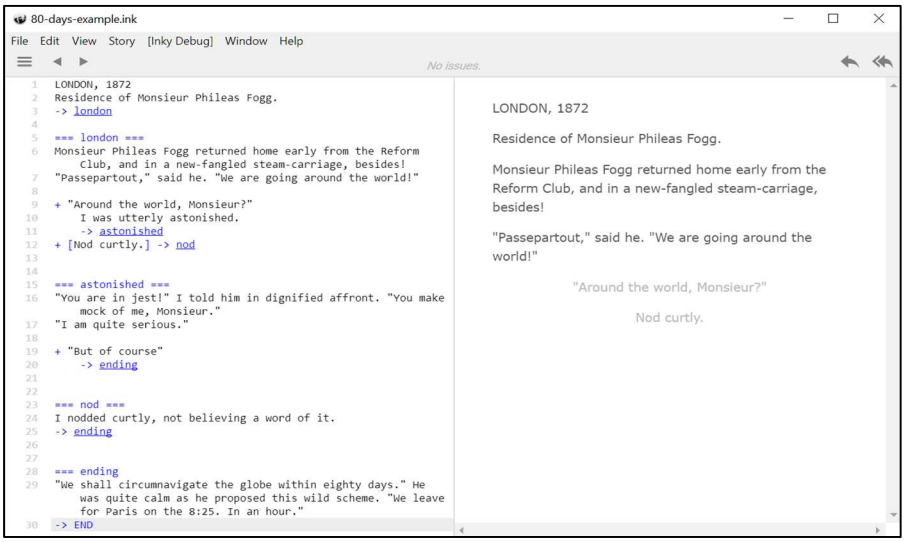

Figura 44. Captura de pantalla de la aplicación Inky, el editor visual de Ink, con un archivo de muestra abierto. Elaboración propia.

[89] Disponible en <https://github.com/inkle/ink-unity-integration>

Conclusiones

Una vez descritas las herramientas, se realizó una tabla comparativa (Tabla 5) para comprobar a simple vista el cumplimiento de los objetivos señalados. Cabe destacar que todas las opciones propuestas son de código abierto y de uso gratuito, de modo que pueden ser fácilmente expandidas. Se ha incluido también la simplicidad de integración, recogiendo los aspectos más relevantes de la documentación de cada herramienta, así como de comparativas como la realizada por Neil [19].

	Twine	Yarn Spinner	Ink
Guion guardado de forma independiente	Sí	Sí	Sí
Estilo del texto	La presentación del texto no se cubre por ninguna de las herramientas, pero permiten etiquetar el texto para su posterior uso.		
Opciones para crear narrativas ramificadas	Sí	Sí	Sí
Simplicidad de integración	Con Cradle y llamada a sus métodos públicos. Variables de diálogo gestionadas internamente.	Nativa, inclusión de componentes en proyecto y en escena. Personalización de guardado de variables.	Con Ink-Unity integration y llamada a sus métodos públicos. Personalización de componentes.
Juegos relevantes desarrollados con la herramienta	-	*A Short Hike* (2019) *Night in the Woods* (2017)	*80 Days*[90] (2015) *Heaven's Vault*[91] (2019)

Tabla 5. Tabla comparativa de sistemas de texto. Elaboración propia.

En general, todas las alternativas propuestas cumplen con los objetivos y características que se buscaban: el guardado en un archivo independiente y las opciones para ramificar el diálogo son las características más populares y demandadas por los equipos de desarrollo; y la inclusión de un editor visual también suele ser un añadido común para abstraer parte del aprendizaje. En cuestión de integración, las herramientas poseen de diversas facilidades para adecuar su sistema a cada desarrollo (principalmente, la especialización de clases base y las llamadas a métodos públicos). Debido a la necesidad de nuestro proyecto para acceder a las conversaciones de forma aleatoria, se consideró también un análisis de los tipos de juego que más se favorecen con cada herramienta, comprobando los juegos existentes desarrollados usando cada una.

En primer lugar, nos encontramos con que Cradle no era muy popular en la creación de juegos y no se encontró ningún juego de renombre o, al menos, que afirmaba públicamente usar la herramienta. Por otro lado, Inkle es una herramienta popular en juegos basados exclusivamente en torno a la interacción a través del diálogo, sirviendo este de motor a través del cual se controla el juego. *80 Days* o *Heaven's Vault* son claros ejemplos: juegos narrativos donde la mecánica principal se desarrolla a través de conversaciones. Sin embargo, el control de las variables de diálogo está restringido (siendo variables que existen dentro de un nodo, pero cuyo valor sería perdido al cambiar de conversación), de modo que limita el acceso a nodos de conversación de forma aleatoria. Finalmente, en el caso de Yarn Spinner se encontró que dos de los referentes considerados en el estudio de mercado (capítulo 2.2), *A Short Hike* y *Night in the Woods*, hacían uso de la herramienta. Ambos son juegos de aventuras donde las mecánicas como movimiento

[90] Más información en <https://en.wikipedia.org/w/index.php?title=80_Days_(2014_video_game)&oldid=960979884>
[91] Más información en <https://en.wikipedia.org/w/index.php?title=Heaven%27s_Vault&oldid=973641476>

del jugador o uso de ítems son centrales, y tienen la misma importancia que el diálogo, el cual puede ser relegado a un segundo plano para acceder a cada nodo bajo demanda y manteniendo el estado de las variables entre conversaciones. De este modo, finalmente se eligió Yarn Spinner.

4.2.2. Integración de Yarn Spinner

La herramienta de Yarn Spinner consta de dos partes: una parte privada, la cual se encuentra precompilada y sirve de librería para las clases internas de la herramienta; y una parte pública, donde se encuentran las clases usables desde Unity que contienen el funcionamiento necesario para accionar el flujo de texto y controlar el diálogo. La estructura de esta parte se puede consultar en la Figura 45 y consta de una clase principal *DialogueRunner*, la cual representa el elemento que administra el sistema: en él se establecen los archivos donde se encuentran los diálogos y contiene métodos para iniciar una conversación. La parte pública parte contiene, además, la definición de dos clases abstractas que subsumen la funcionalidad de guardado de variables de diálogo (*VariableStorageBehaviour*) y de la presentación del texto en pantalla (*DialogueUIBehaviour*). Cabe destacar que, aunque la herramienta también ofrece versiones concretas de estas clases, se trata de implementaciones básicas para demostrar su funcionamiento y se recomienda crear implementaciones personalizadas para adaptar su funcionamiento a las particularidades de cada proyecto.

El correcto funcionamiento de *DialogueRunner* depende de la implementación de estas dos clases, ya que este contiene una referencia a cada una de ellas y va a interactuar con ellas para llevar a cabo el flujo de texto. La clase más relevante en este aspecto es *DialogueUIBehaviour*, la cual contiene métodos que van a ser invocados por *DialogueRunner* para comunicar las líneas de texto y las opciones de diálogo. La implementación de estos métodos afectará la forma de mostrar la conversación en pantalla. En cuanto a *VariableStorageBehaviour*, actúa de forma similar, ofreciendo métodos para obtener y establecer variables, así como para reestablecer y borrar las variables guardadas, que también son llamadas internamente por *DialogueRunner* cuando desea conocer el valor de una variable.

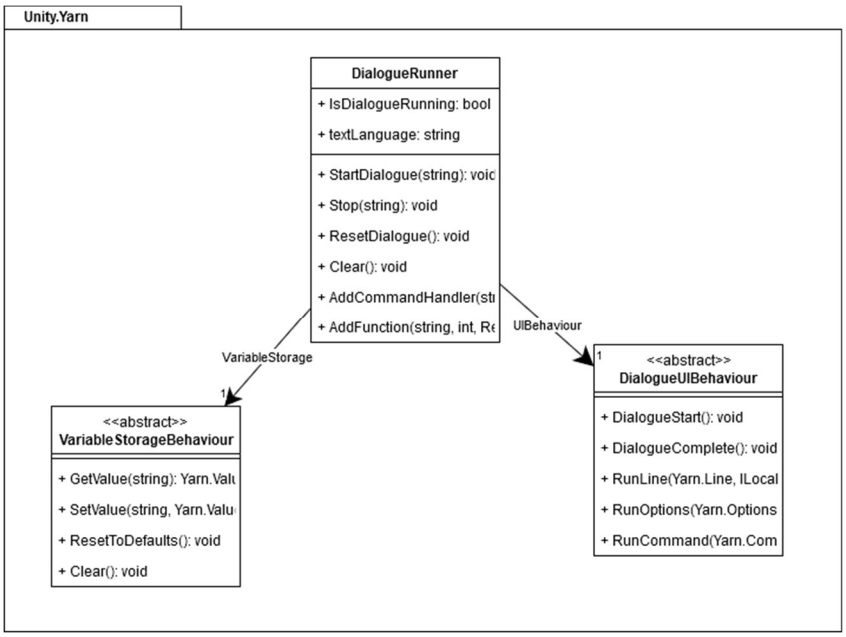

Figura 45. Diagrama de clases UML mostrando las clases públicas de Yarn Spinner. Elaboración propia.

El flujo de texto se realiza, por tanto, de la siguiente forma: inicialmente, el usuario especifica el nodo a correr usando *DialogueRunner.StartDialogue(string)*, a partir de este momento se van a

proporcionar las líneas de texto y elecciones a través de llamadas internas a *DialogueUIBehaviour*; para poder recibir las siguientes líneas, *DialogueUIBehaviour* debe comunicar que ha finalizado de presentar la línea de texto llamando a una función proporcionada llamada *onComplete()* (usualmente llamada tras esperar la interacción del jugador a través de la pulsación de una tecla). Este último paso se repetirá hasta que el sistema comunique a *DialogueUIBehaviour* el fin de la conversación.

De modo que, para finalizar la integración de la herramienta se debieron crear implementaciones concretas de estas dos clases base. En primer lugar, se creó *YarnDialogueBehaviour* con el objetivo de implementar la clase abstracta de *DialogueUIBehaviour*. Tomando como base el ejemplo proporcionado por la herramienta, la implementación toma la línea proporcionada y la comunica de letra en letra a la interfaz, y para permitir el paso a las subsecuentes líneas se espera a la interacción del jugador pulsando cualquier tecla. Además, con el objetivo de mostrar el nombre del personaje junto con su diálogo se estableció que cada línea debía contener el nombre siguiendo este formato: «Nombre: Diálogo». De modo que, una vez separados nombre y diálogo, se actualizan los respectivos componentes de interfaz. Los métodos referentes al control de las opciones de diálogo y el tratamiento de comandos se omitieron dado que eran características de las que no se iba a hacer uso durante este periodo de desarrollo.

En segundo lugar, de forma similar, la implementación de *VariableStorageBehaviour* tomó como base el ejemplo concreto proporcionado, el cual hacía uso de un diccionario para guardar las variables, una estructura de datos de acceso en un tiempo constante ($\Theta(1)$), de modo que se mantuvo en nuestra clase concreta *YarnVariableStorage*. Los métodos para reestablecer las variables se limitaron a limpiar el contenido de este diccionario.

4.2.3. Utilidades de texto

Durante el primer MVP, se pensó en la posibilidad de mostrar el texto con estilo, similar a ejemplos referentes como *Animal Crossing* (2001) o *A Short Hike* (2019), tal que se pudiera alterar el tamaño o color de palabras. Los componentes de texto de Unity permiten el formato de etiquetas en el lenguaje de marcado RichText[92] (RTML), de modo que se optó por usar estas, añadiéndolas manualmente en el guion de Yarn Spinner. Unity soporta principalmente el color, el tamaño, la cursiva y la negrita [20]. Sin embargo, el uso de estas etiquetas combinadas con el mostrado del texto de letra en letra entraba en conflicto: los separadores de inicio y fin de la etiqueta también son proporcionados en el orden en que aparecen, de modo que hasta que el fin de la etiqueta es proporcionado se pueden visualizar las etiquetas en pantalla. En la Figura 46 se puede comprobar este problema en Unity: tomando como línea de texto «Desde que el alcalde Mc Topping» se puede ver como las etiquetas de inicio son visibles hasta que la etiqueta de fin ha sido proporcionada, momento en el que Unity aplica automáticamente el estilo de cursiva al texto encapsulado. Para solucionar este fallo se determinó que, en el caso de encontrar una palabra contenida en una etiqueta, las letras se deberían mostrar una a una pero rodeadas de la etiqueta completa. Con este objetivo se elaboraron una serie de clases que permitieran clasificar el texto en texto simple y texto etiquetado.

[92] Un formato de etiquetas proveniente y basada en la especificación de HTML disponible en <https://www.w3schools.com/html/html_formatting.asp>. Más información del uso práctico de este lenguaje de marcado en otras aplicaciones en <https://www.provue.com/panoramax/help/graphics_rich_text.html>. No confundir con el formato de archivo RTF (<https://en.wikipedia.org/w/index.php?title=Rich_Text_Format&oldid=971285920>).

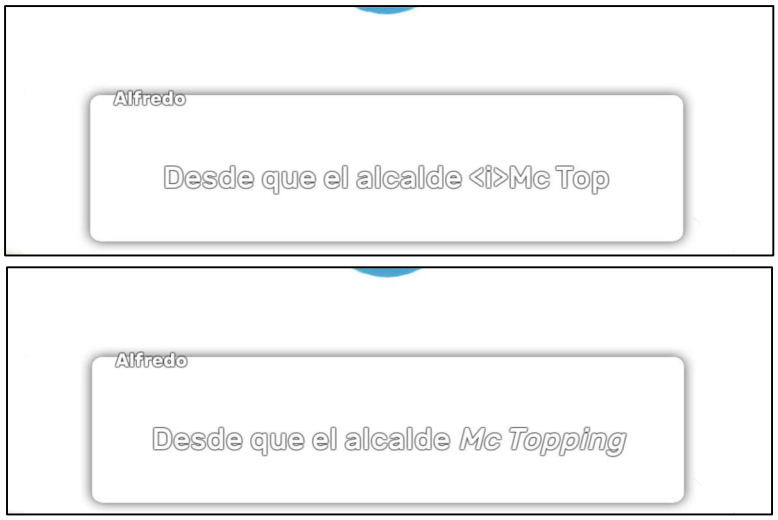

Figura 46. Capturas de pantalla de Unity ejemplificando la problemática del texto con etiquetas (arriba texto empezando sección con etiquetas visible, abajo texto con etiqueta finalizado y estilo en negrita activo). Elaboración propia.

En primer lugar, se elaboró una interfaz *IDialogueText* para abstraer las clases concretas. Esta interfaz consta de dos métodos para añadir texto asociado: *AddText(string)* y *AddText(IDialogueText)*, así como un método para proporcionar el texto de letra en letra: *Parse()* que devuelve una lista de *string* con el texto creciendo de forma acumulada ([“H”, “Ho”, “Hol”, “Hola”]). Las clases concretas fueron las siguientes: *DialogueText*, la forma más simple de texto, ya que no contiene etiquetas, de modo que el texto asociado es de tipo *string* y la implementación de *Parse()* sería trivial, iterando sobre los caracteres del texto; *DialogueTaggedText* representa un trozo de texto con una etiqueta, de modo que para guardar el texto se guarda un elemento de tipo *IDialogueText* representando el texto dentro de la etiqueta y un elemento para guardar el tipo de etiqueta (explicado con más detalle en el siguiente párrafo); y *ComplexDialogueText*, el cual está formado por una lista de *IDialogueText* (implementando así patrón compuesto) y para la implementación de *Parse()* delegaría su funcionamiento al *Parse()* de cada elemento de su lista de *IDialogueText*. En la Figura 47 se puede apreciar la estructura de estas clases.

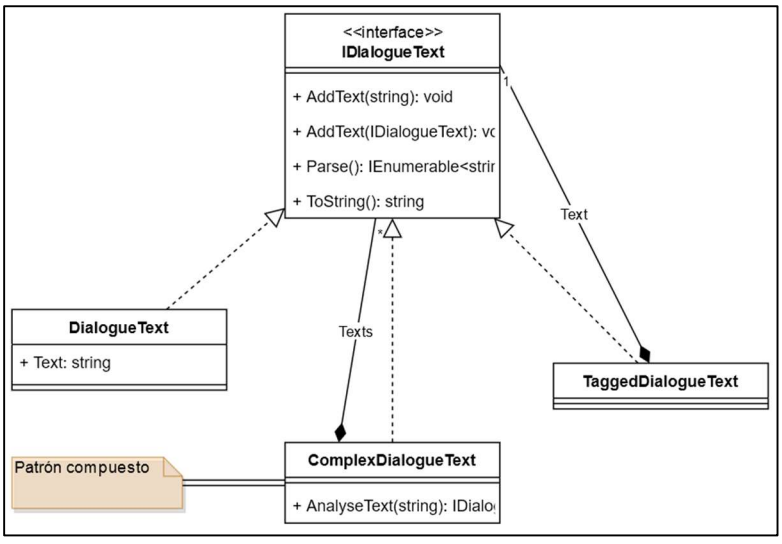

Figura 47. Diagrama de clases UML con componentes relacionados con IDialogueText. Elaboración propia.

Para guardar las etiquetas de texto, se elaboraron de forma similar una serie de clases con el objetivo de representar su anatomía, la cual se puede comprobar en la Figura 48. Por un lado, *TagOption* contiene la opción de la etiqueta, en el caso de *RichText* serían «color=green» o «b».

Por otro lado, *TagFormat* representa el formato de la etiqueta, es decir, cuáles son sus separadores para identificarlo en el texto, en el caso de RichText el formato de la etiqueta es con los caracteres «<» y «>», así como el carácter «/» para denotar el fin de la opción. Finalmente, se incluye *TagType*, el cual representa una etiqueta al completo, de modo que contiene una propiedad de tipo *TagOption* y dos de tipo *TagFormat* (para representar el formato del principio y del final, respectivamente). De este modo, *DialogueTaggedText* guarda una propiedad de tipo *TagType* para denotar la etiqueta que encapsula su texto asociado. La implementación de su método *Parse()* ahora iterará sobre el *Parse()* de su *IDialogueText* interno y encapsulará cada componente recibido con las etiquetas de su propiedad de tipo *TagType*. El hecho que *DialogueTaggedText* contenga una propiedad de tipo *IDialogueText*, en lugar de tipo *string*, es para permitir la anidación de texto con diferente etiqueta. Con esta colección de clases que permiten clasificar y agrupar partes del diálogo dependiendo de su contenido, se puede resolver el problema, ya que ahora cada letra será devuelta con su etiqueta.

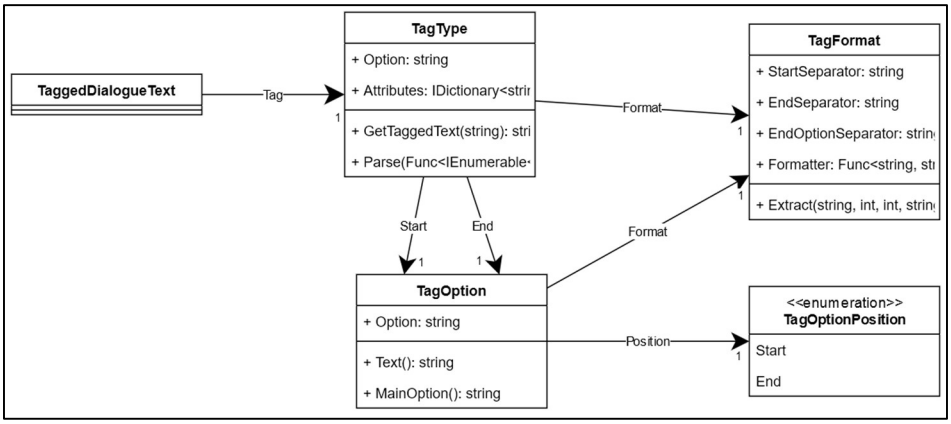

Figura 48. Diagrama de clases UML relacionado con la representación de etiquetas. Elaboración propia.

Finalmente, para ofrecer un punto de entrada para analizar el texto y clasificar sus partes en las clases presentadas se elaboró un método *AnalyseText(string)*, donde el parámetro de tipo *string* representa el diálogo. Este método realiza un análisis del texto proporcionado en busca de etiquetas para poder crear objetos de tipo *DialogueText* o *TaggedDialogueText*, así como combinar los diferentes objetos dentro de un elemento de tipo *ComplexDialogueText*. En la Figura 49 se puede comprobar el desglose generado por este análisis. En cualquier caso, la clase que solicite este análisis recibirá un elemento de tipo *IDialogueText* al que podrá usar *Parse()* para recibir correctamente cada incremento de texto. *YarnDialogueBehaviour* usa, por tanto, este análisis para posteriormente poder actualizar los componentes de interfaz.

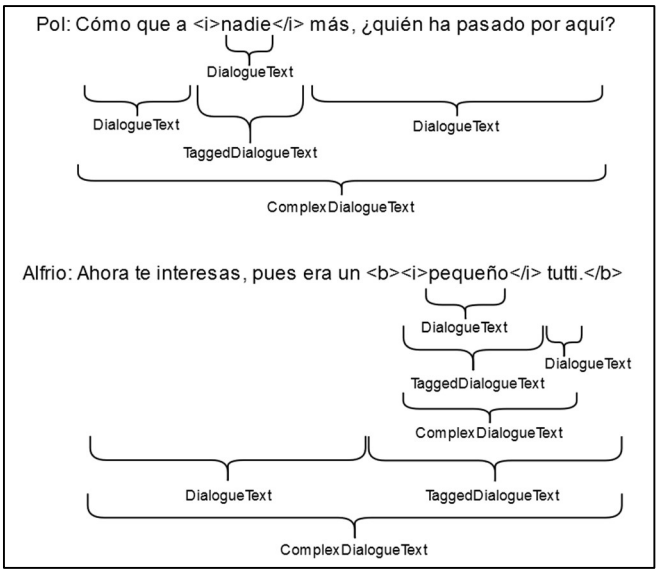

Figura 49. Desglose de líneas de diálogo en componentes de IDialogueText generado por el método AnalyseText. Elaboración propia.

4.2.4. Integración del guion

En la Figura 50 se puede ver un fragmento del guion utilizado durante el primer MVP. Aquí se pueden apreciar tres características: el formato proveniente de Yarn Spinner, denotando el nombre del nodo en *title* y encapsulando la conversación entre la cadena de caracteres «---» y «===», así como un ejemplo del uso de condicionales «*<< if visited("Historia1") is false >>*» para, en este caso, crear diálogo que solo se va a mostrar la primera vez que se ejecuta este nodo; el formato autoimpuesto de «Nombre: Diálogo» para poder posteriormente separar el nombre y el diálogo; y, finalmente, las etiquetas de RichText en «*tutti*» para dar estilo individual a palabras del texto.

```
1  title: Historia1
2  ---
3  <<if visited("Historia1") is false>>
4      Yell: Oye, se ha ido un <b>tutti</b> colina arriba.
5      Yell: Salía de tu cubitera, parecía ir con prisa.
6      Pol: De verdad? mm...
7      Pol: (Debería ver donde ha ido)
8      Pol: (sigue siendo mejor plan que ir a trabajar)
9  <<else>>
10     Yell: No he visto mucho más, lo siento
11     Pol: (Debería ver donde ha ido)
12 <<endif>>
13 ===
```

Figura 50. Fragmento de guion para el nivel. Elaboración propia.

4.3. Diseño y desarrollo de un Gestor de Diálogos

Durante la preparación del segundo MVP se encontró que la mayoría de los añadidos y extensiones relacionadas con el sistema de diálogos se beneficiarían del desacoplamiento de la herramienta actual, Yarn Spinner. Para ello se comprobaron la lista de características existentes de la herramienta, con la finalidad de extraerlas para crear la base de un diseño abstracto. Este diseño, sobre todo, eliminaría la necesidad de que el resto de los componentes de juego conozcan la herramienta concreta que se está haciendo cargo de la implementación.

En primer lugar, se creó una clase abstracta *DialogueManager*, la cual estaría encargada de agrupar los subsistemas necesarios y exponer los métodos para actuar con estos. Inicialmente se introdujeron métodos para empezar y parar el diálogo, así como para conocer si estaba en funcionamiento. El método para empezar el diálogo ahora necesitaría de un parámetro de tipo

DialogueActer, en lugar de uno de tipo string, con el objetivo de eliminar las similitudes con métodos provenientes de *DialogueRunner* de Yarn Spinner, así como permitir la futura entrada de las otras herramientas comentadas en el capítulo 4.2.1. La clase *DialogueActer* también consta de una propiedad de tipo *string*, pudiendo representar el nodo de Yarn Spinner o el pasaje de Ink, sin embargo, ahora otras clases derivadas de esta podrían extender sus características para representar un tipo de diálogo distinto, tal como se describirá en el capítulo 4.7. Además, dado que *DialogueActer* estaría asociado a un personaje, consta de métodos que se ejecutarán una vez se cumplan ciertas condiciones, con el objetivo de comunicar al personaje estos cambios: p. ej., cuando se ha va a empezar el diálogo con su nodo y cuando se ha terminado.

En segundo lugar, se creó una clase *DialogueSystem* para abstraer la funcionalidad del sistema encargado de guardar y proporcionar las líneas de diálogo, especialmente diseñado para abstraer la funcionalidad de *DialogueRunner* de Yarn Spinner, así como otras clases similares en Cradle o Ink. Atendiendo a las funcionalidades de *DialogueRunner*, se introdujeron en *DialogueSystem* métodos para: iniciar y parar el diálogo, establecer y consultar variables de diálogo, y para marcar el idioma de texto. También se incluyeron métodos complementarios para consultar el estado del sistema y para solicitar la siguiente línea de diálogo.

Los sistemas de diálogo descritos en el capítulo 4.2.1 cuentan con la característica de funcionar a través de un guion de texto estático, es decir, las líneas de texto deben ser conocidas de antemano y no se pueden modificar durante el juego. Existen, sin embargo, mecanismos por los que la ejecución del diálogo se puede pausar, en cuyo caso sí se podrían modificar el número de líneas en tiempo de ejecución, introduciendo información proveniente de un servicio web u otra fuente, para posteriormente continuar con la ejecución del sistema de diálogos. De modo que, llegados a este punto, se pueden discernir dos tipos de sistemas de diálogos: el considerado principal, el cual suministra líneas de diálogo en base a un guion y es capaz de ramificar la conversación a través del control de opciones y uso de variables; y un sistema al que denominaremos secundario que solo debe ofrecer líneas de texto en base a un tema concreto y variable según el tiempo. La principal diferencia entre ambos es la fuente del texto, ya que mientras que el sistema principal orquesta y acciona un diálogo predefinido (estático), el sistema secundario tiene el objetivo de ofrecer contenido cambiante (dinámico), el cual ofrece a demanda del principal.

Visto de esta forma las tres herramientas que se han tenido en cuenta se pueden considerar sistemas principales, en cambio, las subsecuentes conexiones con fuentes de información dinámica deberían considerarse sistemas secundarios. Teniendo esto en cuenta se reflejaron estos cambios en el diseño de clases como se puede comprobar en la Figura 51, partiendo de *DialogueSystem* y reorganizando sus métodos para crear:

- Un *MainDialogueSystem* con las funcionalidades de consulta de variables e inicio de diálogo especificando un *DialogueActer*.
- Y un *SecondaryDialogueSystem* con la funcionalidad de iniciar el diálogo.

Existiría en ambos, a través de la herencia de *DialogueSystem*, los métodos para cambiar el lenguaje, parar el sistema, consultar su estado y para solicitar la siguiente línea de diálogo. Adicionalmente, se agregó un método *Switch()*, con el objetivo de comunicar su activación y preparar el sistema interno para recibir solicitudes.

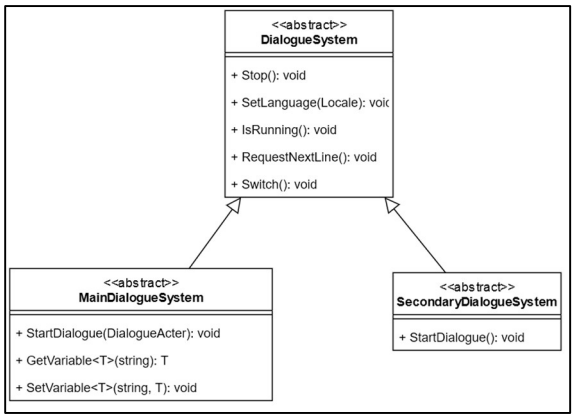

Figura 51. Diagrama de clases UML de DialogueSystem y las subclases MainDialogueSystem y SecondaryDialogueSystem. Elaboración propia.

Con el objetivo de conectar ambas clases, se introdujo en *DialogueManager* una propiedad de tipo *DialogueSystem* para denotar el sistema activo y una propiedad de tipo *MainDialogueSystem*, para requerir un sistema principal con el que empezar los diálogos. Dado que el sistema de diálogos es la pieza clave para el funcionamiento del resto de subcomponentes, es imprescindible el paso de información del sistema a *DialogueManager*. Para ello se crearon dos métodos: uno para comunicar el nombre del personaje y otra para transmitir la línea de diálogo. Adicionalmente, se crearon métodos similares para comunicar el inicio y el fin del diálogo. Cada vez que una línea de diálogo es extraída por el sistema de diálogos, se comunica a *DialogueManager* esta información. De este modo, el resto de los subcomponentes que se vayan añadiendo a *DialogueManager* serán informados de estos cambios de estado y podrán actuar en base a ellos.

Con el objetivo de permitir el cambio de sistemas, se introducen en *DialogueManager* dos métodos: *SwitchToMain()* para indicar que se requiere cambiar al sistema principal y *SwitchToSecondary(string)* para señalar que se desea cambiar a un sistema secundario, identificado por el parámetro. En la implementación concreta se deberá especificar una estructura de datos donde guardar los sistemas secundarios para poder cambiar al adecuado.

En tercer lugar, dado que *DialogueSystem* solo se encarga de proporcionar las líneas de diálogo, se necesita una clase independiente de esta para mostrar dicho texto en pantalla: *TextManager*. Siguiendo el mismo patrón, *TextManager* se trata de una clase abstracta que define el funcionamiento para mostrar el texto en pantalla. Para ello se incluyeron métodos para comunicar el comienzo de una nueva línea, el nombre del personaje y la línea de texto, la cual puede ser recibida de dos formas: por un lado, la línea de texto completa, método que se llamaría una sola vez en caso de querer mostrarla toda; y, por otro lado, solo una letra o carácter, de modo que debería llamarse al método varias veces, pasando un nuevo carácter cada vez tal que se actualice el componente visual. Con el objetivo de introducir el subcomponente se incluyó en *DialogueManager* una propiedad de tipo *TextManager*, de modo que la estructura general de las clases introducidas hasta el momento quedaría tal como se describe en la Figura 52, habiendo reorganizado y abstraído la funcionalidad presente del primer MVP.

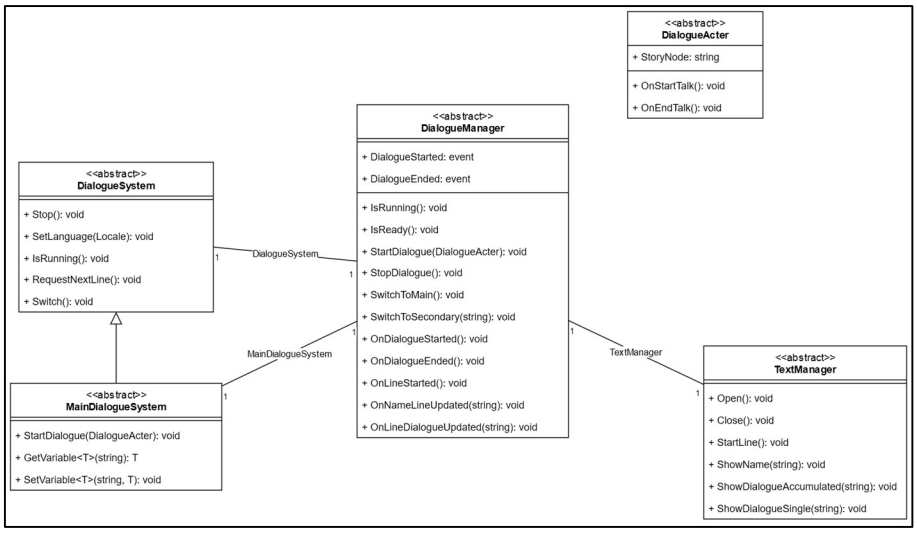

Figura 52. Diagrama de clases UML representando la estructura abstracta necesaria para implementar la funcionalidad del primer MVP. Elaboración propia.

Para finalizar la reestructuración solo se necesitaría implementar cada una de las clases concretas tal que cumplen actúen de la misma forma que su equivalente del primer MVP. Empezando con *MainDialogueSystem*, ahora sería necesario incluir una clase derivada de esta, *YarnDialogueSystem*, para encapsular el funcionamiento de Yarn Spinner. Cabe destacar en esta clase la implementación del método *StartDialogue(DialogueActer)*, el cual se limita a llamar al método *DialogueRunner.StartDialogue(string)* adjuntando como parámetro la propiedad *StoryNode* de *DialogueActer*; y de *SetLanguage(Locale)*, que establece la propiedad de *DialogueRunner.textLanguage* a la proporcionada. Para la comunicación de las líneas de texto, Yarn Spinner depende de la implementación concreta de *DialogueUIBehaviour*. En nuestro caso, ahora la clase concreta se llamaría *YarnDialogueController*, la cual se limita a implementar los métodos de *DialogueUIBehaviour* para: transmitir los eventos de inicio y fin de diálogo a *DialogueManager*; y recibir las líneas de texto para transmitirlas también a *DialogueManager*. La estructura de estas clases se puede comprobar en la Figura 53.

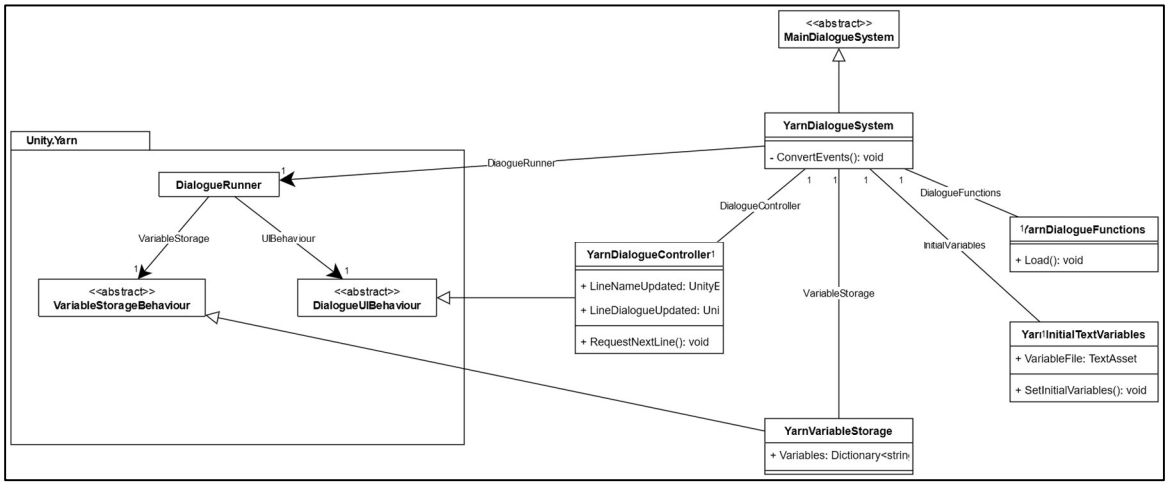

Figura 53. Diagrama de clases UML para relacionar YarnDialogueSystem con Yarn Spinner. Elaboración propia.

La visualización del diálogo en la implementación del primer MVP ya se realizaba dentro de una caja de texto. Sin embargo, esta funcionalidad estaba presente dentro de la clase *DialogueUIYarn*, por tanto, se debía extraer de esta clase para introducirla en una nueva clase concreta de *TextManager*, la cual se nombró *BoxTextManager*. En esta clase se introdujeron las referencias a

los componentes de texto (para el nombre y para el diálogo) usados hasta el momento. Los métodos de la clase abstracta se implementaron de la siguiente forma: *StartLine()* borra el contenido del componente de diálogo para prepararlo para la siguiente línea de texto; *ShowName(string)* sobrescribe el texto del componente de nombre; *ShowDialogueAccumulated(string)* actúa de forma similar sobrescribiendo el texto del componente de diálogo; y, finalmente, *ShowDialogueSingle(string)* agrega el texto proporcionado al existente en el componente de diálogo.

Para completar la reestructuración, cabe destacar el rediseño de las clases relacionadas con *IDialogueText*. En este aspecto se agregó una clase *DialogueTextAnalyser* y se movió aquí el método *AnalyseText(string)* para simplificar el punto de entrada a la clasificación de texto. El método *Parse()* en *IDialogueText* tiene como objetivo devolver una lista de *string* de forma acumulada, sin embargo, sería conveniente tener un método que devolviera una lista donde cada elemento representa el siguiente carácter, tal que "Hola" se devuelve como ["H", "o", "l", "a"], siguiendo en ambos casos las reglas de encapsulación de etiquetas. Para ello, se renombró el método *Parse()* actual a *ParseAccumulated()* y se introdujo un método *ParseSingle()*. Del mismo modo, el método *ToString()* devuelve el texto tal y como se recibió al analizarlo, pero se encontró que si se introducían elementos que no soportaran las etiquetas sería necesario obtener el texto sin estas. Por esta razón, en *IDialogueText* se renombró el método *ToString()* a *ToStringFull()* y se agregó un método *ToStringClean()* para devolver el texto sin etiquetas.

Finalmente, llega el momento de la pieza clave en esta reestructuración: *DialogueManager*. Siguiendo la nomenclatura adoptada en otros sistemas de juego donde existía una estructura similar, se optó por nombrar a la clase concreta *NormalDialogueManager*. Esta clase debía relacionar los métodos de los subsistemas mencionados (hasta el momento *MainDialogueSystem* y *TextManager*). Es de destacar el método *StartDialogue(DialogueActer)*, el cual permite al resto de sistemas externos de juego empezar un diálogo y se limitó a llamar al método homólogo del *DialogueSystem* activo. En la Figura 54 se pueden comprobar las clases concretas desarrolladas hasta el momento.

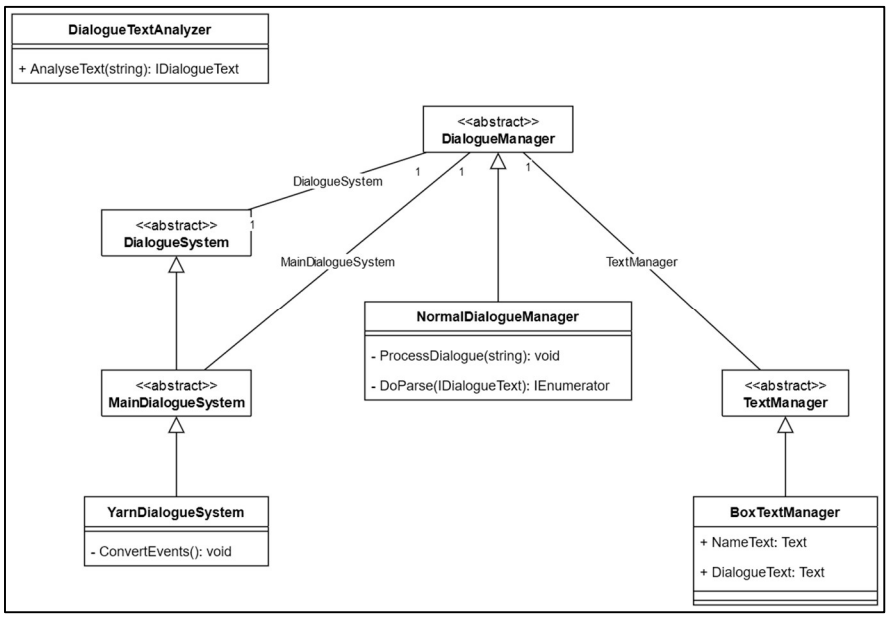

Figura 54. Diagrama de clases UML representando las clases abstractas y concretas necesarias para implementar la funcionalidad del primer MVP. Elaboración propia.

Para la comunicación del diálogo y el flujo de texto, ya se ha comentado que las implementaciones de *DialogueSystem* hacen uso de los métodos creados en *DialogueManager*, de modo que en *NormalDialogueManager* se especificó como actuar ante la recepción de esta información:

- *OnDialogueStarted()* y *OnDialogueEnded()* llaman a los métodos de *TextManager.Open()* y *TextManager.Close()*, respectivamente. También es en estos métodos donde se invocan eventos de inicio y fin de diálogo con el objetivo de que sistemas externos puedan reaccionar en base a ellos (p. ej.: para cerrar menús o impedir el movimiento del personaje).
- *OnLineStarted()* invoca el método *TextManager.StartLine()* para preparar la caja de texto para la línea actual.
- *OnLineNameUpdated(string)* se limita a llamar a *TextManager.ShowName(string)* adjuntando el mismo parámetro recibido.
- Por último, *OnDialogueLineUpdated(string)* llama al método estático *AnalyseText* de *DialogueTextAnalyser* para obtener el texto clasificado. De este modo se puede extraer una lista de caracteres a través de su método *ParseSingle()*, la cual se puede iterar para transmitir un nuevo carácter (junto con sus etiquetas de texto si las tuviera) a través de *TextManager.ShowDialogueSingle(string)*.

En la Figura 55 se puede comprobar el flujo de texto desde el sistema de diálogos, pasando por *NormalDialogueManager*, para finalmente presentar el texto en pantalla. Este nuevo diseño permite la intercambiabilidad de *TextManager* sin afectar la implementación de *NormalDialogueManager*, así como la fácil inclusión de un posible *VoiceManager*, para incluir soporte de voz. Una vez empezado el diálogo también es *NormalDialogueManager* el encargado de detectar la pulsación de teclas para avanzar el texto, en cuyo caso se avanzaría en suministrar letras a *TextManager* o, en caso de ya haber terminado la línea, llamaría al método *DialogueSystem.RequestNextLine()* para solicitar al sistema la siguiente línea.

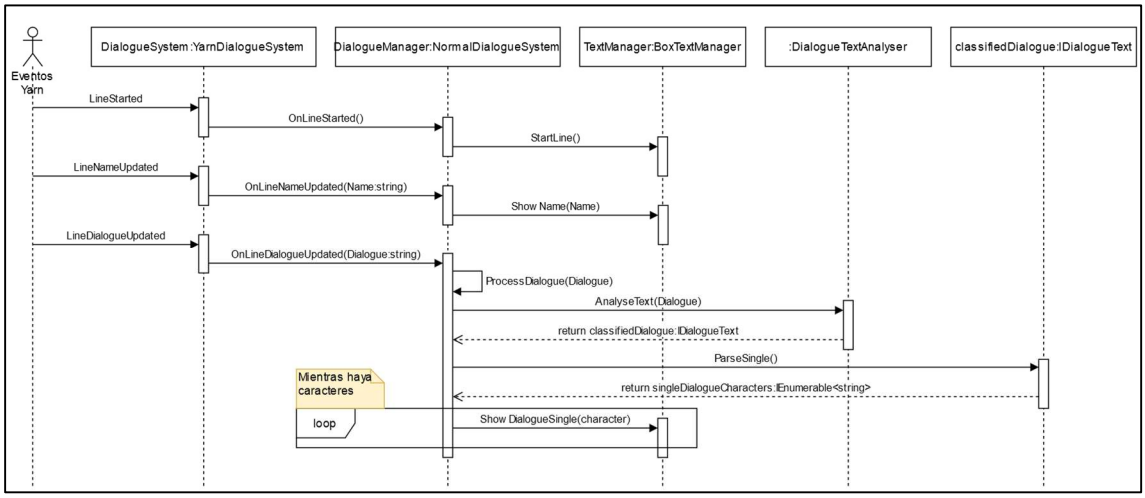

Figura 55. Diagrama de secuencia para representar el flujo de texto desde la invocación de los eventos de Yarn Spinner hasta su presentación en pantalla. Elaboración propia.

4.4. Alternativas de Voz

Similarmente a la problemática con el sistema de diálogo durante el inicio del desarrollo, al inicio del segundo MVP se decidió incluir voz a los personajes para pronunciar sus líneas de diálogo. Dada la naturaleza de los personajes del videojuego y su estética de dibujo animado, se optó por un sistema en el que los personajes hablaran de forma distinta a cualquier idioma existente, similar sobre todo a los ejemplos de voz sintética del «animalese» de *Animal Crossing*. En esta serie de

juegos, al igual que en nuestro proyecto, se da voz a personajes no humanos con un sistema simple pero efectivo: dada una línea de diálogo, se pronuncia cada fonema de forma estricta, con diferente tono y entonación para cada personaje. A continuación, se describen las alternativas existentes en el mercado con posibilidad de conseguir este efecto, tanto servicios web como herramientas independientes.

Cabe remarcar que, durante la búsqueda de estas opciones también se encontraron soluciones que, pese a no poder ser integradas al no tratarse de herramientas independientes o ser creadas en lenguajes de programación incompatibles, demostraron la sencillez de las implementaciones dedicadas directamente a crear el efecto deseado. Son el caso de: RPG Dialogue Generator[93], un proyecto personal creado por Zack Bogucki con el objetivo de demostrar la simplicidad de crear un sistema de síntesis de voz para Unity basado en sonidos y fonemas; y Animalese Audio Generator[94], un proyecto de código abierto y de simple estructura desarrollado con Python para simular el efecto de voz «animalese».

Servicios *text-to-speech* en la nube

En la actualidad existen servicios en la nube, también conocidos como software como servicio o SaaS[95] por su siglas en inglés, que ofrecen una utilidad concreta a través de Internet. Dentro de los servicios ofrecidos existen aquellos relacionados con la síntesis de voz, donde se incluyen funcionalidades para conseguir voz sintética a partir de texto. Estos servicios son ofrecidos bajo un servicio de suscripción, en base a la cantidad de audio generado o número de caracteres convertidos a audio. En este ámbito hay soluciones ofrecidas por empresas populares dentro de la computación, que son capaces de usar redes neuronales con el objetivo de generar voces más naturales. Este es el caso de Microsoft Azure Text to Speech[96], Amazon Polly[97], Google Cloud Text-to-Speech[98] o IBM Watson[99]. Según opiniones de sus usuarios[100], la calidad del servicio ofrecido por estas empresas es similar, de modo que la diferencia competitiva reside en aspectos concretos: la alternativa de Google ofrece la mejor disponibilidad y tiempo de respuesta frente a sus competidores; Amazon Polly ofrece como baza principal un bajo precio de introducción y un buen servicio al cliente; el servicio de Microsoft tiene una documentación más completa y fácil de entender. Debido a su similitud, finalmente solo se tuvo en cuenta el servicio de Microsoft, ya que dispone de kits de desarrollo y tutoriales específicos para Unity.

Asset Store

La Asset Store de Unity es una tienda donde se pueden encontrar y obtener herramientas para facilitar el desarrollo mediante Unity en diferentes ámbitos, incluyendo el de audio y voz. A través de esta tienda se encontraron dos opciones a destacar: por una parte, RT Voice Pro[101], una solución de síntesis de texto de forma local que ofrece opciones de tono, género y velocidad para generar habla natural; y, por otra parte, Klattersynth TTS[102], una herramienta para producir voz a partir

[93] Disponible en <http://www.zbogucki.com/portfolio/dialogue-generator>
[94] Disponible en <https://github.com/equalo-official/animalese-generator>
[95] Más información en <https://es.wikipedia.org/w/index.php?title=Software_como_servicio&oldid=127035992>
[96] Disponible en <https://azure.microsoft.com/en-us/services/cognitive-services/text-to-speech/>
[97] Disponible en <https://aws.amazon.com/es/polly/>
[98] Disponible en <https://cloud.google.com/text-to-speech>
[99] Disponible en <https://www.ibm.com/cloud/watson-text-to-speech>
[100] Extraído de <https://www.g2.com/compare/amazon-polly-vs-azure-text-to-speech-api-vs-google-cloud-text-to-speech-vs-ibm-watson-text-to-speech>
[101] Disponible en <https://assetstore.unity.com/packages/tools/audio/rt-voice-pro-41068>
[102] Disponible en <https://assetstore.unity.com/packages/tools/audio/klattersynth-tts-95453>

de texto tal que deforma el habla natural sintetizado para generar un resultado de voz intencionadamente mecánico.

Conclusiones

En la Tabla 6 se pueden discernir las características descritas de los sistemas de voz propuestos con el objetivo de comparar sus cualidades y simplificar su elección.

	Azure TTS	RT Voice Pro	Klattersynth TTS
Conexión a internet	Necesaria, servicio en línea.	No	No
Integración en Unity	A través de conexión con API. Kit de desarrollo disponible para Unity.	Paquete autosuficiente y creado para Unity.	
Opciones de configuración	Voz (*persona*), tono, género, pronunciación, pausas, entonación.	Tono, género, velocidad.	Personalización de fonemas usados.
	Configuración existente de componente de audio de Unity.		
Coste	Suscripción basada en horas de audio generado y número de usos concurrentes.	Pago único de 69,68 €.	Pago único de 21,44 €.

Tabla 6. Tabla comparativa de sistemas de voz. Elaboración propia.

Las alternativas consideradas cuentan, en general, de opciones de configuración suficientes para crear el efecto «animalese» deseado. La opción de Microsoft ofrece opciones de género, velocidad y tono, sin embargo, la necesidad de mantener una conexión a internet para funcionar dificulta su elección, ya que, al tratarse de un videojuego para un jugador, se podrían suceder experiencias distintas según la conexión del usuario final. Por otro lado, la herramienta RT Voice Pro cuenta, en comparación, con un menor número de opciones de configuración, pero se trata de un paquete local y autosuficiente. Ambas opciones podrían beneficiarse de las opciones adicionales ofrecidas por el componente de audio (*AudioSource*) de Unity para crear el efecto deseado. Klattersynth TTS ofrece, sin embargo, una propuesta distinta, similar a la que se necesita para crear el efecto «animalese» y con un coste menor, donde se pueden personalizar los fonemas utilizados para conseguir un mejor resultado.

Pese a ello, la aproximación de estas herramientas usa la síntesis de voz para generar un habla humana natural, un efecto preferible para otros juegos o para aplicaciones donde se necesita de la accesibilidad que esta funcionalidad ofrece, exceptuando la alternativa de Klattersynth TTS. De modo que, tras deliberar las opciones y tener en cuenta estas consideraciones, se determinó que la funcionalidad deseada se podía implementar por el propio equipo de desarrollo para obtener un mayor control de la solución y de la voz, tomando como inspiración los proyectos independientes anteriormente mencionados (RPG Dialogue Generator y Animalese Audio Generator). Por otro lado, dado que la popularidad de los servicios en la nube es innegable y, en concreto, de la síntesis de voz natural, también se propuso la implementación de la alternativa de Azure para introducir este servicio en el juego, ya que existe un nivel gratuito en el caso de mantener una sola conexión concurrente y no sobrepasar los 5 millones de caracteres.

4.5. Implementación de voz

Previamente a implementar los aspectos que permitirían a los personajes hablar en un idioma similar a «animalese», cabía introducir los mecanismos para permitir la funcionalidad de voz dentro de la estructura presentada en el capítulo 4.2.4. Para ello se siguió una estructura similar a la utilizada con *TextManager*, creando una clase abstracta llamada *VoiceManager*, de modo que el texto recibido por el sistema sería enviado tanto a *TextManager* como a *VoiceManager*. Esta clase incluye métodos para abrir y cerrar el sistema, así como para comunicar el texto actual para ser pronunciado. Similarmente se agregó una propiedad de tipo *VoiceManager* a *DialogueManager* con tal de poder comunicarle la información y el estado del diálogo a través de los métodos descritos, tal como se puede apreciar en la Figura 56. Finalmente, el flujo de texto realizado en *NormalDialogueManager* ahora también incluiría los aspectos de *VoiceManager*: cuando se inicia una línea (*OnStartLine*) sería llamado su método *StartLine();* y la comunicación del texto de diálogo (*OnLineDialogueUpdated*) se realizaría a través del método *SpeakAccumulated(string)* adjuntando el texto sin etiquetas (a través de *IDialogueText.ToStringClean()*), ya que estas no deberían ser pronunciadas.

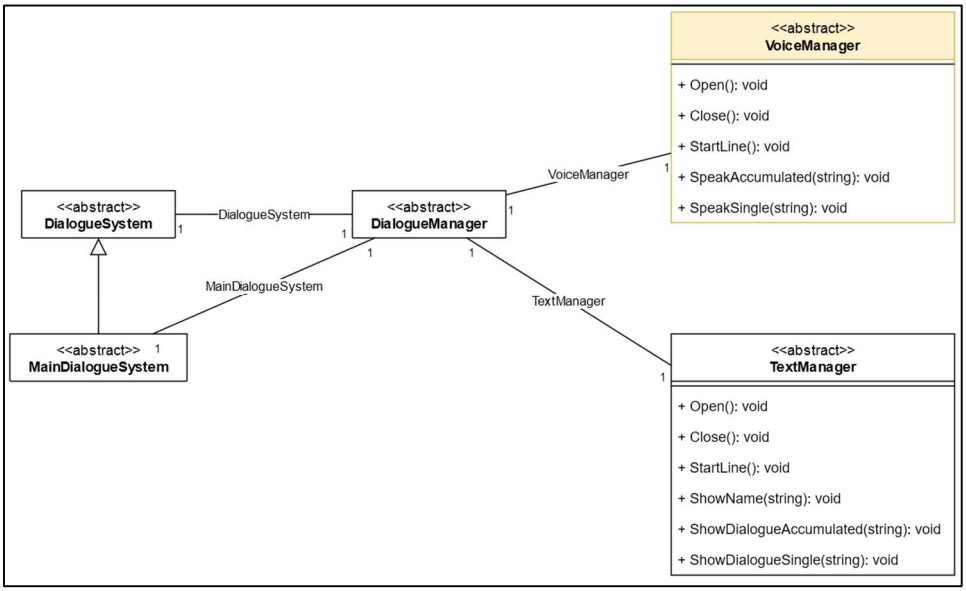

Figura 56. Diagrama de clases UML para representar la adición de VoiceManager (resaltado en amarillo) a la estructura. Elaboración propia.

Implementación propia

Para la implementación del estilo de voz de «animalese» se creó una clase concreta de *VoiceManager* llamada *AnimaleseVoiceManager*. Una vez *DialogueManager* proporcionara el texto se efectuaría un análisis para dividir el texto en frases, entendido como aquellas que terminan en un signo de puntuación, de modo que se obtiene una lista de frases independientes que facilitará la pausa entre ellas. Con esta lista se procede a iterar sobre ella para encontrar y pronunciar cada fonema de la frase actual: se extrae cada carácter y se analiza qué fonema le corresponde, en el caso de haber confusión con un dígrafo también se extrae el siguiente carácter para determinar el fonema exacto (ej.: en inglés «c» puede cambiar su fonema si va seguido de «h» o de «k»); después de cada fonema, se efectúa un retardo de tiempo para simular una pausa en la conversación. Este proceso se repite hasta que no quedan más caracteres por pronunciar, momento en el que se realiza un retardo de medio segundo y se empieza el proceso con la siguiente frase. Dado que el conjunto de dígrafos varía con cada idioma, y uno de los puntos fuertes de este estilo de voz es la no correspondencia con ningún idioma, se debía elegir previamente un idioma

sobre el que basar la comprobación de dígrafos. Finalmente se optó por el idioma inglés debido a la simplicidad de este. Cabe destacar que esta decisión se tomó puramente por conveniencia y no afecta la experiencia final del jugador, ya que se seguirá percibiendo un sonido sin sentido basado en el texto y con una vaga correspondencia a cualquier idioma real. Los archivos de audio para los fonemas se tomaron del proyecto de código abierto Animalese Audio Generator, donde se proporciona cada fonema en el formato de sonido *Waveform audio* (.wav). En la Figura 57 se puede observar estos sonidos importados dentro del inspector de Unity junto con su asociación en *AnimaleseVoiceManager*.

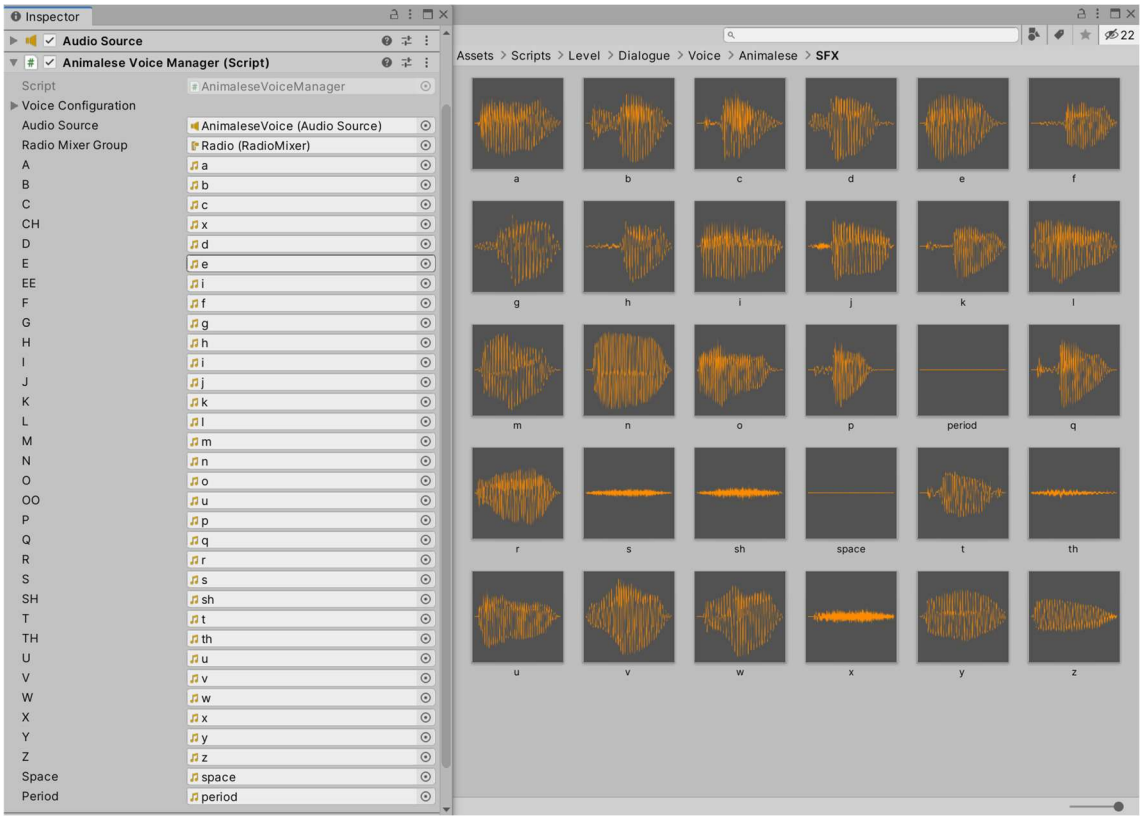

Figura 57. Captura de pantalla de Unity donde se encuentra el inspector seleccionando el script AnimaleseVoiceManager en la escena (izquierda) y los archivos de audio vistos desde Unity (derecha). Elaboración propia.

Azure Text To Speech

Dado que se trata de un software como servicio sería necesario cumplimentar una serie de pasos con el objetivo de obtener el permiso para usar la herramienta en nuestro proyecto. En primer lugar, se debió crear una cuenta de Microsoft Azure, la cual se creó a través de una cuenta existente de Microsoft. Una vez activada esta cuenta, se procedió a la creación de un recurso, visto como un contenedor donde se agrupan servicios relacionados con un mismo proyecto. En nuestro caso se creó un nuevo recurso llamado «Frozen-Out» y, dentro de él, se agregó el servicio «Servicios cognitivos», el cual contiene las funcionalidades de voz a texto (Speech to Text), texto a voz (Text to Speech), traducción de voz (Speech Translation) y reconocimiento de voz (Speech Recognition). Tal como se describe en su documentación[103], hay dos parámetros necesarios para identificarse en el servicio y acceder a sus funcionalidades: la región del servicio, un valor de tipo *string* que representa la región del mundo desde la que se tiene permiso para acceder al servicio, en nuestro caso el oeste de Europa («westeurope»); y la clave de suscripción, una cadena de

[103] Disponible en <https://docs.microsoft.com/es-es/azure/cognitive-services/speech-service/get-started-text-to-speech?tabs=script%2Cwindowsinstall&pivots=programming-language-csharp>

caracteres alfanumérica única para autenticar la conexión. En la Figura 58 se pueden ver estos conceptos en el portal de Azure.

Las utilidades de este servicio se pueden acceder a través de llamadas a su API REST[104], proporcionando los parámetros mencionados para autenticarse con el servicio. Sin embargo, desde Microsoft proveen de un kit de desarrollo, llamado Speech SDK[105], con el objetivo de abstraer las llamadas REST y facilitar el acceso. Este kit se encuentra disponible para Unity, de modo que se hizo uso de él para delegar la autenticación y administración de tokens de seguridad, y centrarse en obtener un resultado convincente de voz a través de este.

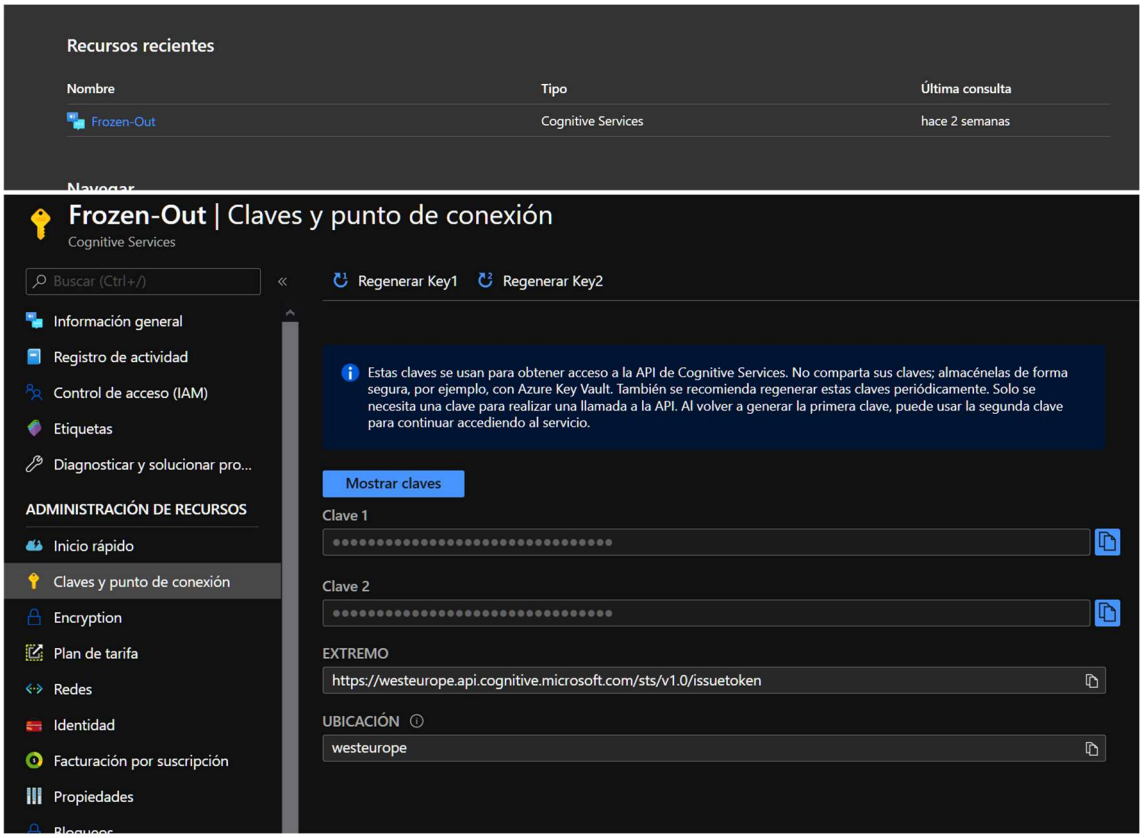

Figura 58. Captura de pantalla del portal de Azure listando los recursos creados (arriba) y las claves de conexión para los servicios cognitivos agregados (abajo). Elaboración propia.

Una vez integrado el kit de desarrollo se podía proceder a la implementación de una clase derivada de *VoiceManager*, llamada *AzureTTSManager*, con tal de poder recibir las líneas de diálogo y actuar en base a ellas. La primera vez que se instancia esta clase se establece la configuración de la conexión: estableciendo la región y la clave de suscripción. Además, tal y como se recomienda en los tutoriales [21] y clases de ejemplo para Unity [22], se configura el formato de respuesta al tipo de audio monoaural de 16 bits y 16 KHz (tipo interno «Raw16Khz16BitMonoPcm»). Tomando estas consideraciones, cuando se recibe el texto del diálogo se crea un *SpeechSintesizer* con la configuración inicial y se procede a llamar al servicio con este texto. Los datos de audio recibidos no se pueden reproducir directamente en Unity y deben ser convertidos para poder ser reproducidos a través de un componente de *AudioSource*. Esta conversión está basada en la presente en las clases de ejemplo.

[104] Disponible en <https://docs.microsoft.com/es-es/azure/cognitive-services/speech-service/rest-text-to-speech>
[105] Disponible en <https://docs.microsoft.com/es-es/azure/cognitive-services/speech-service/speech-sdk>

Cabe destacar que, al tratarse de un servicio externo y dependiente de la conexión de la red, existe un lapso entre la llamada y la recepción del audio, de modo que tampoco se postula como una herramienta de confianza para un videojuego, donde se requiere que todos los jugadores perciban la misma experiencia, independientemente de su conexión a la red. Además, en caso de jugar sin conexión, se debería tener en cuenta un plan de contingencia y proporcionar una herramienta de voz alternativa para reemplazarlo, otro aspecto que segregaría la base de jugadores y crearía experiencias distintas. A continuación, en la Figura 59, se pueden apreciar las clases concretas de *VoiceManager*, donde se encuentran tanto *AnimaleseVoiceManager* como la recientemente creada *AzureTTSManager*.

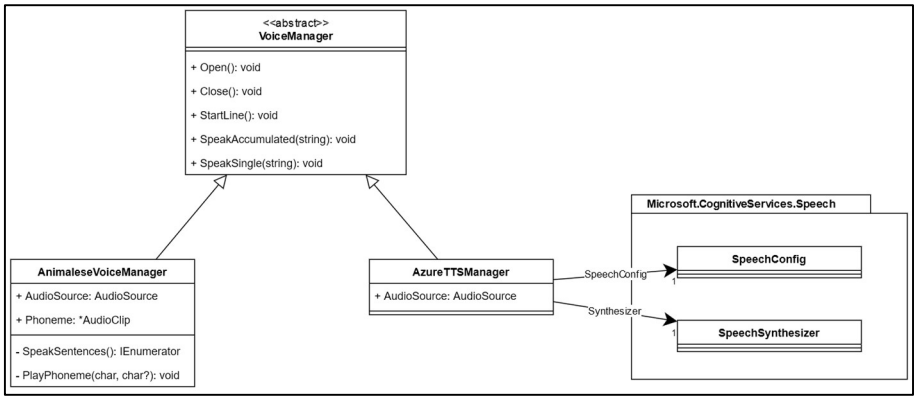

Figura 59. Diagrama de clases UML para representar las clases concretas de VoiceManager. Elaboración propia.

4.6. Estilo de diálogo

Con la estrategia de presentación de texto y voz actual, todos los personajes del juego van a poseer el mismo estilo de texto y voz, disminuyendo así el atractivo estético de encontrar nuevo diálogo. Con el objetivo de ofrecer un fácil control de los estilos para cada personaje y unificar las diferentes opciones de texto y voz, se diseñaron dos nuevas clases *TextStyle* y *VoiceStyle*. Dentro de ellas se introducirían las opciones disponibles de tal modo que se puedan modificar para configurar cada personaje. Estas clases estarían finalmente agregadas dentro de una clase *DialogueStyle* para ofrecerlas desde un mismo punto de control, así como para extraer funcionalidad común y conectar opciones de estilo relacionadas.

Previamente a la descripción de las opciones de cada estilo, cabe destacar cómo se incluyen dentro del diseño introducido, ya que su objetivo es ofrecer un sistema donde se pueden agregar implementaciones distintas, para ofrecer distinta funcionalidad. Por esta razón, no se puede presuponer que cada herramienta use los mismos valores de estilos y puede ser conveniente la calibración de estos valores para obtener un mismo resultado, es por ello que se decidió que el control del desarrollador debería ser sobre un valor relativo, dentro de un rango de -1 a 1, para posteriormente, en base a la herramienta de texto y voz usada, convertir estos valores a valores entendibles y usables. De este modo cada herramienta debe presentar sus valores mínimos, máximos y por defecto de cada opción, con el objetivo de convertir el valor relativo recibido en un valor interpolado y entendible por la implementación concreta, tal como se puede comprobar en la Figura 60. Para almacenar estos tres valores se creó la clase *StyleConfiguration*, la cual presenta tres propiedades para representar el valor por defecto, mínimo y máximo. Para agrupar las configuraciones de texto y de voz se creó *TextStyleConfiguration* y *VoiceStyleConfiguration*, respectivamente, para contener propiedades de tipo *StyleConfiguration*. Para denotar la configuración de cada herramienta debe presentar se introdujo en *TextManager* una variable de *TextStyleConfiguration*, y del mismo modo con *VoiceManager* y *VoiceStyleConfiguration*.

```
public class DialogueStyle
{
    [...]
    0 references
    private static float InterpolateRelative(float relativeValue, float defaultValue, float minValue, float maxValue)
    {
        float finalValue = defaultValue;

        if(relativeValue < 0)
        {
            finalValue = Mathf.Lerp(minValue, defaultValue, (1 + relativeValue));
        }
        else if(relativeValue > 0)
        {
            finalValue = Mathf.Lerp(defaultValue, maxValue, relativeValue);
        }

        return finalValue;
    }
    [...]
}
```

Figura 60. Fragmento de código encargado de obtener el valor interpolado dado un valor relativo, un valor por defecto, un valor mínimo y un valor máximo. Elaboración propia.

Las opciones de estilo también deberían ser transmitidas a *TextManager* y *VoiceManager* durante el flujo de información desde *DialogueManager*. Para ello se introdujo un método *OnStyleNameUpdated(string)*, para poder comunicar desde *DialogueStyle* a *DialogueManager* el estilo a aplicar en la línea de diálogo actual. Paralelamente, se introdujo en *TextManager* un método *SetStyle(TextStyle)* y, similarmente, en *VoiceManager* un método *SetStyle(VoiceStyle)*, para comunicar el estilo a aplicar en la línea de diálogo actual. De este modo, el sistema de diálogos proveerá el nombre del estilo (podría ser distinto al nombre del personaje) a través de *OnStyleNameUpdated(string)*, este parámetro deberá ser usado para obtener el objeto de estilo (*DialogueStyle*) asociado para poder finalmente transmitir el estilo de texto a *TextManager* y el estilo de voz a *VoiceManager*.

Cabe destacar que tener un método independiente para comunicar el nombre del personaje y el nombre del estilo permite guardar estilos basados en un estado de ánimo o emoción. Estos cambios en la estructura se pueden consultar en la Figura 61. Esta estrategia introduce el problema de almacenar el nombre del estilo en la línea de texto. Hasta el momento se imponía un formato «Nombre: Diálogo» para cada línea, de modo que se amplió para añadir el nombre del estilo: «Nombre++Estilo: Diálogo». Con este nuevo formato se eligió el separador «++», una sucesión de caracteres poco común de aparecer en el diálogo. La separación de componentes de la línea de texto ahora extraería el diálogo, el estilo y el nombre, exceptuando en ocasiones en el que no se especifique el estilo, caso en el que se establece el estilo igual que el nombre (entendido como que el estilo por defecto de un personaje tiene su mismo nombre).

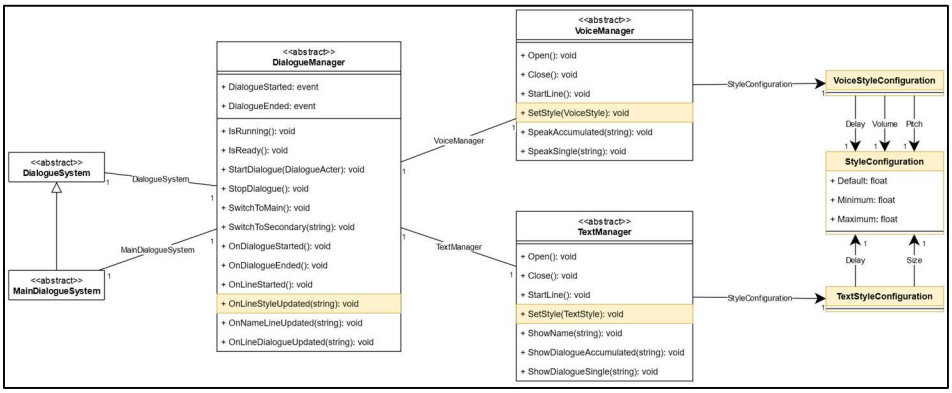

Figura 61. Diagrama de clases UML representando la estructura tras la adición del soporte para estilo (resaltando en amarillo los cambios). Elaboración propia.

4.6.1. *TextStyle*

Para el texto se decidió incluir la configuración de la fuente, el tamaño y el color, así como agregar efectos de salto (breve salto de las letras), de fundido (tenue fundido de las letras), de contorno de letras y de irregularidad (convertir parte del texto en otros símbolos para imitar una comunicación entrecortada). De este modo, se incluyó en *TextStyle* dos propiedades de tipo *Font* y *Color*, provenientes de Unity, para representar la fuente y el color, respectivamente. Por otro lado, el tamaño del texto se puede establecer según un valor relativo, dejando en manos de la implementación de *TextManager* decidir los rangos de tamaño permitidos. Es por ello que se agregó una propiedad para representar el valor relativo y otra propiedad para el tamaño final interpolado. En la clase *TextStyleConfiguration* creada anteriormente se incluyó una variable de tipo *StyleConfiguration* para representar el rango permitido del tamaño. Los efectos mencionados se representaron por un *enumerator* y se agregó una propiedad de ese tipo en *TextStyle*: *Jumping* para el efecto de salto, *Fading* para el efecto de fundido, *Highlighted* para el efecto del contorno, *Interrupted* para el efecto de texto interrumpido y, finalmente, un elemento adicional llamado *None* para marcar que no se desea aplicar ningún efecto.

La aproximación inicial para implementar estos estilos desde *BoxTextManager* fue aprovechar las capacidades existentes en los elementos de texto de Unity. Estos elementos permiten cambiar la fuente, el color y el tamaño de la fuente, así como la alineación del texto y el estilo de la fuente, entre otras opciones. De esta forma, una vez recibido el estilo se pueden reemplazar los valores de la caja de diálogo con aquellos provenientes del estilo. Además, durante el primer MVP el único estilo disponible era aquel que se aplicaba al guion. Recuperando la estrategia del lenguaje de marcado RichText (RTML), existirían ahora dos formas de afectar al estilo del texto, a través de la configuración de estilo de *TextStyle* o a través del formato RichText en el guion. Para evitar confusiones, se decidió limitar el uso de RichText para palabras o frases concretas de un personaje (ej.: para destacar una palabra o dirigir la atención a una parte concreta del texto) y delegar a *TextStyle* la forma de hablar usual de cada personaje. En cuanto a la configuración específica de *BoxTextManager*, visible en la Figura 62, se estableció un tamaño por defecto de texto de 30, el cual resultó ser un tamaño con buena visibilidad en las resoluciones en las que se iba a distribuir el juego, y un tamaño mínimo y máximo de 25 y 50.

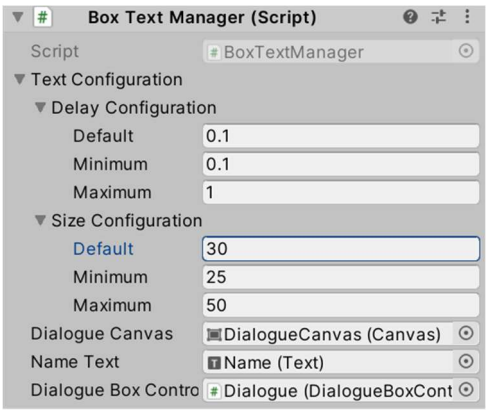

Figura 62. Captura de pantalla de Unity con el inspector seleccionando el script BoxTextManager en escena. Elaboración propia.

La idea para abordar la creación de los efectos sería crear una animación para cada efecto, tal que, según el efecto seleccionado, la animación se produjera para cada letra de forma ordenada. Sin embargo, el componente de texto no permite animar las letras de forma individual, sino el texto completo, de modo que sería necesario cambiar la forma en la que el texto es proporcionado a la caja de texto: creando un componente de texto por cada letra proporcionada. Sería necesario

también crear las líneas de forma manual y procurar que el texto se mantiene dentro de la caja de texto.

Para ello, se identificó la estructura básica de una frase y se creó un *prefab*[106] para cada uno: el párrafo, la línea, la palabra, y finalmente la letra, entendida como el componente indivisible y el cual debe animarse, y que, por tanto, contendría el animador[107]. De forma trivial, el párrafo contendría líneas de forma vertical, la línea constaría de palabras ordenadas horizontalmente, y la palabra estaría compuesta de letras de manera horizontal. Siguiendo esta estrategia, ahora el texto proporcionado, que ya se suministra de letra en letra por el sistema de diálogos, crearía uno de los diferentes *prefab* tal como se puede comprobar en la Figura 63. Para su posterior animación de las letras, se recorrería la lista de letras y, según la opción elegida en *TextStyle*, la animación tendría lugar y daría la sensación del efecto deseado.

Figura 63. Captura de pantalla de TextStyle incorporado a Frozen Out demostrando la creación de líneas, palabras y letras para presentar la línea de texto. Elaboración propia.

4.6.2. *VoiceStyle*

En cuanto a la configuración relativa a la voz, se examinaron los ejemplos existentes del juego en el que se basaban (*Animal Crossing*), de modo que se establecieron las siguientes opciones: el tono, el volumen y la velocidad, así como un efecto de radio necesario para una porción del nivel. Para tener en cuenta la configuración de cada herramienta se agregó en *VoiceStyle* una propiedad para representar el valor relativo y otra para guardar el valor absoluto, para cada una de estas opciones. Los efectos de audio se agregan como un *enumerator* donde se distingue el elemento *None*, para determinar que no se aplica ningún efecto, y el elemento *Radio*. Paralelamente se agregó una propiedad de este *enumerator* en *VoiceStyle*. La configuración necesaria para los tres valores relativos (tono, volumen y velocidad) se añadieron en *VoiceStyleConfiguration* como tres propiedades de tipo *StyleConfiguration*.

Para implementar el estilo en *AnimaleseVoiceManager* se tuvieron en cuenta las capacidades del componente de audio de Unity (*AudioSource*), donde se pueden configurar el volumen y el tono, además de ajustar la espacialidad del audio. De modo que para integrar las opciones en *AnimaleseVoiceManager*, se establecen los valores interpolados recibidos de *VoiceStyle* a las propiedades correspondientes en el componente *AudioSource* asignado. Las configuraciones de

[106] Componente de Unity que almacena objetos con configuraciones y propiedades predefinidas tal que se facilita su reusabilidad. Más información en <https://docs.unity3d.com/Manual/Prefabs.html>
[107] Componente de Unity que administra y reproduce las animaciones de un objeto.

VoiceStyleConfiguration para esta clase también están basadas en los valores permitidos por el componente *AudioSource*: el tono puede tomar valores entre 1 y 3, estableciendo un valor por defecto de 2, los cuales preservan el audio agudo característico de *Animal Crossing*; el volumen toma valores entre 0 y 1, siendo el valor por defecto 0,9 para conseguir un diálogo audible pero también dejando margen para elevarlo y disminuirlo si se desean conseguir otros efectos. Por otro lado, la velocidad se representa como el retardo de tiempo en segundos entre la pronunciación de cada letra, de modo que tomó valores entre 0,1 y 1, estableciendo un valor por defecto de 0,1. El valor del retardo proveniente de *VoiceStyle* se aplica en la función encargada de analizar los caracteres y los fonemas para esperar la cantidad de tiempo requerida. En la Figura 64 se pueden apreciar los valores de la configuración en escena.

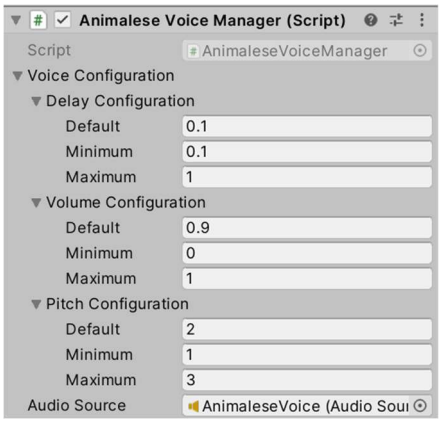

Figura 64. Captura de pantalla de Unity con la configuración de estilo de AnimaleseVoiceManager. Elaboración propia.

El efecto de radio mencionado también se puede obtener a través de componentes *AudioMixer*, los cuales modifican la salida de audio de un componente *AudioSource*. De este modo, si el estilo tiene seleccionado el efecto *Radio* se puede aplicar al *AudioSource* asignado un *AudioMixer* que produzca un efecto de audio de baja calidad y similar a una radio vieja. Para conseguir este efecto se basó en implementaciones de otros desarrolladores[108] aplicando la configuración final visible en la Figura 65.

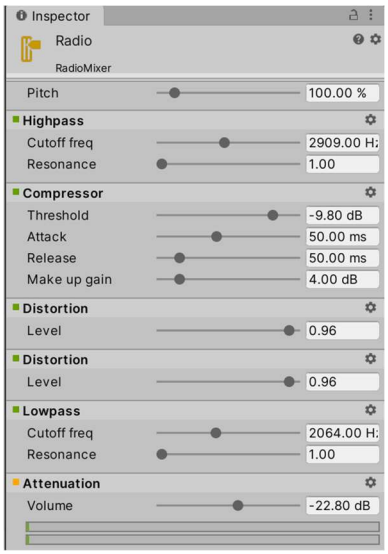

Figura 65. Captura de pantalla de Unity con el inspector seleccionando el AudioMixer con la configuración necesaria para generar el efecto de radio. Elaboración propia.

[108] Más información en <https://community.gamedev.tv/t/radio-distortion-sound-effect/15600>

4.6.3. *DialogueStyle*

Hasta el momento, las dos configuraciones de estilo se encuentran de forma independiente, sin embargo, existen opciones que tienen relación entre ellas como: el tamaño de texto y el volumen, un texto más grande debería también suponer un volumen más alto de voz, y viceversa; y la velocidad de texto y de voz, ya que las palabras que se van revelando en pantalla deberían coincidir con su pronunciación. Además de estas relaciones, la estructura de código se beneficiaría de una clase que congregara ambas clases de estilo para transmitirlas de forma conjunta. Por ello, se desarrolló una clase *DialogueStyle* con una propiedad de tipo *TextStyle* y otra de tipo *VoiceStyle*. Para introducir la relación entre tamaño y volumen se agregó en *DialogueStyle* una propiedad que representa el valor relativo de la intensidad del diálogo, la cual, aplicando la configuración de las herramientas en uso, se convertiría en el valor de tamaño y de volumen, respectivamente. Para reflejar estos cambios y relaciones se eliminó el valor relativo de tamaño en *TextStyle* y el valor relativo de volumen en *VoiceStyle*. Similarmente, para la opción de la velocidad se incluyó una propiedad que representaría el valor relativo de la velocidad, la cual afectaría al retardo de *TexStyle* y el retardo de *VoiceStyle*. Finalmente, con el objetivo de ofrecer un punto de entrada para establecer los valores interpolados se crearon dos métodos en *DialogueStyle*: *UpdateText(TextStyleConfiguration)*, el cual usa los valores de la configuración proporcionada para establecer los valores finales de *TextStyle*; y *UpdateVoice(VoiceStyleConfiguration)*, que actúa de forma similar para los valores de *VoiceStyle*. Esta estructura definitiva se puede consultar en la Figura 66.

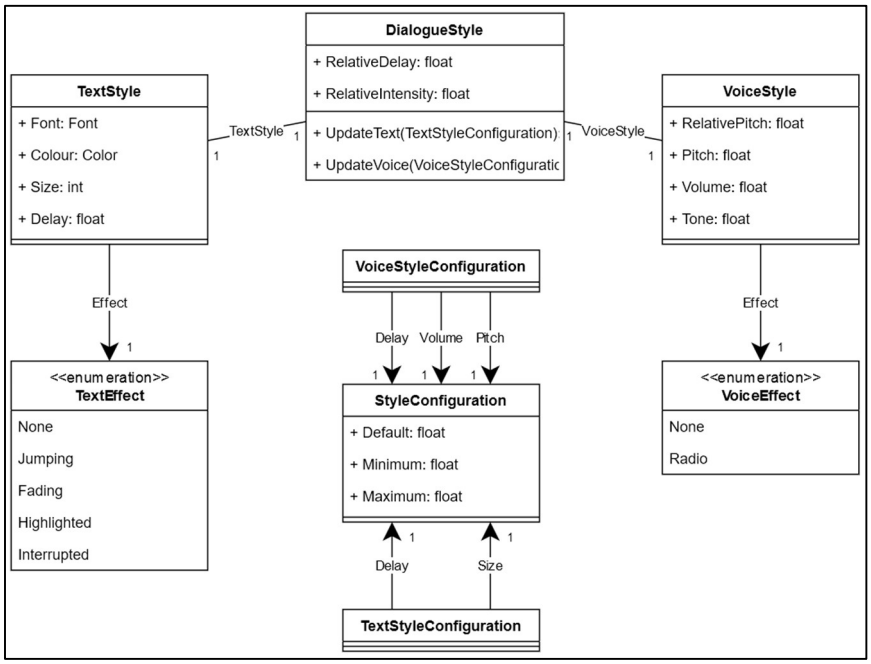

Figura 66. Diagrama de clases UML relacionado con el estilo de diálogo. Elaboración propia.

4.6.4. Guardado de estilos

El objetivo final de estas clases es tener la capacidad de establecer un estilo para cada personaje, por ello es necesario el guardado de estos estilos para su posterior recuperación. Con este objetivo se creó una clase abstracta *StyleStorage* con un único método público *GetStyle(string)*, el cual recibe el nombre de un estilo y devuelve el *DialogueStyle* asociado a este. Dentro de *DialogueManager* se incluyó una propiedad de *StylesStorage*, la cual se usaría cuando se obtuviera el nombre del estilo (*OnStyleNameUpdated*) para obtener el estilo correspondiente y comunicarlo a *TextManager* y *VoiceManager*. Para la implementación concreta se decidió

almacenar la relación entre el personaje y el estilo en un diccionario, de modo que se nombró *DictionaryStylesStorage*. Dado que en Unity los diccionarios no pueden ser modificados a través de la interfaz, se creó una clase modelo *CharacterDialogueStyle* donde se encuentra una propiedad de tipo *string*, para representar el nombre del personaje, y una segunda propiedad de tipo *DialogueStyle*. De este modo, *DictionaryStylesStorage* convertiría una lista de *CharacterDialogueStyle* en el diccionario interno. Por último, se agregó una propiedad de tipo *DialogueStyle* para denotar el estilo por defecto, el cual sería usado en los casos en los que no se especifica un estilo. Esta estructura de clases se puede comprobar en la Figura 67. Cabe destacar que, al recuperar el estilo correspondiente al personaje, se aplicará la configuración de texto y de voz de las herramientas en uso, a través de los métodos creados anteriormente *DialogueStyle.UpdateText* y *DialogueStyle.UpdateVoice*.

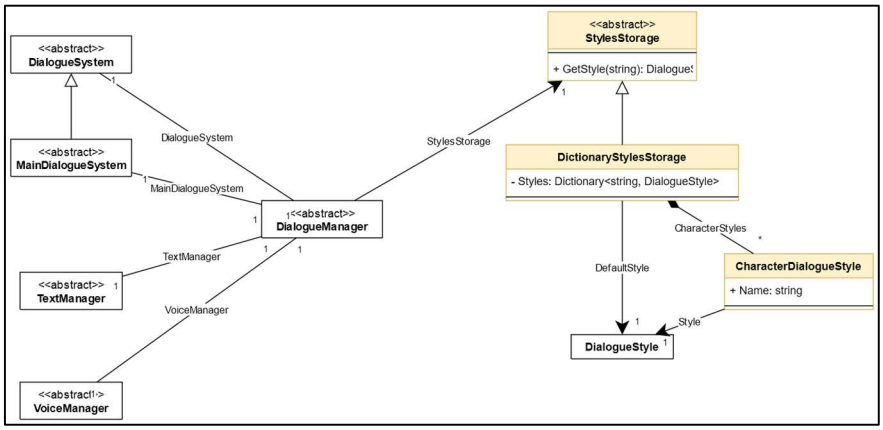

Figura 67. Diagrama de clases UML representando los añadidos para permitir el guardado de estilos (resaltando en amarillo los cambios). Elaboración propia.

4.7. Interacción

La interacción entre el jugador y el diálogo en el videojuego se puede clasificar en tres apartados principales: la perteneciente a iniciar el diálogo; la interacción permitida durante la conversación; y la relativa a elegir opciones de diálogo.

4.7.1. Inicio de diálogo

El diálogo con los personajes o elementos del entorno se puede iniciar de dos formas, dependiendo del nivel de control que tiene el jugador: por un lado, mediante la pulsación de una tecla determinada cuando el personaje principal se encuentra cerca de un personaje, el caso más habitual en videojuegos; y, por otro lado, de forma automática cuando el personaje principal se encuentra cerca de una zona concreta.

Antes de indagar en la implementación, cabe destacar cómo se distingue dentro de Unity si un personaje se encuentra cerca de otro. En Unity cada objeto, ya sea un personaje o un ítem, puede tener un colisionador asociado, el cual se puede entender como un objeto invisible que suele envolver al objeto. En el caso de que el colisionador de otro objeto entre en contacto se emite un evento, estas colisiones se conocen como *triggers*, traducido como disparador, ya que la colisión dispara dicho evento. En Unity existen dos *triggers* principales: para cuando se entra dentro del colisionador y para cuando se sale del rango del colisionador.

La emisión de estos eventos es la clave para que los personajes de un videojuego actúen en base a la proximidad de otros. De modo que, considerando los dos casos mencionados, se pueden diferenciar entre ellos atendiendo a cómo reaccionan los personajes cuando el personaje principal

colisiona con ellos. Para ello se elaboraron dos clases: *TriggerTalkDialogue* y *TriggerAutoDialogue*.

La funcionalidad de *TriggerTalkDialogue* se basa en empezar la conversación solo si se está cerca del personaje principal y se ha pulsado una tecla concreta. Sin embargo, esta situación propone dos problemas a resolver: por una parte, se deberá controlar la repetición de las llamadas para comprobar si la tecla deseada se ha pulsado, y, por otra parte, en el caso de que dos o más personajes a la vez se encuentren cerca del personaje principal, con tal de decidir cuál de ellos será el elegido. Incidentalmente, ambos problemas se podían resolver creando un controlador central que recibiera la información de los personajes que entren y salgan de la colisión. En primer lugar, se podrá establecer en una sola parte del código la comprobación de la tecla pulsada, pudiendo desactivar esta comprobación dependiendo de otros factores. En segundo lugar, a medida que entren y salgan personajes del rango del personaje principal, se clasificarán dentro de este controlador, de tal modo que siempre hay un principal candidato.

Para permitir esta funcionalidad se creó una clase abstracta *DialoguePromptController*, con tal de permitir otras implementaciones en otros proyectos, con dos métodos *Open(DialogueActer)* y *Close(DialogueActer)* para indicar la entrada y salida de personajes, respectivamente. Para nuestro proyecto se desarrolló una implementación con tal de favorecer el personaje que se encontraba más cercano al personaje principal, tal como se puede comprobar en la Figura 68, al que denominamos *ClosestPromptController*. Paralelamente, se añadió una propiedad de *DialoguePromptController* en la clase *DialogueManager* y, finalmente, para agregar un punto de acceso público se ofreció un nuevo método *OpenDialoguePrompt(DialogueActer)* y *CloseDialoguePrompt(DialogueActer)*. De este modo, la funcionalidad de *TriggerTalkDialogue* se simplifica a llamar a *OpenDialoguePrompt* (cuando entre dentro del rango) y *CloseDialoguePrompt* (cuando salga del rango) del *DialogueManager* actual.

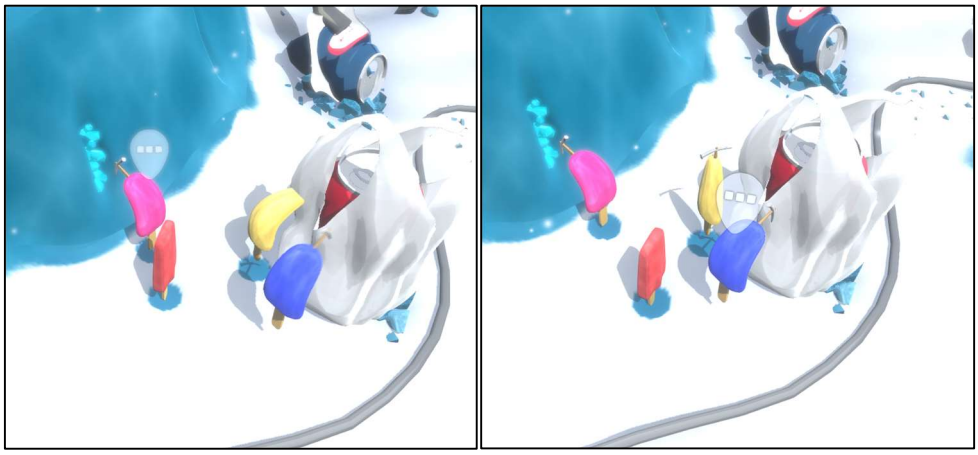

Figura 68. Captura de pantalla de juego haciendo uso de ClosestPromptController y DialogueActer para activar el indicador del personaje con diálogo más cercano. Elaboración propia.

La funcionalidad de *TriggerAutoDialogue* es relativamente más sencilla, ya que, una vez detectado la entrada del personaje principal se llama directamente al método StartDialogue del *DialogueManager* actual. Cabe destacar que, dado que cada personaje tiene una propiedad de tipo *DialogueActer* y esté es transmitido por estas clases *trigger*, se consideró que se podía comunicar al personaje el estado de su diálogo a través de su *DialogueActer*, es por ello que se añadieron métodos para informar que el jugador estaba cerca o se alejaba y para cuando el jugador había sido seleccionado o deseleccionado desde *DialoguePromptController*. La posibilidad de ampliar y modificar métodos abstractos en implementaciones concretas de *DialogueActer* es útil para

personalizar las reacciones de cada personaje. Esta funcionalidad es la que permite activar el indicador en la Figura 68, y la que, en otras partes del juego, permite a los personajes rotar para mirar al personaje principal cuando hablan. Los añadidos a las clases abstractas existentes y las nuevas clases el control de colisiones y candidatos se puede apreciar en la Figura 69.

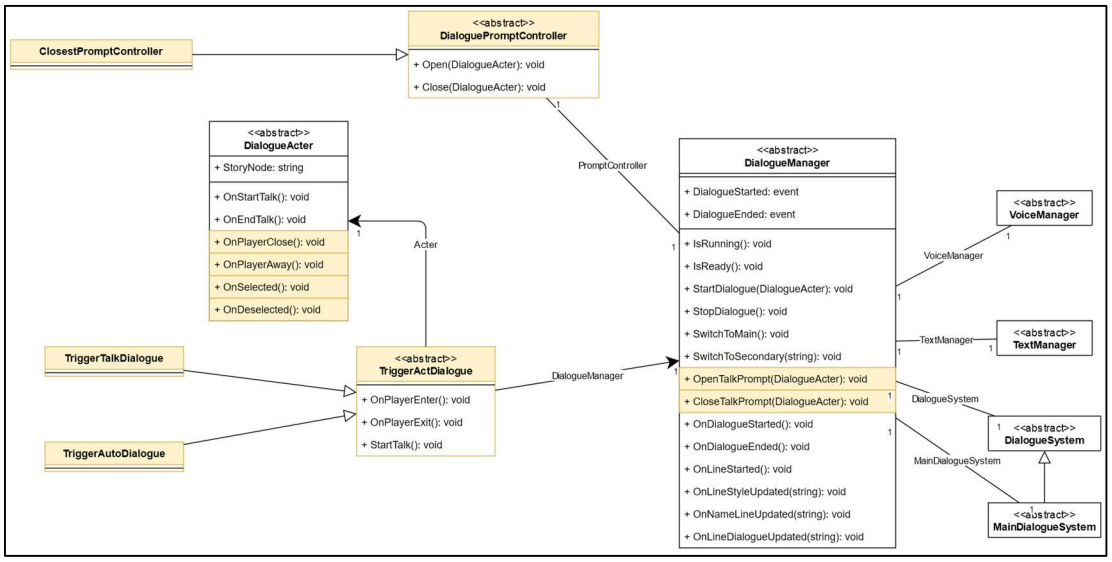

Figura 69. Diagrama de clases UML con añadidos para soportar la detección y selección del candidato para empezar la conversación (resaltando en amarillo los cambios). Elaboración propia.

4.7.2. Control durante el diálogo

Una vez iniciado el diálogo, es el *DialogueActer* del personaje el que contiene la información sobre la interacción del jugador permitida durante el transcurso de este. De modo que, se plantean variantes de *DialogueActer* atendiendo a dos factores: en primer lugar, en cuanto al bloqueo de movimiento del personaje principal (el grado de bloqueo), y, en segundo lugar, relacionado con el control del jugador para pasar a las sucesivas líneas de diálogo (el grado de automatización).

La configuración bloqueante inmoviliza al personaje principal y restringe el movimiento de este, estableciendo el diálogo como el elemento central al que el jugador debería prestar atención. El uso del resto de sistemas durante el diálogo también permanece bloqueado, exceptuando el menú de pausa. Por otro lado, la configuración opuesta, no bloqueante, no ofrece ninguna restricción de movimiento al jugador, de modo que la conversación se puede entender como un elemento secundario. La configuración no automática cede al jugador el control para avanzar en las sucesivas líneas de diálogo, requiriendo que el jugador pulse una determinada tecla para, o bien terminar la línea de texto actual, o pasar a la siguiente si la actual ya ha terminado. En contraposición, la configuración automática elimina este requerimiento, y aplica un retardo de tiempo concreto al finalizar una línea antes de pasar de forma automática a la siguiente, creando la sensación de que la conversación no está bajo el control del jugador.

Con estos dos factores presentes, se pude elaborar una tabla (Tabla 7) para examinar con facilidad las diferentes combinaciones y tipos de *DialogueActer* resultantes:

	Bloqueante	No bloqueante
Automática	*DialogueCinematic*	*DialogueAnnouncer*
No automática	*DialogueTalker*	-

Tabla 7. Variantes de DialogueActer atendiendo al grado de bloqueo y automatización. Elaboración propia.

La variante automática y bloqueante restringe totalmente la interacción por parte del jugador, es por ello que se puede relacionar con escenas cinemáticas, donde el jugador solo puede seguir la conversación y no puede controlar su duración, de modo que se denominó *DialogueCinematic*. La variante automática y no bloqueante releva la conversación a un segundo plano, como si se tratara de un anuncio que ocurre sin su control (razón principal de su nombre, *DialogueAnnouncer*). Finalmente, *DialogueTalker* representa uno de los modos más comunes de diálogo en los videojuegos, ya que ofrece un mínimo de interactividad al jugador para seguir actuando con el juego (no automático) mientras limita sus otras actividades para centrar su atención en la conversación (bloqueante).

Como se puede comprobar, la variante no bloqueante y no automática no se considera al tratarse de una configuración contraproducente, ya que se trataría de una conversación en la que el jugador es libre de moverse (no bloqueante), pero sigue teniendo la necesidad de interactuar para avanzar en el diálogo (no automático), proporcionando al jugador dos acciones activas que demandan su atención de forma simultánea. En la Figura 70 se puede comprobar los añadidos al diseño de clases en esta sección.

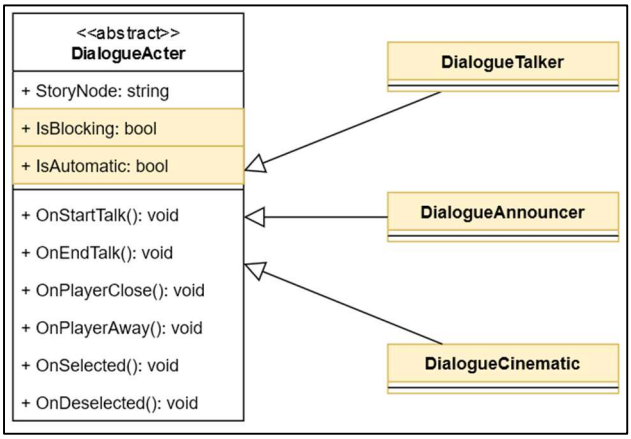

Figura 70. Diagrama de clases UML con los añadidos para accionar y controlar la interacción del diálogo (resaltando en amarillo los cambios). Elaboración propia.

4.7.3. Elección de opciones de diálogo

La interacción del jugador para elegir opciones y ramificar la narrativa se puede elaborar de distintas formas, tal como se ha comprobado durante el capítulo 4.1. Sin embargo, previamente a este desarrollo cabe elaborar un mecanismo a través de *DialogueManager* para permitir la abstracción de un sistema de elecciones. En los sistemas de diálogo descritos en el capítulo 4.2.1, se posibilita la interrupción del diálogo para que el jugador elija entre las opciones que se han establecido a través del guion. Dado que dejan en manos de los equipos de desarrollo la presentación de estas opciones se puede elaborar un sistema externo, tal que reciba la señal de inicio de una sección de opciones, junto con las opciones para ese momento. Esta funcionalidad se encapsuló en una clase abstracta denominada *ChoiceSystem*, donde se incluyó un método para comunicar la lista de opciones. También se elaboró una clase *DialogueChoice* para unificar la representación de una opción, ya que en cada sistema se utiliza una clase distinta y esta no debería ser conocida por *ChoiceSystem*. Esta se basó en las clases homólogas de los tres sistemas de referencia, con dos propiedades: una de tipo *string*, para denotar el texto asociado a la opción; y una de tipo entero, para representar el identificador de la opción. De este modo *ChoiceSystem* recibiría una lista de *DialogueChoice*. Para el flujo de opciones desde el sistema de diálogos se introduce en *DialogueManager* un método *OnChoicesStarted(*DialogueChoice)*. De forma opuesta también se introduce *OnChoiceSelected(DialogueChoice)* para permitir la comunicación

de la opción elegida por parte del sistema de opciones. Con el objetivo de acomodar *MainDialogueSystem* a esta nueva funcionalidad se agrega en este un nuevo método abstracto *RequestSelectChoice(DialogueChoice)*, tal que se transmita la opción elegida y se devuelva el control al sistema de diálogos.

Una vez aplicadas estas modificaciones solo sería necesario implementar una clase concreta de *ChoiceSystem*, atendiendo a la interacción deseada. Con el objetivo de hacer uso de la reusabilidad de sistemas se crearon dos clases concretas. En primer lugar, una implementación donde cada opción representa un botón, llamada *ButtonChoiceSystem*, de forma que cada botón muestra el texto de la opción y el botón pulsado constituye la elección resultante en el diálogo. Dado que el número de opciones en cada momento es variable, se divisó un sistema para crear tantos botones como opciones se habían recibido y organizarlos de forma justa en pantalla. Para ello se decidió utilizar los componentes de disposición automática[109] presentes en Unity, los cuales organizan sus componentes de forma dinámica para ocupar un espacio determinado. De esta forma, se pueden instanciar tantos botones como sean necesarios y colocarlos dentro de un componente de disposición horizontal (*HorizontalLayoutGroup*), tal que se coloquen de forma horizontal sin que haya superposición entre ellos y queden automáticamente organizados. De este modo, *ButtonChoiceSystem* hará uso de este componente al recibir la lista de *DialogueChoice* y, una vez el jugador ha seleccionado el botón, llamaran al método *OnChoiceSelected* adjuntando como parámetro el *DialogueChoice* asignado a botón.

En segundo lugar, se divisó una implementación alternativa donde la elección del jugador es a través de palabras introducidas en el teclado, tal que las opciones no son visibles pero el sistema elige internamente la opción más relevante atendiendo a la palabra introducida. Este sistema se denominó *InputChoiceSystem* debido a la necesidad de introducir (*input*) palabras por parte del jugador. En esta implementación de *ChoiceSystem* las opciones entrantes se guardan para su posterior uso y se muestra en pantalla un campo de texto. Una vez el jugador ha introducido el texto, se recorre la lista de *DialogueChoice* para encontrar una opción cuyo texto contenga esta palabra y se selecciona para llamar a *OnChoiceSelected*. Este mecanismo de elección presenta dos problemas a resolver: por un lado, en el caso de que dos o más opciones contengan la misma palabra, se deberá tener en cuenta cual de esas opciones es la seleccionada; por otro lado, en el caso en el que no se encuentre la palabra en ninguna opción se deberá proporcionar una opción por defecto. Para solucionar el problema de la duplicidad, se decidió que la primera opción que cumpliera con el requisito sería la elegida, independientemente de las subsiguientes opciones. Sin embargo, para atajar mejor estos problemas, se consideró tomar dos precauciones previas. Por un lado, el guion que se proporcionara al sistema de diálogos debería contener opciones tal que su texto está formado por palabras clave sueltas, así como procurar que no se repitan estas palabras en otras opciones, de este modo se facilitaría el posterior reconocimiento de la opción más relevante. Por otro lado, se debería procurar, también desde el guion, que la última opción se considerarara la opción por defecto (de modo que su texto no se tendría en cuenta para la comprobación). Con estas dos consideraciones previas se asegura un óptimo funcionamiento para este sistema de opciones. En la Figura 71 se puede apreciar las clases añadidas y modificaciones propuestas en el diseño de clases.

[109] Disponible en <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/comp-UIAutoLayout.html>

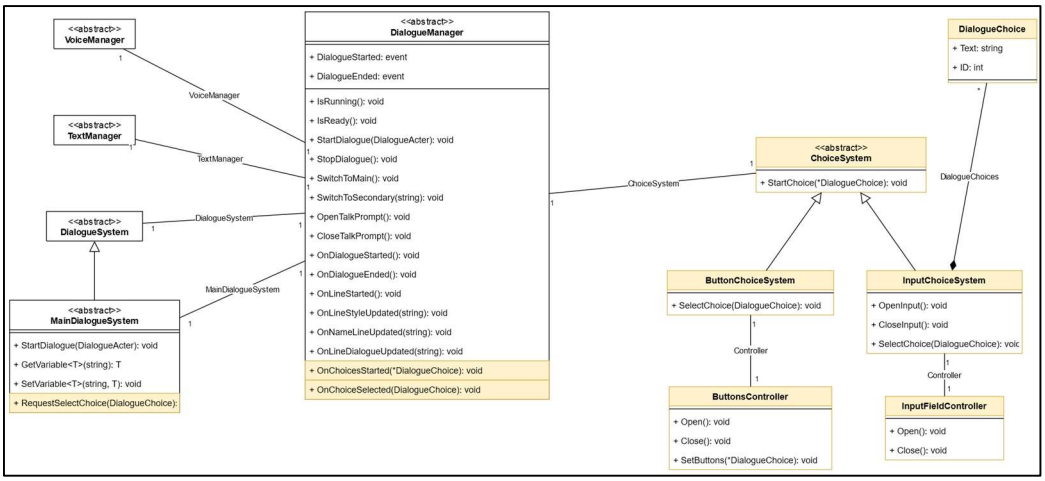

Figura 71. Diagrama de clases UML con las adiciones relacionadas con el control de opciones de diálogo (resaltando en amarillo los cambios). Elaboración propia.

4.8. Internacionalización del diálogo

Uno de los objetivos del segundo MVP fue añadir soporte para el inglés en todo el juego. Los componentes de interfaz fueron localizados a través de la librería nativa de Unity (Unity Localization), sin embargo, para los diálogos, donde existen variables que se pueden incrustar y marcas de formato (con RichText) no se pudo recurrir a las funcionalidades de esta herramienta. La localización del guion se delegó a la ofrecida por el sistema actual de diálogos, Yarn Spinner. La estrategia de esta herramienta es simple: utilizando el archivo de texto base, la herramienta introduce un identificador a cada línea de texto, de modo que permite crear archivos *Comma Separated Value* (CSV)[110] donde relacionar el identificador de la línea con su contenido traducido. Para ello, el archivo de texto, reconocido como el archivo base, se le debe asignar un idioma, de modo que solo es necesario crear un archivo CSV por cada idioma adicional. Las variables de diálogo que puedan existir aparecen en estos archivos CSV como representaciones simbólicas, de modo que no es necesario conocer el nombre de la variable y se centraliza su uso en el archivo de texto base. En cambio, cualquier etiqueta añadida, como en el caso de las provenientes de RichText, sí deben ser copiadas al archivo CSV. Un ejemplo real se puede comprobar en la Figura 72, donde se aprecian los identificadores al final de cada línea, las cuales reaparecen en el CSV para la traducción al inglés. Por último, el idioma a aplicar se indica a *DialogueRunner* estableciendo una variable interna *textLangauge*. De este modo, al iniciar el nivel, *YarnDialogueSystem* extrae el valor del idioma actual a través de la librería de localizaciones de Unity y establece la propiedad *textLanguage* del *DialogueRunner* actual acorde con su valor.

```
16 title: Naranjito
17 ---
18 Naranjito: Deberías volver a la ciudad antes de que te vean <b>los helados</b> #line:0628603
19 Naranjito: No permiten que nadie merodee por aquí fuera de su turno #line:0e1915c
20 Naranjito: Recuerda usar {$crouchKey} para que no te detecten #line:0e01947
21 ===
```

```
4 line:0628603,Orangey: You should go back to the city before those <b>ice creams</b> see you,Lv1_Dialogue,Naranjito,11
5 line:0e1915c,Orangey: They don't let anyone wonder about around here off-shift,Lv1_Dialogue,Naranjito,12
6 line:0e01947,Orangey: Remember to use {0} to remain undetected,Lv1_Dialogue,Naranjito,14
```

Figura 72. Captura de pantalla con un fragmento de guion en castellano en el archivo de texto base (arriba) y su fragmento de guion traducido al inglés en archivo CSV (abajo). Elaboración propia.

[110] Archivo similar en estructura y utilidad a Microsoft Excel, usado para almacenar tablas. Más información en <https://es.wikipedia.org/w/index.php?title=Valores_separados_por_comas&oldid=128403877>

4.9. Conexión con Instagram

Para una apartado promocional del segundo MVP se pensó en integrar la cuenta del juego existente en la red social de Instagram, ya que es una plataforma centrada en compartir imágenes y vídeos. Dado que una de las características más relevantes del proyecto desarrollado es la estética y el entorno de juego, se consideró la plataforma más apropiada para popularizar el videojuego. El objetivo de esta integración sería introducir un personaje en el nivel que se habría encontrado un teléfono móvil y, desconociendo su utilidad, le hablaría al personaje sobre aquello que veía en la pantalla: la cuenta del juego en Instagram y los comentarios de los usuarios a sus publicaciones. De modo que, a efectos de este proyecto, sería necesario acceder a las últimas publicaciones de la cuenta y consultar los comentarios realizados por los seguidores.

Esta funcionalidad requeriría operar con los servicios API de Instagram, así como ofrecer los datos devueltos en forma de línea de texto para poder ser presentada en pantalla y pronunciada. La conexión con Instagram debería considerarse, por tanto, un sistema de diálogos secundario, ya que los datos extraídos serían cambiantes dependiendo del momento de la consulta y no se podría conocer de antemano el número de comentarios.

Con este objetivo, se procedió a la búsqueda e integración de su API. Existen dos tipos de acceso disponible para su API [23]: la llamada *Instagram Basic Display API*, la cual está dirigida a usuarios comunes de la plataforma, y contiene puntos de acceso de solo lectura a sus publicaciones; y *Instagram Graph API*, dirigida a cuentas profesionales de Instagram (cuentas de negocio y de creadores), las cuales realizan un uso comercial de la plataforma con el objetivo de vender una marca o producto. Este último servicio de API ofrece un rango de funciones más amplio, permitiendo modificar publicaciones y acceder a datos de tracción de la cuenta, además, el requisito de acceder a los comentarios solo está disponible en esta *Instagram Graph API*, de modo que se decidió por acceder a través de esta.

Sin embargo, para poder empezar a usar sus servicios se debían realizar cambios en la cuenta existente: en primer lugar, se debía cambiar el tipo de cuenta a una de tipo profesional, de modo que se eligió cambiar a una cuenta de negocio, ya que es la apropiada para una marca o producto [24]; en segundo lugar, se necesita crear y vincular una página de Facebook, ya que las llamadas a la API son inicialmente dirigidas a Facebook, que actúa como punto central desde donde acceder a los servicios relacionados (como los de una cuenta de Instagram). De modo que se creó una cuenta de Facebook, donde se generó una nueva página del juego y se vinculó con la cuenta del juego en Instagram. Una vez realizados estos cambios se procede a activar las funcionalidades de desarrollador en la cuenta de Facebook y crear una aplicación, la cual se entiende como el contenedor de los servicios contratados. Dentro de esta nueva aplicación, llamada «Frozen-Out», se activaron los servicios de *Instagram Graph API*. Estos elementos se pueden consultar en la Figura 73. Por último, la propiedad necesaria para autenticarse en el servicio es un token de acceso, el cual se puede obtener a través de la página de desarrollador de Facebook. Estos tokens de acceso, sin embargo, tienen un periodo de validez y requerirían de una renovación pasado este periodo. Dado que esta funcionalidad se pretendía usar para la demo, la cual debería ser usada y probada por los usuarios durante las primeras semanas tras su publicación, se procuraría un token de acceso de larga duración (en concreto dos meses) para asegurarse de que esta funcionalidad sería accesible durante el periodo de circulación de la demo.

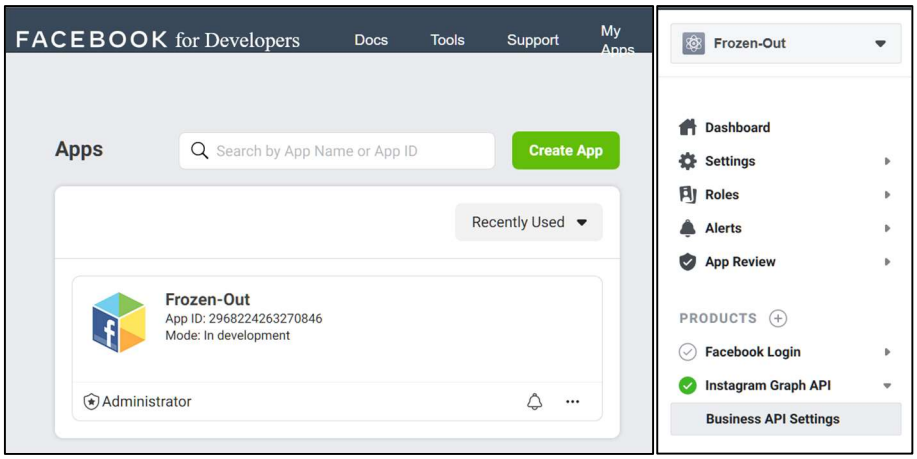

Figura 73. Captura de pantalla del portal para desarrolladores de Facebook listando las aplicaciones creados (izquierda) y los productos asociados a esta aplicación (derecha) entre las que se encuentra Instagram Graph API. Elaboración propia.

El kit de desarrollo proporcionado por Facebook para acceder a su API no está disponible para Unity, de modo que se optó por implementar las llamadas en formato REST. En primer lugar, se crearon clases para representar los objetos devueltos en formato JSON, basados en la estructura presente en la documentación: en cuanto a las entidades pertenecientes a Facebook[111], sería necesario incluir clases para *FacebookUser*, el cual representa el usuario de Facebook y *FacebookAccount*, que simboliza las cuentas de Facebook (donde entran las páginas y cuentas personales); para los objetos correspondientes a Instagram[112], se introdujeron las clases *InstagramAccount*, *InstagramMedia*, que representa las publicaciones de Instagram (ya sea compuesta por imágenes o vídeo) e *InstagramComment*, el cual encapsula los comentarios a las publicaciones.

De forma similar, se proporcionaron clases para representar los modelos que se iban a usar en las clases, ya que las entidades descritas tienen la función de guardar y encapsular los objetos JSON de las llamadas REST y todas sus propiedades son cadenas de caracteres. Estas clases modelo serían similares en contenido a las clases entidades, sin embargo, contendrían la información en tipos de dato nativos: *FacebookPage* representa la entidad *FacebookAccouont* específica para páginas de Facebook; *User* contiene propiedades de *InstagramAccount*; *Post* y *Comment* tienen el objetivo de abstraer el contenido de *InstagramMedia* y *InstagramComment*, respectivamente. Las propiedades para cada elemento se pueden consultar en la Figura 74. Con la ayuda de estas clases, se puede proceder a crear una clase repositorio para acceder a los servicios de la API, obtener los datos encapsulados en las clases entidad y devolverlos en clase modelo para facilitar su acceso a las clases que usen el repositorio. Esta clase repositorio contaría con las siguientes funciones y llamadas:

— *GetPosts(User)*, realiza una llamada para obtener las publicaciones del usuario de Instagram proporcionado y las encapsula en objetos de tipo *Post*: https://graph.facebook.com/v7.0/{user.Id}/media?fields=id,username,caption,timestamp,like_count,comments_count.
— *GetComments(Post)*, obtiene los comentarios de la publicación proporcionada (ya que se trata de llamadas distintas) en forma de lista de *string*: https://graph.facebook.com/v7.0/{post.Id}?fields=comments.

[111] Disponible en <https://developers.facebook.com/docs/graph-api/reference/page/>
[112] Disponible en <https://developers.facebook.com/docs/instagram-api/reference>

— *GetUserFromFacebookPage(FacebookPage)*, realiza una llamada para extraer la cuenta de Instagram (*User*) vinculada a la página de Facebook proporcionada: https://graph.facebook.com/v7.0/{page.Id}?fields=instagram_business_account.
— *GetCurrentFacebookUser()* obtiene los datos del usuario de Facebook (*FacebookUser*) con el que se ha autenticado: https://graph.facebook.com/v7.0/me?fields=id,name.
— *GetCurrentFacebookUserPages()* consulta y devuelve una lista de páginas (*FacebookPage*) que están a cargo del usuario con el que se ha autenticado: https://graph.facebook.com/v7.0/me/accounts.

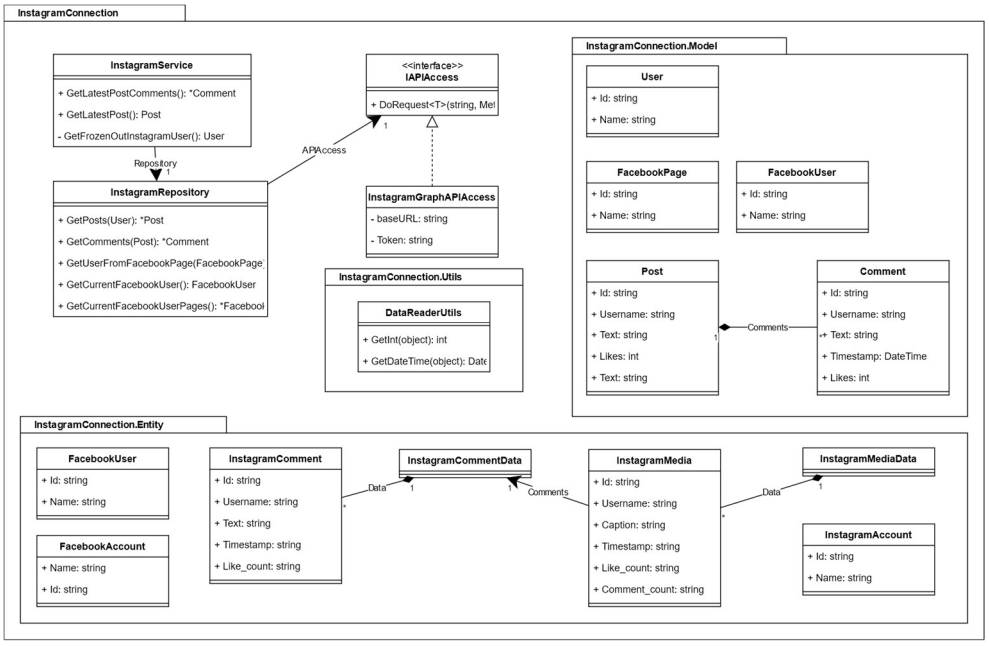

Figura 74. Diagrama de clases UML representando las clases relacionadas con la conexión con Instagram. Elaboración propia.

Para poder efectuar las llamadas REST y ayudar en la serialización de datos provenientes de la API de Facebook se utilizó la librería RestSharp, en concreto una versión modificada pensada para su uso en Unity. Esta librería permite hacer llamadas web y serializar datos provenientes en formato XML o JSON directamente a propiedades de clases. Con el objetivo de encapsular su funcionamiento y tener accesible las propiedades necesarias para autenticar la conexión, en este caso el token de acceso, se facilitó una clase *InstagramGraphAPIAccess* con un método *DoRequest<T>* proporcionando como parámetro la cadena de caracteres que identifica los recursos que se desean acceder y el método HTTP sobre el que efectuar la llamada (en nuestro caso siempre serían GET). Esta clase también permite centralizar el control de errores. En la Figura 75 se puede comprobar el uso de esta clase: la llamada obtiene un objeto entidad que posteriormente es convertido a un objeto modelo (proceso también conocido como mapeo de objetos). Finalmente, se creó una clase *InstagramService* para encapsular el proceso de acceder a la cuenta de Instagram a partir de la cuenta de Facebook, así como ofrecer directamente la última publicación y los comentarios de esta publicación.

```
1 reference
public ICollection<Comment> GetComments(Post post)
{
    string resourceQuery = $"{post.Id}?fields=comments" + "{id,username,text,timestamp,like_count}";

    IRestResponse<InstagramMedia> response = ApiAccess.DoRequest<InstagramMedia>(resourceQuery, Method.GET);

    ICollection<Comment> comments = new List<Comment>();
    foreach (InstagramComment instaComment in response.Data.Comments.Data)
    {
        Comment comment = MapInstagramComment(instaComment);
        comments.Add(comment);
    }

    return comments;
}
```

Figura 75. Fragmento de código perteneciente al método GetComments de la clase InstagramRepository. Elaboración propia.

En cuestión de diálogo, era necesario proveer una clase concreta de *SecondaryDialogueSystem* para usar este servicio y proporcionar las líneas a *DialogueManager*. Esta clase se llamó *InstagramDialogueSystem* y contendría una propiedad de tipo *InstagramService*. La implementación del método *StartDialogue()* obtiene la última publicación y llama a los métodos de *OnLineStarted()* y *OnLineDialogueUpdated(string)* de *DialogueManager* para introducirse en el flujo de texto. Uno de los problemas con los comentarios y texto de publicaciones online es el uso de emoticonos, ya que los componentes de texto de Unity no soportan estos caracteres, de modo que antes de comunicar las líneas de texto se efectuó una limpieza del texto recibido (tanto de los comentarios como de los nombres de usuario) para eliminar estos caracteres; por otro lado, en los comentarios también se recortó su longitud a 80 caracteres para asegurarse que el texto tuviera cabida en el componente de interfaz donde se presentara.

Al tratarse de un servicio web, este está regido por la conexión a Internet del jugador, de modo que se realizaron comprobaciones para detectar fallos de red y establecer un mecanismo para comunicar al sistema principal estos posibles errores. La comunicación del resultado de la conexión se realizó a través de las variables de diálogo, de este modo serviría para que, una vez devuelto el control al sistema principal, este pudiera mostrar líneas de texto en consecuencia. El uso de estas variables se realizó en los siguientes casos: por un lado, cuando, por problemas de red, no ha sido posible crear una instancia de *InstagramService* se establece una variable «instagramerror»; similarmente, en el caso en el que el servicio se ha podido crear pero se detectan excepciones provenientes de RestSharp relacionadas con un fallo de conexión o *timeout* (demora de tiempo) también se establece la variable «instagramerror»; por otro lado, si no ha habido errores de conexión pero la última publicación no contiene comentarios se establece una variable llamada «instagramempty». En todo caso, se usaría el método *DialogueManager.SwitchToMain()* para devolver el control al sistema principal y continuar la conversación.

Desde *NormalDialogueManager* se incluyó una propiedad de tipo *SecondaryDialogueSystem* para referenciar el sistema de Instagram, y se modificó el método *SwitchToSecondary(string)* para cambiar el sistema activo a este sistema secundario (ignorando por tanto el parámetro de tipo *string* ya que solo existe un sistema secundario). El cambio de sistema se producirá por el sistema de diálogos principal, a través de llamadas que permiten invocar el método externo *SwitchToSecondary*, tal que se pausa el funcionamiento de este sistema principal hasta que las líneas de texto se han finalizado en el sistema secundario.

5. Demos en Unity

Durante esta sección se planea presentar situaciones prácticas donde los sistemas descritos se puedan combinar entre ellos a través del gestor de diálogos, el cual representa el sistema central que posibilita la comunicación de información entre ellos. Estas situaciones también servirán para poner de manifiesto la intercambiabilidad de implementaciones que permite este gestor.

El mencionado gestor de diálogos se incluyó a las escenas de Unity en forma de *prefab*, tal que los componentes mínimos necesarios para actuar el diálogo estuvieran incluidos. En la Figura 76 se puede comprobar la estructura de este *prefab*, donde se pueden distinguir los diferentes sistemas descritos durante el capítulo 4. Por un lado, dentro del objeto DialogueManager se encuentra la implementación concreta de *DialogueManager*, en nuestro caso *NormalDialogueManager*, así como la clase de guardado de estilos (*DictionaryStylesStorage*). Además, se encuentran organizados como subobjetos de este los sistemas implementados en este proyecto: el objeto YarnSystem contiene la clase *YarnDialogueSystem*, así como las clases provenientes de Yarn Spinner; el objeto InstagramSystem contiene la clase *InstagramSystem*; InputChoiceSystem contiene la clase *InputChoiceSystem*, aunque los componentes de interfaz se encuentran en el objeto exterior InputChoiceCanvas, con el cual se debe comunicar; similarmente, el objeto ButtonChoiceSystem contiene la clase *ButtonChoiceSystem* pero los componentes de interfaz residen en el objeto ButtonChoiceCanvas.

Por otro lado, en los objetos hermanos se encuentran: la clase *BoxTextManager*, dentro de DialogueCanvas, donde residen los componentes de interfaz para el nombre y la caja de diálogo; la clase *ClosestDialoguePrompt* dentro de DialoguePrompt; la implementación de voz «animalese», *AnimaleseVoiceManager* dentro del objeto AnimaleseVoice; y, similarmente, la clase *AzureTTSManager* reside en AzureTTSVoice.

Cabe destacar la importancia de *NormalDialogueManager*, ya que es aquí donde se establecen los sistemas que se encargaran de la proporción de líneas (propiedad *MainDialogueSystem*), de la presentación (propiedades *TextManager* y *VoiceManager*), del guardado de estilos (propiedad *StylesStorage*), de la administración de personajes para hablar (propiedad *PromptController*), de la administración de opciones (propiedad *ChoiceSystem*) y de la proporción de líneas de Instagram (propiedad *InstagramDialogueSystem*). Adicionalmente, se puede ver la propiedad *LevelManager* para poder consultar y accionar sistemas externos a través del administrador del nivel. Estas propiedades se pueden establecer arrastrando los objetos anteriormente mencionados al campo deseado en *NormalDialogueManager*. Como se puede comprobar, en el *prefab* ya se han cumplimentado los campos con los sistemas que se usan por defecto en el juego.

Por parte de los objetos de personajes, en la Figura 77 se encuentra la instancia de un personaje habitual con capacidad para entrar en una conversación. En este se puede ver que reside una propiedad de tipo *DialogueActer* para identificar el diálogo que activan (en concreto este personaje contiene uno de tipo *DialogueTalkerDirect*, el cual extiende el funcionamiento de *DialogueTalker* para el uso de indicadores y permitir el giro del personaje si se habla con él). También se puede comprobar que contiene una propiedad *TriggerTalkDialogue* para poder detectar al personaje principal.

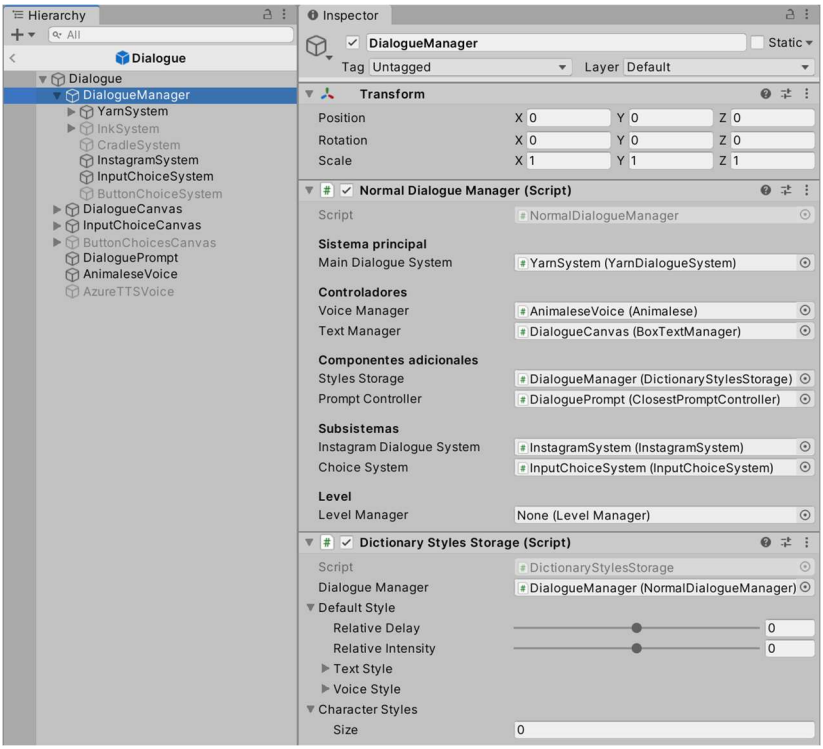

Figura 76. Captura de pantalla de Unity mostrando el prefab de diálogos: su jerarquía (izquierda) y el detalle del inspector del objeto DialogueManager (derecha). Elaboración propia.

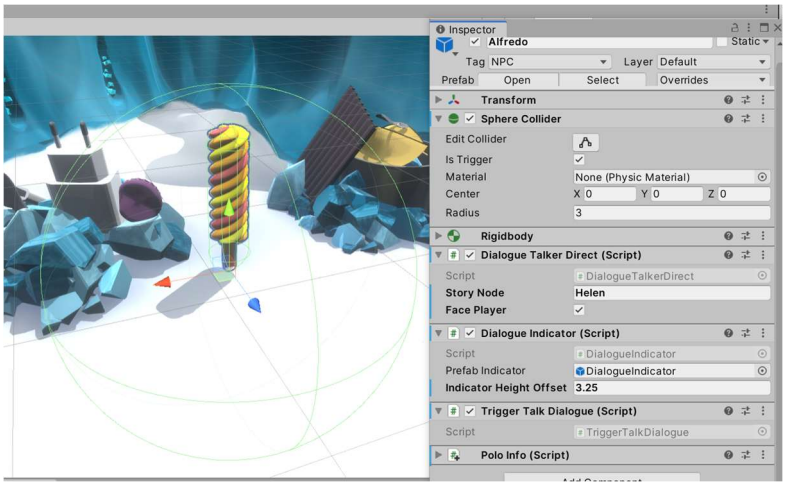

Figura 77. Captura de pantalla de Unity mostrando los componentes de diálogo de un objeto de personaje. Elaboración propia.

5.1. Estilos de texto

En esta sección se incluyen ejemplos relacionados con el uso de estilos. En primer lugar, se elaboró una escena con el objetivo de combinar los estilos globales, establecidos a través de *StylesStorage*, con efectos focalizados de las etiquetas RichText. En la Figura 78 se puede comprobar el guion usado para este ejemplo, donde se puede observar la referencia a estilos globales con «++Panico» y «++Feliz». Estos estilos son configurados en Unity a través de la propiedad *Styles* de *DictionaryStylesStorage* tal como se puede ver en la Figura 79, donde se aprecian el cambio de fuente y de color, así como de su intensidad (que afectará al tamaño de fuente). Una vez establecidas las configuraciones, el texto sería mostrado en el juego tal como se captura en la Figura 80. Cabe remarcar que los efectos dinámicos, como el de salto, no se pueden apreciar en la figura al estar ocurriendo en tiempo real y efectúan una animación a cada letra y en bucle.

```
===
title: Palanquilla
---
<< if not visited("Palanquilla") >>
    Palanquilla++Panico: Madre mía... he perdido la <b>manivela</b> y ahora no se puede usar el <b>ascensor</b>
<< endif >>
<< if has_item("lever") >>
    Palanquilla++Feliz: Me has salvado la vida... ahora mismo pongo en marcha este trasto #line:0f87ade
    Palanquilla: Mira como gira, el alcalde estaría orgulloso de mí #line:0723cae
<< endif >>
===
```

Figura 78. Fragmento de guion en formato de Yarn Spinner para ejemplificar el uso de estilos globales. Elaboración propia.

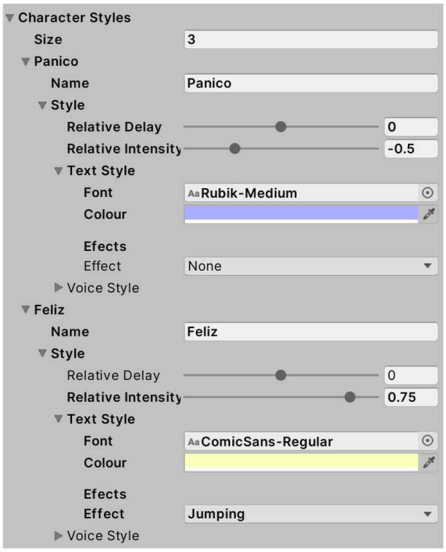

Figura 79. Captura de pantalla del inspector de Unity mostrando la configuración de estilos globales de «Panico» y «Feliz». Elaboración propia.

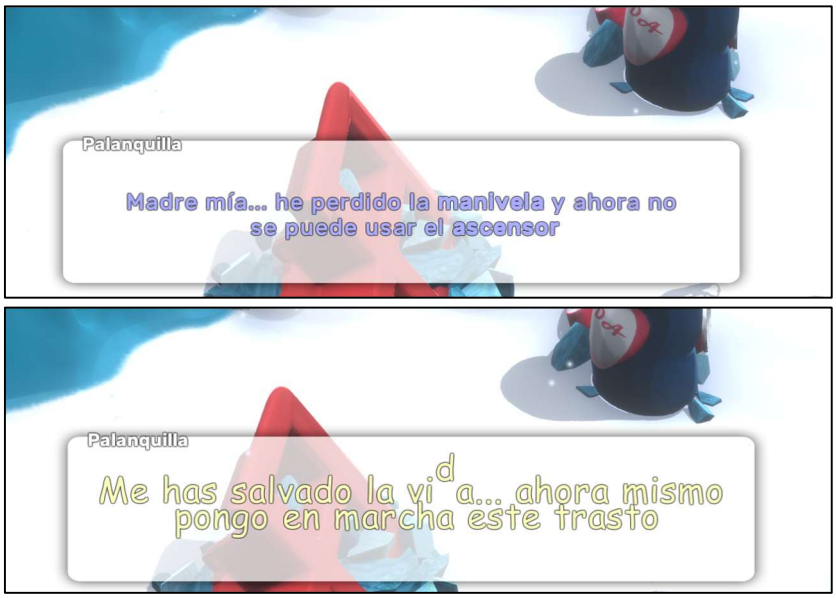

Figura 80. Captura de pantalla de Unity ejemplificando la acción de los estilos globales descritos: «Panico» (arriba) y «Feliz» durante efecto dinámico de salto (abajo). Elaboración propia.

En cuanto al estilo de voz, pese a la dificultad de mostrar este efecto en un medio escrito, se adjunta por completitud. En la Figura 81 se especifica un estilo de voz ligeramente más grave y con el efecto de radio aplicado, así como un efecto de texto interrumpido para simular textualmente la distorsión de voz de un auricular. En la Figura 82 se puede comprobar este estilo en acción, adjuntando el estado de la aplicación de mezcla de sonidos de Windows para demostrar la salida de audio desde Unity.

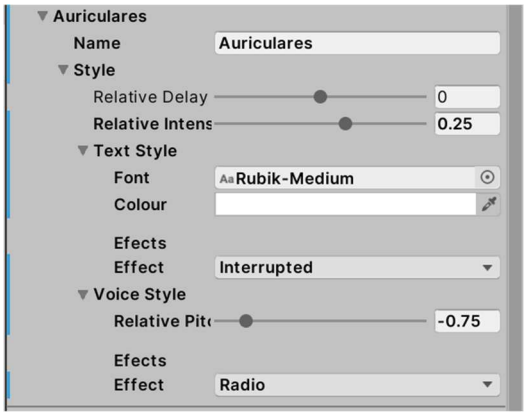

Figura 81. Captura de pantalla del inspector de Unity mostrando la configuración de estilo «Auriculares». Elaboración propia.

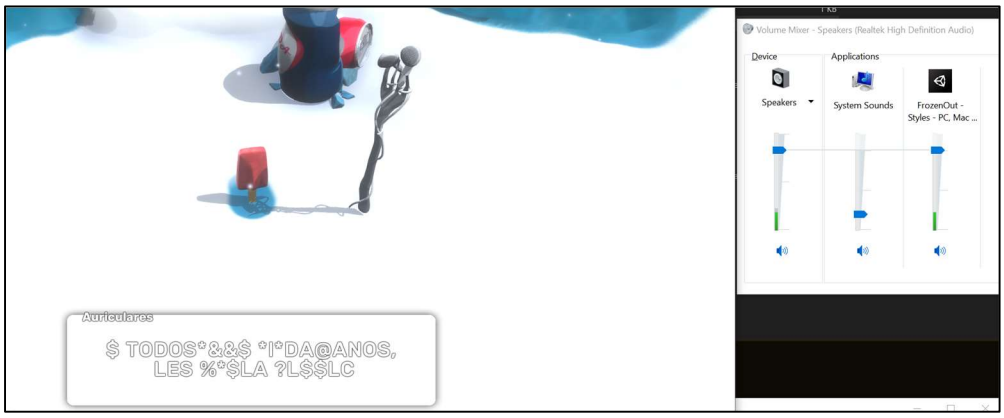

Figura 82. Captura de pantalla de Unity ejemplificando el uso del estilo «Auriculares» (izquierda) y el estado del mezclador de sonido de Windows durante ese momento (derecha). Elaboración propia.

5.2. Opciones

Las opciones establecidas en el guion se pueden mostrar, en nuestro caso, de dos formas principales: a través de botones o a través de la interacción con un campo de texto. Para soportar las opciones se elaboró una pequeña conversación (Figura 83) donde existen cuatro opciones. En la Figura 84 se puede comprobar como las opciones del guion se corresponden con los botones y su texto mostrado en pantalla; la elección del jugador de la opción deseada devuelve el control al sistema de diálogos para mostrar las líneas derivadas de la elección. Por otro lado, en la Figura 85 se ha cambiado el sistema de opciones para mostrarlas siguiendo la implementación de *InputChoiceSystem*: abriendo una caja de texto y dejando que el jugador escriba una palabra que represente la idea que quiere transmitir.

```
title: Options
---
Crema: ¿Qué es lo que quieres saber? #line:0258c5c
[[¿Qué hago aquí?|Trabajo]] #line:0c10584
[[¿Qué es esa nevera tan <b>grande</b>?|Nevera]] #line:0a606a2
[[¿Cómo puedo esconderme de los <color=red>guardias</color>?|Guardias]] #line:062bead
[[Nada más|Nada]] #line:098fd04
===

title: Trabajo
---
Crema: Pues trabajar, que es lo que esperabas #line:025c4fb
===
title: Nevera
---
Crema: La nevera es donde viven los helados pero ni se te ocurra acercarte #line:0484e26
===
title: Guardias
---
Crema: Pulsa {$crouchKey} para hacer menos ruido #line:04ed88d
===
title: Nada
---
Crema: Tú mismo #line:0e94312
===
```

Figura 83. Fragmento de guion en formato de Yarn Spinner para ejemplificar el uso de opciones. Elaboración propia.

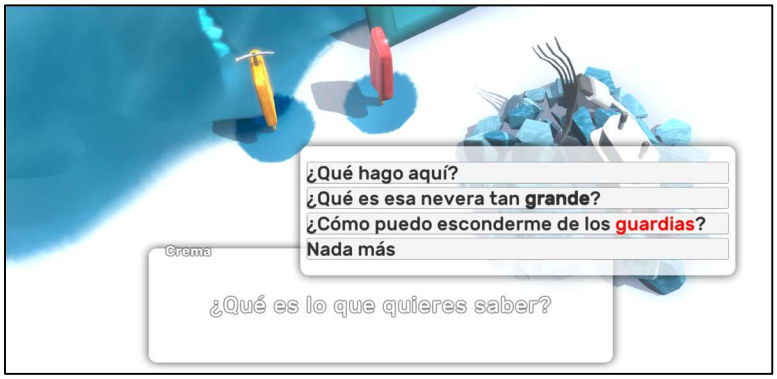

Figura 84. Captura de pantalla de Unity mostrando un evento de opciones en forma de botones. Elaboración propia.

Figura 85. Captura de pantalla de Unity mostrando un evento de opciones a través de una campo de texto. Elaboración propia.

5.3. Sistemas secundarios

Como se ha comentado, la idea de los sistemas secundarios tiene el objetivo de mostrar información proveniente de fuentes externas y de esta forma complementar los sistemas de diálogos que se nutren de información predefinida. Para este proyecto se implementó la conexión

con la cuenta social de Instagram para mostrar los comentarios a publicaciones de forma diegética como conversaciones de juego. En la Figura 86 se puede comprobar la última publicación de la cuenta y los comentarios que recibió. A través de una conversación en Yarn Spinner (Figura 87) se indica el cambio de sistema y se establece un mecanismo para reaccionar a los problemas que puedan ocurrir tras la conexión. Durante el juego, y tal como se ve en la Figura 88, la llamada a Instagram es capaz de recibir y mostrar la información remota.

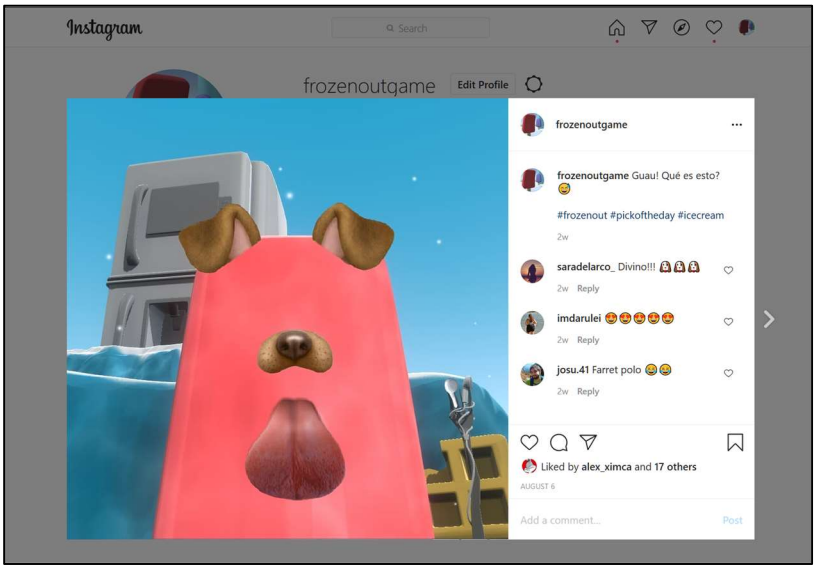

Figura 86. Captura de pantalla de la última publicación y comentarios de la cuenta del juego en Instagram. Elaboración propia.

```
1  title: Instagramo
2  ---
3  Instagramo: Bueno, vamos a ver que dice la gente #line:0b32ab2
4  << startinstagram >>
5  << if $instagramerror >>
6      Instagramo: Parece que este cacharro no funciona #line:03590b8
7  << elseif $instagramempty >>
8      Instagramo: Parece que no tienen nada que decir #line:0ac1e4d
9  << else >>
10     Instagramo: Pues eso ha sido todo #line:00753fa
11 << endif >>
12 ===
13
```

Figura 87. Fragmento de guion en formato de Yarn Spinner ejemplificando la llamada a Instagram y la gestión de posibles errores. Elaboración propia.

Figura 88. Captura de pantalla de Unity mostrando el comentario de Instagram como línea de diálogo. Elaboración propia.

6. Conclusiones

Atendiendo a los objetivos generales planteados al principio de la memoria, cabe destacar que ambos se han podido lograr con éxito. Por un lado, el equipo fue capaz de realizar una segunda demo jugable con características y funcionalidad que han permitido aportar más valor al producto. La distribución de esta demo ha permitido al público objetivo participar en el proceso de evaluación de la idea gracias a los comentarios y reacciones que han compartido en las encuestas y en redes sociales. Estas respuestas han resultado ser de una gran utilidad para conocer que existe interés en la idea y estética, así como en su historia, sin embargo, las mecánicas y objetivos deberían ser revisados para mejorar la experiencia de los jugadores. El desarrollo de este producto ha permitido, sobre todo, el trabajo en equipo y la interacción con alumnos de distintas disciplinas, los cuales han aportado ideas y soluciones necesarias para completar el proyecto. La divergencia en el conocimiento de las herramientas de desarrollo también ha posibilitado la ayuda mutua y el descubrimiento de nuevas formas de trabajar para facilitar el proceso de desarrollo.

Por otro lado, con respecto a las extensiones relacionadas con los diálogos, se ha podido comprobar durante el desarrollo del capítulo 4 que se ha expandido la funcionalidad inicial para cumplir con los objetivos propuestos. En primer lugar, tras reestructurar el trabajo realizado en el primer MVP se crearon subsistemas para permitir el habla de los personajes siguiendo la forma de síntesis de texto de juegos referentes, con el objetivo de simular su tono. En segundo lugar, la configuración de estilos también ha sido ampliada para soportar efectos estáticos y dinámicos, tanto para el texto como para la voz, y se establecieron relaciones entre configuraciones para simplificar y abstraer la creación de estilos. La combinación de estas opciones de estilo facilitó la creación de perfiles globales que imiten emociones o tonos de conversación para dotar a los personajes de mayor expresividad. En tercer lugar, se ha dado soporte a las opciones de diálogos presentes en las herramientas, creando una infraestructura para incluir sistemas propios que permitan mostrar estas opciones e interactuar con ellas de formas distintas. En cuarto lugar, la inclusión de sistemas secundarios permite ahora el acceso a fuentes externas para ofrecer contenido dinámico en las conversaciones, esto ha posibilitado integrarse con los recursos de Instagram para ofrecer una conexión dentro del juego con la red social y así ayudar en la popularización del proyecto con sus potenciales usuarios. Adicionalmente, la internacionalización del texto también ha sido conseguida gracias a las capacidades existentes de las herramientas de diálogo, las cuales se han integrado con la configuración de idioma para el resto del juego.

Finalmente, todos estos sistemas coexisten gracias a un gestor de diálogos, el cual posibilita la comunicación del texto y los estilos para poder ser mostrado y hablado según las implementaciones concretas deseadas. Dada la intercambiabilidad de los distintos sistemas, este trabajo se podrá reutilizar en proyectos futuros del equipo o de otros desarrolladores interesados.

7. Trabajo futuro

Al finalizar el segundo MVP, el producto se puede considerar una prueba de concepto pulida, ya que consta del primer nivel completo del juego, y ha cumplido con el objetivo de atraer interés en el proyecto y la idea original. Es por ello que hay potencial para seguir con el desarrollo para distribuir la experiencia completa, así como para corregir problemas y bugs que puedan surgir gracias a la interacción continua con los usuarios.

En cuanto al desarrollo de diálogos, ha habido una serie de funcionalidades previstas que fueron descartadas y habrían supuesto cambios en el gestor de diálogos actual: es el caso de la implementación de puntos de control y guardado, el cual habría supuesto crear una gestión de estados de diálogos (que debiera recoger el guardado de los nodos visitados y variables establecidas, entre otros) para posibilitar su exportación e importación. Por otro lado, los estilos en diálogos están estrechamente relacionados con las necesidades de este proyecto, pero, ya que se puede proveer de un paquete independiente, se podría explorar una variedad de estilos más amplia u ofrecer un sistema para configurar las opciones dependiendo del proyecto con el objetivo de ser personalizable para cada tipo de juego. En el campo de la integración con sistemas externos y dinámicos (los aquí denominados sistemas secundarios), en concreto la conexión con redes sociales, surgieron ideas para permitir el flujo de datos en el sentido contrario, tal que la cuenta del jugador o del juego sea capaz de reaccionar a los sucesos que el jugador desbloquea durante su partida: p. ej. automatizar la creación de publicaciones cuando un número determinado de personas terminan el juego o cumplen un objetivo, o permitir el envío de mensajes directos a la cuenta del jugador con consejos o acertijos para completar un nivel.

Bibliografía

[1] AEVI, «La industria del videojuego en España,» 2019.

[2] Unity, «Unity real-time development platform | 3D, 2D VR & AR visualizations,» 2020. [En línea]. Available: https://unity.com/.

[3] A. J. Carrasco, «FROZEN OUT (I): DISEÑO Y DESAROLLO ARTÍSTICO DE UN VIDEOJUEGO CRÍTICO EN 3D,» 2020.

[4] A. R. Sáez, «FROZEN OUT (II): Diseño de interacción y animación de caracteres de un prototipo de videojuego crítico 3D,» 2020.

[5] T. R. Martín, «Videojuego “Frozen Out”. Programación, desarrollo e integración de mecánicas,» 2020.

[6] P. L. Orrios, «Desarrollo de un videojuego con Unity: implementación de la inteligencia artificial y funciones back-end,» 2020.

[7] A. Jiménez Carrasco, Frozen Out Pitch Doc, 2019.

[8] Newzoo, «Top 10 Countries/Markets by Game Revenues,» 2020. [En línea]. Available: https://newzoo.com/insights/rankings/top-10-countries-by-game-revenues/.

[9] AppInChina, «Content Restrictions and Requirements For Publishing Games In China (2020),» 2020. [En línea]. Available: https://www.appinchina.co/services/game-publishing/content-restrictions-for-publishing-games-in-china/.

[10] D. Pearson, «Steam controls 50-70% of PC download market,» 14 febrero 2011. [En línea]. Available: https://www.gamesindustry.biz/articles/2011-02-14-forbes-steam-controls-50-70-percent-of-pc-download-market.

[11] R. Valentine, «Valve adds revenue share tiers for developers,» 1 diciembre 2018. [En línea]. Available: https://www.gamesindustry.biz/articles/2018-12-01-valve-adds-revenue-share-tiers-for-developers.

[12] itch.io, «Open revenue sharing,» 2020. [En línea]. Available: https://itch.io/docs/general/about.

[13] Steam, «Joining The Steamworks Distribution Program,» 2020. [En línea]. Available: https://partner.steamgames.com/steamdirect.

[14] Secret Lab, «Yarn Spinner,» 2020. [En línea]. Available: https://yarnspinner.dev/.

[15] G. M. Smith, «Computer Games Have Words, Too: Dialogue Conventions in Final Fantasy VII.,» diciembre 2002. [En línea]. Available: http://www.gamestudies.org/0202/smith/.

[16] The Digital Game Museum, «The Rise and Fall of Adventure Games,» 2011. [En línea]. Available: https://www.digitalgamemuseum.org/the-rise-and-fall-of-adventure-games/.

[17] S. Domsch, «Dialogue in video games,» 2017. [En línea]. Available: https://www.academia.edu/37187222/Dialogue_in_video_games.

[18] J. Stoeber, «Animal Crossing's fake language sounds different in Japanese,» 22 marzo 2020. [En línea]. Available: https://www.polygon.com/videos/2020/3/22/21188355/animal-crossing-new-horizons-language-video.

[19] K. Neil, «Twine 2 vs Ink vs Yarn Unity integration,» 8 mayo 2018. [En línea]. Available: https://medium.com/@haikus_by_KN/twine-2-vs-ink-a-quick-and-dirty-unity-integration-comparison-99fe1e4549d.

[20] Unity, «Rich Text | Unity UI,» 2020. [En línea]. Available: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/StyledText.html.

[21] T. Bye, «Get started with speech synthesis,» 2020. [En línea]. Available: https://docs.microsoft.com/es-es/azure/cognitive-services/speech-service/quickstarts/text-to-speech.

[22] R. Hurey y B. Mouncer, «Quickstart: Synthesize speech in Unity using the Speech SDK for Unity,» 3 diciembre 2019. [En línea]. Available: https://github.com/Azure-Samples/cognitive-services-speech-sdk/tree/master/quickstart/csharp/unity/text-to-speech.

[23] Facebook, «Instagram Platform API,» 2020. [En línea]. Available: https://developers.facebook.com/docs/instagram.

[24] J. Meech, «Breaking Down Instagram Creator vs. Business Profiles,» 16 octubre 2019. [En línea]. Available: https://blog.dashhudson.com/instagram-creator-profile/.

Índice de figuras

Figura 1. Imagen promocional de A Short Hike. A Short Hike (2019). 13
Figura 2. Imagen promocional de Summer in Mara. Summer in Mara (2020). 13
Figura 3. Imagen promocional de A Hat in Time. A Hat in Time (2017). 14
Figura 4. Imagen promocional de Spyro Reignited Trilogy. Spyro Reignited Trilogy (2019). 14
Figura 5. Imagen promocional de Crash Bandicoot 4. Crash Bandicoot 4: It's About Time (2020). 15
Figura 6. Imagen promocional de Yooka-Laylee. Yooka-Laylee (2017). 15
Figura 7. Imagen promocional de Night in the Woods. Night in the Woods (2017). 16
Figura 8. Gráfica sobre el resultado dividido en trimestres. Elaboración propia. 22
Figura 9. Mapa de características. Elaboración propia. 28
Figura 10. Logotipos de las tecnologías utilizadas. De izquierda a derecha y de arriba a abajo: Unity, Visual Studio Community (2019), Git, GitHub, Trello, Yarn Spinner y RestSharp. 29
Figura 11. Mapa de características seleccionadas para el primer MVP (marcadas en azul). Elaboración propia. 32
Figura 12. Arte conceptual para el nivel exterior. Elaboración propia. 32
Figura 13. Mockup del mapa para el primer MVP con la distribución de enemigos, patrullas y puntos clave para el jugador. Elaboración propia. 33
Figura 14. Primer mockup de indicadores de diálogo y presentación de diálogo con bocadillos. Elaboración propia. 34
Figura 15. Segundo mockup de indicadores de diálogo y presentación de diálogo en una caja de texto. Elaboración propia. 34
Figura 16. Menú principal durante el primer MVP. Elaboración propia. 34
Figura 17. Menú de pausa del juego durante el primer MVP. Elaboración propia. 35
Figura 18. Encuesta primer MVP, resultados pregunta 1. Elaboración propia. 35
Figura 19. Encuesta primer MVP, resultados pregunta 2. Elaboración propia. 36
Figura 20. Encuesta primer MVP, resultados pregunta 3. Elaboración propia. 36
Figura 21. Encuesta primer MVP, resultados pregunta 4. Elaboración propia. 36
Figura 22. Encuesta primer MVP, resultados pregunta 5. Elaboración propia. 37
Figura 23. Encuesta primer MVP, resultados pregunta 6. Elaboración propia. 37
Figura 24. Encuesta primer MVP, resultados pregunta 7. Elaboración propia. 37
Figura 25. Encuesta primer MVP, resultados pregunta 8. Elaboración propia. 38
Figura 26. Mapa de características seleccionadas para el segundo MVP (en azul las nuevas características; en rojo las características a modificar o arreglar en base a los comentarios recibidos; en gris las características finalizadas en el MVP anterior). Elaboración propia. 39
Figura 27. Mockup de la versión del mapa para el segundo MVP. Elaboración propia. 40
Figura 28. Zonas del mapa del segundo MVP vistas desde arriba, incluyendo el páramo (zona marrón). Elaboración propia. 40
Figura 29. Mockup de la interfaz para el inventario. Elaboración propia. 41
Figura 30. Menú principal de Frozen Out tras su rediseño en el segundo MVP. Elaboración propia. 42
Figura 31. Encuesta segundo MVP, resultados pregunta 1. Elaboración propia. 42
Figura 32. Encuesta segundo MVP, resultados pregunta 2. Elaboración propia. 43
Figura 33. Encuesta segundo MVP, resultados pregunta 3. Elaboración propia. 43
Figura 34. Encuesta segundo MVP, resultados pregunta 4. Elaboración propia. 43
Figura 35. Encuesta segundo MVP, resultados pregunta 5. Elaboración propia. 44
Figura 36. Encuesta segundo MVP, resultados pregunta 6. Elaboración propia. 44

Figura 37. Encuesta segundo MVP, resultados pregunta 7. Elaboración propia. 44
Figura 38. Encuesta segundo MVP, resultados pregunta 8. Elaboración propia. 45
Figura 39. Encuesta segundo MVP, resultados pregunta 9. Elaboración propia. 45
Figura 40. Encuesta segundo MVP, resultados pregunta 10. Elaboración propia. 45
Figura 41. Equipo inicial de desarrollo de Frozen Out durante la feria de diciembre. De izquierda a derecha: Vicent Pla Madrid, Adrián Reina Sáez, Alejandro Jiménez Carrasco, Alejandro Gómez Noe, Tomás Ruiz Martín y Pablo López Orrios. 47
Figura 42. Captura de pantalla de la aplicación Twine con un archivo de muestra abierto. Elaboración propia. 54
Figura 43. Captura de pantalla de la aplicación web Yarn Spinner Editor con un archivo de muestra abierto. Elaboración propia. 55
Figura 44. Captura de pantalla de la aplicación Inky, el editor visual de Ink, con un archivo de muestra abierto. Elaboración propia. 55
Figura 45. Diagrama de clases UML mostrando las clases públicas de Yarn Spinner. Elaboración propia. 57
Figura 46. Capturas de pantalla de Unity ejemplificando la problemática del texto con etiquetas (arriba texto empezando sección con etiquetas visible, abajo texto con etiqueta finalizado y estilo en negrita activo). Elaboración propia. 59
Figura 47. Diagrama de clases UML con componentes relacionados con IDialogueText. Elaboración propia. 59
Figura 48. Diagrama de clases UML relacionado con la representación de etiquetas. Elaboración propia. 60
Figura 49. Desglose de líneas de diálogo en componentes de IDialogueText generado por el método AnalyseText. Elaboración propia. 61
Figura 50. Fragmento de guion para el nivel. Elaboración propia. 61
Figura 51. Diagrama de clases UML de DialogueSystem y las subclases MainDialogueSystem y SecondaryDialogueSystem. Elaboración propia. 63
Figura 52. Diagrama de clases UML representando la estructura abstracta necesaria para implementar la funcionalidad del primer MVP. Elaboración propia. 64
Figura 53. Diagrama de clases UML para relacionar YarnDialogueSystem con Yarn Spinner. Elaboración propia. 64
Figura 54. Diagrama de clases UML representando las clases abstractas y concretas necesarias para implementar la funcionalidad del primer MVP. Elaboración propia. 65
Figura 55. Diagrama de secuencia para representar el flujo de texto desde la invocación de los eventos de Yarn Spinner hasta su presentación en pantalla. Elaboración propia. 66
Figura 56. Diagrama de clases UML para representar la adición de VoiceManager (resaltado en amarillo) a la estructura. Elaboración propia..... 69
Figura 57. Captura de pantalla de Unity donde se encuentra el inspector seleccionando el script AnimaleseVoiceManager en la escena (izquierda) y los archivos de audio vistos desde Unity (derecha). Elaboración propia..... 70
Figura 58. Captura de pantalla del portal de Azure listando los recursos creados (arriba) y las claves de conexión para los servicios cognitivos agregados (abajo). Elaboración propia. 71
Figura 59. Diagrama de clases UML para representar las clases concretas de VoiceManager. Elaboración propia. 72
Figura 60. Fragmento de código encargado de obtener el valor interpolado dado un valor relativo, un valor por defecto, un valor mínimo y un valor máximo. Elaboración propia. 73
Figura 61. Diagrama de clases UML representando la estructura tras la adición del soporte para estilo (resaltando en amarillo los cambios). Elaboración propia. 73

Figura 62. Captura de pantalla de Unity con el inspector seleccionando el script BoxTextManager en escena. Elaboración propia. 74
Figura 63. Captura de pantalla de TextStyle incorporado a Frozen Out demostrando la creación de líneas, palabras y letras para presentar la línea de texto. Elaboración propia. 75
Figura 64. Captura de pantalla de Unity con la configuración de estilo de AnimaleseVoiceManager. Elaboración propia. 76
Figura 65. Captura de pantalla de Unity con el inspector seleccionando el AudioMixer con la configuración necesaria para generar el efecto de radio. Elaboración propia. 76
Figura 66. Diagrama de clases UML relacionado con el estilo de diálogo. Elaboración propia. 77
Figura 67. Diagrama de clases UML representando los añadidos para permitir el guardado de estilos (resaltando en amarillo los cambios). Elaboración propia. 78
Figura 68. Captura de pantalla de juego haciendo uso de ClosestPromptController y DialogueActer para activar el indicador del personaje con diálogo más cercano. Elaboración propia. 79
Figura 69. Diagrama de clases UML con añadidos para soportar la detección y selección del candidato para empezar la conversación (resaltando en amarillo los cambios). Elaboración propia. 80
Figura 70. Diagrama de clases UML con los añadidos para accionar y controlar la interacción del diálogo (resaltando en amarillo los cambios). Elaboración propia. 81
Figura 71. Diagrama de clases UML con las adiciones relacionadas con el control de opciones de diálogo (resaltando en amarillo los cambios). Elaboración propia. 83
Figura 72. Captura de pantalla con un fragmento de guion en castellano en el archivo de texto base (arriba) y su fragmento de guion traducido al inglés en archivo CSV (abajo). Elaboración propia. 83
Figura 73. Captura de pantalla del portal para desarrolladores de Facebook listando las aplicaciones creados (izquierda) y los productos asociados a esta aplicación (derecha) entre las que se encuentra Instagram Graph API. Elaboración propia. 85
Figura 74. Diagrama de clases UML representando las clases relacionadas con la conexión con Instagram. Elaboración propia. 86
Figura 75. Fragmento de código perteneciente al método GetComments de la clase InstagramRepository. Elaboración propia. 87
Figura 76. Captura de pantalla de Unity mostrando el prefab de diálogos: su jerarquía (izquierda) y el detalle del inspector del objeto DialogueManager (derecha). Elaboración propia. 89
Figura 77. Captura de pantalla de Unity mostrando los componentes de diálogo de un objeto de personaje. Elaboración propia. 89
Figura 78. Fragmento de guion en formato de Yarn Spinner para ejemplificar el uso de estilos globales. Elaboración propia. 90
Figura 79. Captura de pantalla del inspector de Unity mostrando la configuración de estilos globales de «Panico» y «Feliz». Elaboración propia. 90
Figura 80. Captura de pantalla de Unity ejemplificando la acción de los estilos globales descritos: «Panico» (arriba) y «Feliz» durante efecto dinámico de salto (abajo). Elaboración propia. 90
Figura 81. Captura de pantalla del inspector de Unity mostrando la configuración de estilo «Auriculares». Elaboración propia. 91
Figura 82. Captura de pantalla de Unity ejemplificando el uso del estilo «Auriculares» (izquierda) y el estado del mezclador de sonido de Windows durante ese momento (derecha). Elaboración propia. 91

Figura 83. Fragmento de guion en formato de Yarn Spinner para ejemplificar el uso de opciones. Elaboración propia. 92
Figura 84. Captura de pantalla de Unity mostrando un evento de opciones en forma de botones. Elaboración propia. 92
Figura 85. Captura de pantalla de Unity mostrando un evento de opciones a través de una campo de texto. Elaboración propia. 92
Figura 86. Captura de pantalla de la última publicación y comentarios de la cuenta del juego en Instagram. Elaboración propia. 93
Figura 87. Fragmento de guion en formato de Yarn Spinner ejemplificando la llamada a Instagram y la gestión de posibles errores. Elaboración propia 93
Figura 88. Captura de pantalla de Unity mostrando el comentario de Instagram como línea de diálogo. Elaboración propia. 93
Figura 89. Captura de pantalla de juego en la sección inicial. Elaboración propia 102
Figura 90. Captura de pantalla de juego durante conversación con Palanquilla. Elaboración propia. 102
Figura 91. Captura de pantalla de juego mostrando a Pol en el pedal en funcionamiento. Elaboración propia. 103
Figura 92. Captura de pantalla de juego durante la conversación con los cortes de helado. Elaboración propia. 103
Figura 93. Captura de pantalla de juego durante la cinemática de los conos. Elaboración propia. 104
Figura 94. Captura de pantalla de juego mostrando el montón de nieve encima del cono de helado. Elaboración propia. 104
Figura 95. Captura de pantalla de juego mostrando la caja de hielo vacía y el cono enterrado tras volcar el montón de nieve. Elaboración propia. 105
Figura 96. Captura de pantalla de juego de la sección de la batidora. Elaboración propia 105
Figura 97. Captura de pantalla de juego durante la cinemática de la bola de nieve. Elaboración propia. 106
Figura 98. Captura de pantalla de juego después de haber lanzado la bola y habiendo vaciado las cajas de hielo. Elaboración propia. 106
Figura 99. Captura de pantalla de juego durante la cinemática final. Elaboración propia 106
Figura 100. Diagrama de clases UML representando el conjunto de clases y paquetes relacionados con la implementación de los diálogos. Elaboración propia 107

Índice de tablas

Tabla 1. Tabla comparativa de juegos competidores con la propuesta a desarrollar. Elaboración propia. 17
Tabla 2. Matriz DAFO para el proyecto Frozen Out y sus componentes. Elaboración propia. 19
Tabla 3. Proyección económica del proyecto a tres años. Elaboración propia 21
Tabla 4. Lean Canvas realizado para el proyecto. Elaboración propia 25
Tabla 5. Tabla comparativa de sistemas de texto. Elaboración propia. 56
Tabla 6. Tabla comparativa de sistemas de voz. Elaboración propia. 68
Tabla 7. Variantes de DialogueActer atendiendo al grado de bloqueo y automatización. Elaboración propia. 80

Anexo I: Partida guiada de segundo MVP de Frozen Out

Al principio de la partida el personaje principal, Pol, se encuentra en un extremo de la mina (Figura 89). Delante suya tiene un ítem, el pico, el cual puede recoger para que se añada a su inventario. En la primera sección del juego existe un personaje, un polo llamado Alfredo, con el cual puede hablar y le recordará que no puede merodear por la mina, ya que debería estar trabajando, de lo contrario los guardias (representados como conos de helado) le podrían arrestar. Esto indica al jugador que debe esconderse de los guardias. Hay tres personajes más en esta sección, un cono de helado y dos polos a los que está vigilando, razón por la que no se puede interactuar con ellos. Cabe destacar también el carro con hielo que está custodiando el cono, ya que será útil en un futuro.

Figura 89. Captura de pantalla de juego en la sección inicial. Elaboración propia.

Moviendo el personaje a la sección más a la derecha, nos encontramos con un cono que está rondando por la zona. También hay un polo, Palanquilla, que informa al jugador de que ha perdido una palanca y no sabe dónde está (Figura 90). Por suerte, en esta zona se encuentra dicha palanca, la cual se puede recoger para devolvérsela a Palanquilla. Una vez devuelta, este activará el pedal que tiene detrás y posibilitará a Pol alcanzar las plataformas superiores (Figura 91).

Figura 90. Captura de pantalla de juego durante conversación con Palanquilla. Elaboración propia.

Figura 91. Captura de pantalla de juego mostrando a Pol en el pedal en funcionamiento. Elaboración propia.

En primer lugar, en la plataforma superior derecha nos encontramos con dos cortes de helado, los cuales custodian la salida de la mina y si se entra en conversación con ellos revelarán que Pol necesita 100 trozos de hielo para dejarle pasar (Figura 92). Salir de la mina es el objetivo final del nivel. Es también en esta parte donde se puede activar una cinemática donde se ven dos conos guardando dos carros llenos de hielo (Figura 93).

Figura 92. Captura de pantalla de juego durante la conversación con los cortes de helado. Elaboración propia.

Figura 93. Captura de pantalla de juego durante la cinemática de los conos. Elaboración propia.

Volviendo a subir al pedal y dirigiéndose a la plataforma superior de la izquierda se puede observar como una montaña de nieve se encuentra convenientemente encima del guardia del principio (Figura 94). Si se llega al extremo de esta plataforma se puede conseguir el ítem de la cuchara. Equipándose este ítem se puede actuar con la montaña de nieve para enterrar el cono. Ahora se puede bajar para obtener el hielo que estaba custodiando (Figura 95), de modo que se tendrán 10 trozos de hielo, así como hablar con los polos que estaban siendo vigilados.

Figura 94. Captura de pantalla de juego mostrando el montón de nieve encima del cono de helado. Elaboración propia.

Figura 95. Captura de pantalla de juego mostrando la caja de hielo vacía y el cono enterrado tras volcar el montón de nieve. Elaboración propia.

El siguiente objetivo está en el otro extremo de la mina, la cual se puede acceder a través de la plataforma con los cortes de helado y bajando a la sección de la derecha. Aquí se encuentra una batidora (Figura 96), que se puede hacer uso para subir a la plataforma superior de la derecha. En esta sección hay un polo enterrado en la nieve, el cual se puede liberar usando nuevamente la cuchara. Este polo agradecerá a Pol su acción y le permitirá jugar con sus bolas de nieve. Interactuando con la caja de bolas de nieve que hay al lado se puede extraer una, la cual se puede hacer rodar y manejar hacia el borde inferior para lanzarla colina abajo (Figura 97). Esta bola ahuyentara a los dos conos que estaban custodiando las cajas de hielo. De modo que ahora se pueden obtener para tener un total de 100 trozos de hielo (Figura 98). Cabe destacar que con el pico equipado se pueden picar los 4 riscos de hielo esparcidos por el nivel para conseguir también trozos de hielo, sin embargo, solo se pueden obtener 4, 5, 6 y 7 trozos adicionales, respectivamente.

Figura 96. Captura de pantalla de juego de la sección de la batidora. Elaboración propia.

Figura 97. Captura de pantalla de juego durante la cinemática de la bola de nieve. Elaboración propia.

Figura 98. Captura de pantalla de juego después de haber lanzado la bola y habiendo vaciado las cajas de hielo. Elaboración propia.

Finalmente, se puede dirigir a los cortes de helado para que se aparten y así permitir a Pol salir de la mina (Figura 99). Con esto concluye la demo y se muestran los créditos al jugador.

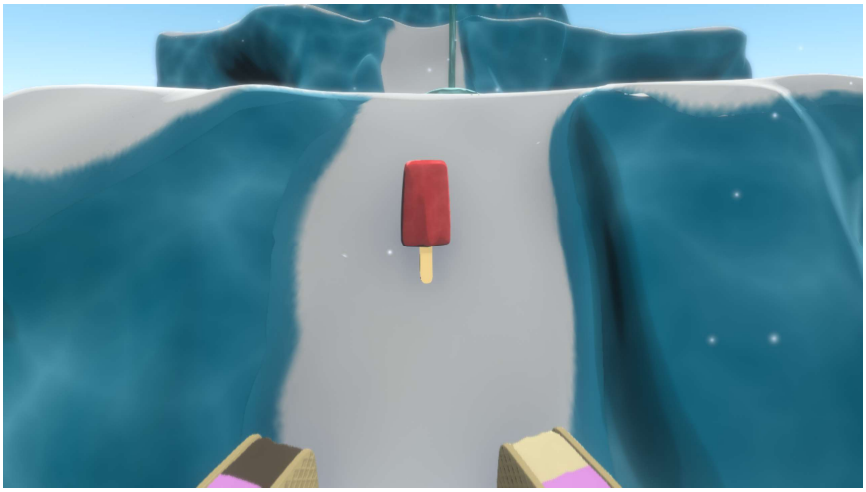

Figura 99. Captura de pantalla de juego durante la cinemática final. Elaboración propia.

Anexo II: Diagrama de diálogos completo

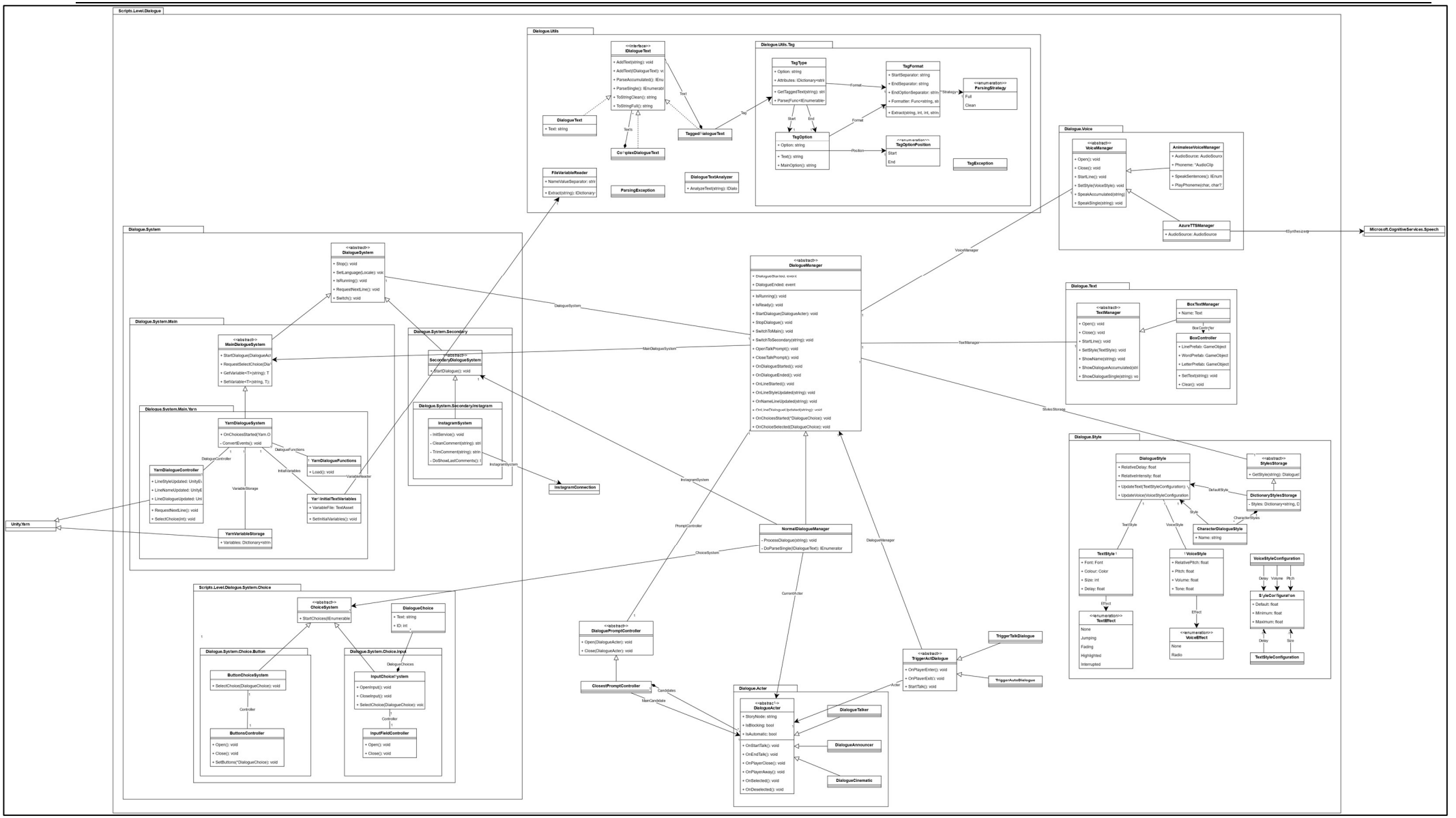

Figura 100. Diagrama de clases UML representando el conjunto de clases y paquetes relacionados con la implementación de los diálogos. Elaboración propia.