Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Migración de juego educativo a Unity3D

Autor:
Gonzalo Fernández González

Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Migración de juego educativo a Unity3D

Autor:
Gonzalo Fernández González

Tutores:
Mario Corrales Astorgano
Cristian Tejedor García

A mi familia y amigos que me apoyaron desde el primer momento en esta nueva etapa

Agradecimientos

Muchas personas me han ayudado y apoyado con este proyecto. Primeramente, me gustaría dar las gracias a mis tutores del TFG, tanto Mario Corrales Astorgano como Cristian Tejedor García, los cuales han sido fundamentales en el desarrollo. Dándome la oportunidad en un primer momento de poder realizar dicho proyecto con ellos y ayudándome posteriormente con todo el desarrollo y revisiones para poder finalizarlo de la mejor manera posible. También agradecer a toda mi familia y amigos, los cuales me han apoyado día a día y me han animado durante todo el desarrollo del trabajo.

Resumen

Los proyectos de software desarrollados con Flash se eliminaron por completo y quedaron obsoletos de todos los navegadores, debido al fin del soporte de Flash a finales de 2020. Un videojuego desarrollado en el año 2015 en el TFM de Mario Corrales, llamado Pradia, el cual está dirigido a ayudar a personas con Síndrome de Down a mejorar sus capacidades lingüísticas, quedó inoperativo. Mediante el uso de dinámicas introducidas en la narrativa del juego su objetivo es que el jugador no perciba la herramienta como una sucesión de actividades de aprendizaje, sino que aprenda y mejore mientras juega. Por ello, a lo largo del presente TFG, se ha migrado la aplicación de Pradia a un motor gráfico más actual, como es Unity, manteniendo la idea de construir el juego con dinámicas para favorecer el aprendizaje, así como una interfaz de usuario cuidada, ya que las personas con Síndrome de Down tienen más dificultades a la hora de interactuar con aplicaciones informáticas, lo que dificulta la integración de las tecnologías de la información.

Abstract

Software projects developed with Flash have been completely removed and made obsolete from all browsers, caused by the end of support of Flash by the end of 2020. One of these applications was Pradia, a video game developed in 2015 in Mario Corrales Astorgano master's thesis project, which is aimed at helping people with Down Syndrome to improve their linguistic abilities. Through the use of dynamics introduced in the narrative of the game, its goal is that the player does not perceive the tool as a succession of learning activities, but instead learns and improves while playing. Therefore, throughout this bachelor's thesis, the Pradia application has been migrated to a more current graphics engine, Unity. This has been done by keeping the idea of building the game with dynamics to favor learning, as well as a detailed user interface, since people with Down Syndrome have more difficulties when interacting with computer applications, making it difficult to integrate information technology.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

En el año 2015 surgió la oportunidad de desarrollar Pradia [1], cuyo autor es Mario Corrales Astorgano. Pradia es un videojuego en 2D desarrollado en Flash [2] y dirigido a las personas con síndrome de Down. La idea principal de este proyecto era acercar a estar personas a situaciones cotidianas de la vida para que pudieran practicar y mejorar sus intervenciones con otras personas, especialmente en lo relacionado con la comunicación hablada. Por ello surgió la posibilidad de desarrollar un juego con dinámicas de reproducción y grabación de audio, para poder ayudarles a mejorar e integrarse en cualquier tipo de situación. Mediante la supervisión de otra persona, se validaría de manera subjetiva si los ficheros de audio grabados por los usuarios eran correctos o no, mientras que el resto de las situaciones de elección serían validadas por el propio juego.

Este proyecto surge de la propuesta de Trabajo Fin de Grado (TFG) publicada en la web de ofertas de TFG de la Escuela de Ingeniería Informática de Valladolid por Mario Corrales y Cristian Tejedor. El objetivo principal de esta propuesta no es el de continuar con el desarrollo del código de Pradia para mejorarlo, sino migrarlo a un motor gráfico más actual como es Unity [3], ya que el anterior estaba obsoleto y anticuado, pero manteniendo la esencia buscada en el proyecto anterior.

La propuesta inicial de este TFG es la de conseguir generalizar el proyecto de Pradia, originalmente desarrollado en Flash, para que cualquier desarrollador sin referencia pueda crear nuevos niveles de juego simplemente arrastrando los elementos que desee en el panel visual de Unity. De esta forma, se pretende ahorrar innumerables horas de trabajo habiendo generalizado las dinámicas de interacción con el usuario. En cuanto a la otra premisa principal, cabe destacar la importancia de la interacción con el usuario mediante la reproducción de sonidos y la grabación de estos. Particularmente, se busca mantener el sistema de Pradia para realizar el juego mediante la supervisión de otra persona que sea la que valide si las frases han sido correctamente dichas o no por el usuario, ya que es más difícil que el reconocedor de habla actúe de forma correcta en estos casos. Finalmente, se desea integrar tecnología

del habla como reconocimiento automático y síntesis de voz para futuros niveles de juego.

1.2. Motivación

Los cambios constantes en la tecnología de desarrollo de software de aplicaciones o juegos, implican un mantenimiento de los proyectos desarrollados. En el caso de la aplicación de Pradia, las restricciones y cambios del motor gráfico con el que ha sido desarrollado durante los últimos años, ha implicado una reducción de la posibilidad de refactorización y expansión del proyecto con visiones futuras. Flash [4], el motor gráfico usado para el desarrollo, se ha visto envuelto en diversos inconvenientes que han motivado el cambio hacia otros motores. Entre los motivos de su deterioro cabe destacar las vulnerabilidades [5] encontradas en repetidas ocasiones, las cuales podrían haber sido utilizadas por criminales con fines maliciosos. También, el futuro poco prometedor de Flash se ha visto involucrado por la competencia emergente y a la poca evolución que han mostrado por el motor, dejando de lado el desarrollo de este. Además, el problema con la exportabilidad ha terminado deteriorándolo por completo, debido a los problemas con productos Apple o con dispositivos móviles.

Los objetivos principales marcados en el desarrollo de Pradia, motivan la realización de este proyecto, debido principalmente a los experimentos de investigación que han sido realizados por el grupo ECA-SIMM. En cuanto a los resultados obtenidos del anterior proyecto, podemos encontrar dos artículos en los que se reportan los resultados obtenidos con Pradia [6] [7], entre los que destacamos los siguientes:

- Conseguir que el juego mantenga a los usuarios atentos y sin distracciones.
- Una recepción positiva por parte de los logopedas asistentes.
- Validez de las muestras de voz recogidas para futuros análisis en profundidad.

1.3. Objetivos

El objetivo general del proyecto es migrar parte de las funcionalidades existentes en Pradia al motor gráfico Unity. Este objetivo principal se puede dividir en los siguientes objetivos particulares:

- Migrar las principales funcionalidades de Pradia a Unity estableciendo una base para una futura migración completa.
- Realizar la migración de la forma más generalista posible para poder ser reutilizada en otros juegos educativos que incluyan tecnologías del habla.
- Implementar la conexión con un reconocedor de voz para poder incluir dicha opción a la hora de interactuar con el usuario.

1.4. Metodología

Una vez comprobadas las características del proyecto y otros aspectos importantes como la estabilidad de los requisitos, los usuarios finales de producto, o el equipo de desarrollo, se ha optado por utilizar una metodología iterativa e incremental [8]. En este caso no es óptimo utilizar otro tipo de metodologías ágiles, ya que estas se aplican en grupos de trabajo con varios trabajadores, por lo que en este caso, al solo disponer de un integrante, no resulta relevante.

En cuanto a las ventajas que avalan el uso de este desarrollo se encuentran varias, como el hecho de que el cliente pueda obtener resultados importantes y usables ya desde las primeras iteraciones, por lo que se aprecia si el proyecto va por buen camino sin tener que esperar al final de este, como es propio de las metodologías en cascada. Por otra parte, se reduce el riego de implementar funcionalidades no requeridas por los clientes, también conocido como *gold-plating*. Por último, permite que el proyecto pueda ser abandonado temporalmente, lo que supone un beneficio clave teniendo en cuenta las circunstancias del estudiante.

En la Figura 1.1 podemos observar los pasos que se van a seguir durante un desarrollo iterativo incremental. Cabe remarcar, que en el capítulo 4 se va a hablar en profundidad de cada una de las acciones realizadas en las iteraciones del proyecto.

Figura 1.1: Esquema de la metodología [9]

1.5. Estructura de la memoria

La memoria actual está dividida en los siguientes apartados principales:

- **Introducción:** Se presenta el proyecto de una forma general, mediante el trato de su contexto, motivación, los objetivos visualizados y la metodología a seguir.

- **Estado de la cuestión:** Se describe el estado actual de la aplicación, el nuevo entorno y herramientas usadas, así como la comparación con otros proyectos similares.
- **Planificación:** Análisis, descripción y estimación de las iteraciones con sus tiempos de entrega, estimación del coste del proyecto y descripción de riesgos y planes de acción.
- **Descripción de las iteraciones:** Se describe el trabajo real realizado en el proyecto en cada iteración, desde el inicio hasta finalizarlo.
- **Estado final de la aplicación:** Se describe el análisis y diseño de la aplicación al completo, agrupando todos los diagramas y figuras realizados, así como los pasos necesarios para poder crear un nuevo nivel de juego.
- **Pruebas:** Descripción de las pruebas realizadas, así como de los test implementados para ello.
- **Conclusiones:** Se describen los conocimientos y conclusiones obtenidas tras la realización del proyecto, así como una lista de trabajos futuros que podrían realizarse para mejorar las funcionalidades que ofrece la aplicación.
- **Apéndices:** Distintos documentos adicionales como los acrónimos, un manual de despliegue completo sobre el despliegue, el manual de uso de Pradia, el detalle de la funcionalidad implementada, los datos recabados en el test de usabilidad y el contenido entregado en el TFG.
- **Bibliografía:** Referencias a páginas web, libros o proyectos consultados a lo largo del desarrollo del proyecto.

Capítulo 2

Estado de la cuestión

2.1. Pradia

Según se ha comentado en la sección 1.1, Pradia es un videojuego desarrollado en el grupo ECA-SIMM de la Universidad de Valladolid. Consiste en un juego en 2D con reproducción y grabación de audios para mejorar la capacidad del habla en personas con síndrome de Down o con problemas en el habla. El objetivo del juego es conseguir un aprendizaje más dinámico y llevadero, haciendo el proceso más entretenido mediante un videojuego con una historia detrás.

Este juego está pensado para ordenador, aunque ha sido exportado también a dispositivos Android. En cuanto a su funcionamiento presenta diferentes escenarios interactivos con el usuario. De esta forma, la persona que lo pruebe puede interactuar directamente con los diversos elementos que se presentan. En cuanto al apartado referente a la reproducción y grabación de audios, se divide por dinámicas, que son las que introducen los ejercicios de práctica del habla. En cada pantalla podemos encontrar diferentes acciones a realizar, mediante el uso del ratón del ordenador. En todo momento se guarda información acerca de la actividad de los usuarios con la aplicación mediante ficheros de log, ya sean los eventos producidos (como entrada a una dinámica o selección de un objeto del inventario), o las grabaciones realizadas. En las figuras 2.1 y 2.2 podemos ver la interfaz representativa de algunas pantallas del juego, ya sea en un escenario exterior o en medio de una de las dinámicas comentadas con anterioridad.

A continuación, vamos a proceder a explicar las funcionalidades principales que contenía Pradia, de las cuales algunas han sido introducidas de forma breve anteriormente:

- Login: Para acceder a la aplicación no se necesitan unas claves que serán comprobadas. En su defecto hay una serie de campos a rellenar para poder empezar el juego como tal. Primero de todo, se debe elegir el nombre del jugador que va a iniciar la partida, mediante un campo de texto dónde será introducido. En segundo lugar, se dispone de una serie de avatares seleccionables,

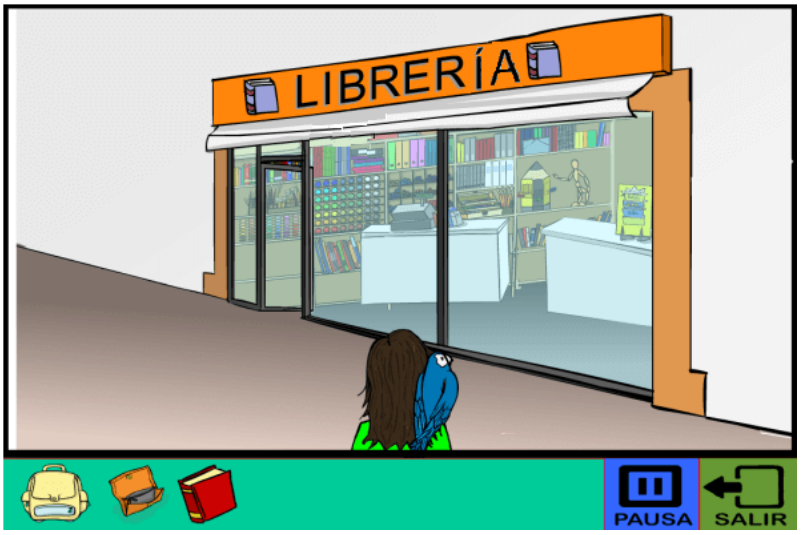

Figura 2.1: Interfaz exterior de un escenario del juego

Figura 2.2: Interfaz interior de un escenario del juego

por lo que se puede elegir la apariencia que se desee para el protagonista de la historia. Por último, resalta el campo para elegir dificultad. En este caso se compone de tres opciones, las cuales son básico, normal y avanzado.

- Modo historia: Para situarnos en contexto, este proyecto solo dispone de un modo individual. Esto se debe a que no se ha visto necesario realizar un modo multijugador que mejore excesivamente la experiencia del usuario. De este modo se dispone de un modo historia en el menú principal del juego, al pulsar sobre el botón Jugar. De esta forma, el usuario accede directamente al desarrollo de la misma para empezar con las aventuras.

- Episodios: Por otra parte, se ha visto un gran avance el hecho de poder acceder cuando se desee a cualquier escenario de la historia, sin necesidad de haber realizado los anteriores para haber podido llegar en un curso normal. Esta funcionalidad permite acceder a cualquier pantalla en cualquier momento, por lo que se puede realizar cualquier dinámica de juego. Esto supone una ventaja, ya que el terapeuta que acompaña a los jugadores puede hacer que estos repitan alguna

dinámica concreta sin la necesidad de repetir todos los escenarios. Así mismo, si todavía no se ha llegado a un escenario con una dinámica que nos resulte interesante, no hace falta pasarse el modo historia hasta ese punto, sino que se puede acceder directamente como se muestra en las imágenes siguientes.

Figura 2.3: Acceso a episodios

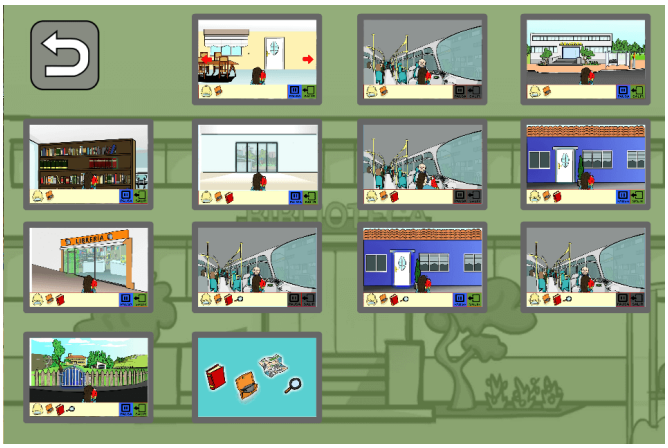

Figura 2.4: Acceso a episodios

Una vez comentadas las principales funcionalidades que presente el proyecto de Pradia, vamos a comentar un poco más detalladamente las dinámicas de este juego. Cabe remarcar, que para el desarrollo del juego referente a este TFG, se ha hecho énfasis en dos dinámicas del proyecto de Pradia, pese a que este juego dispone de otras orientadas más a añadir variedad al juego que al entrenamiento propiamente dicho. Las principales dinámicas son las siguientes:

- Dinámica de Producción: Estas dinámicas consisten primero de la reproducción de una frase la cual hay que repetir. Para ello se dispone de un panel con un micrófono para realizar la grabación. Si la frase es correcta y se repite de forma adecuada, se procede a avanzar, de lo contrario se tiene la opción de repetir hasta un número limitado de intentos. Para ello, la idea original de este juego es realizar estas dinámicas con un logopeda junto al usuario, el cual se encarga de validar la respuesta en función de si es correcta o no. Esto se debe a que, para los usuarios con síndrome de Down o problemas en el habla, resulta más complicado utilizar un reconocedor

de voz interprete correctamente la frase que han pronunciado. Esto no quita, como explicaremos más adelante, que para la nueva implementación del juego si se dispondrá de un reconocedor de voz. Por tanto y ante la falta del anterior mencionado, es una persona externa la que valida la frase, utilizando para ello el teclado del ordenador donde se reproduce el juego. El aspecto de esta dinámica es como el que podemos observar en la figura 2.5.

Figura 2.5: Ejemplo dinámica de producción

- Dinámica de Comprensión: El funcionamiento de esta dinámica se basa primero en escuchar una frase que hay que identificar. Una vez reproducido el audio, se muestran dos paneles con la misma frase, pero cambiando elementos como por ejemplo una en modo afirmativo y otra en modo interrogativo. El usuario tiene la capacidad de reproducir ambas opciones para ver cual se asemeja más a la frase que tiene que identificar. Cuando crea que tiene la solución, debe pulsar el botón del panel que crea correcto. En caso contrario dispone de varias opciones para lograrlo. El aspecto de esta dinámica es como el que se muestra en la figura 2.6.

Figura 2.6: Ejemplo dinámica de comprensión

2.2. Flash

Adobe Flash [4] es una herramienta para la creación de contenido multimedia e interactivo. En su momento de auge, destacaba como motor gráfico ya que permitía conectar componentes gráficos con programación técnica, es decir, un sistema con el mismo comportamiento que Unity, como veremos

más adelante. Por tanto, cabía la posibilidad de crear nuevos objetos gráficos y, al mismo tiempo, darle sentido mediante el código, el cual utiliza ActionScript. Antes de profundizar en las funcionalidades clave que dieron sentido a su uso, cabe destacar que Flash se usaba en su momento para todo tipo de proyectos, aunque destacaba por el uso para el desarrollo de juegos simples, sin muchas mecánicas. También, es evidente que es más adecuado para crear juegos en 2D, ya que requiere menos conocimiento sobre el lenguaje que la creación en 3D.

En el momento de desarrollo del videojuego de Pradia (2015), se optó por utilizar este motor por su popularidad en el momento, así como por las funcionalidades que permitían y facilitaban el desarrollo de este. Como puntos más importantes para el desarrollo, según hemos mencionado anteriormente, se permite la creación de objetos gráficos los cuáles pueden ser animados incluso sin código, aunque mediante el lenguaje ActionScript se pueden programar fácilmente. Otro de los factores claves para elegir Flash fue el motivo de la interacción, la cual es clave para este desarrollo, ya que es posible responder a las entradas de ratón y teclado, las cuales se usan, por ejemplo, para validar si una grabación ha sido realizada correctamente. Como otro punto importante, se puede resaltar que los elementos de audio se pueden añadir de forma rápida y fácil. Esto es primordial para el desarrollo, ya que el sentido del juego está enfocado en la reproducción y grabación de audios, por lo que una facilidad como esta para utilizar los audios es clave. Por último, cabe resaltar la posibilidad de introducir código dinámico gracias a ActionScript, lo que facilitó la creación del juego.

Por otra parte, el desarrollo de Pradia se desarrolló en su momento con Flash debido a su popularidad en la época, la cantidad de documentación que existía para poder desarrollar y solucionar problemas, así como la comunidad que la envolvía entre otras características. Con el paso del tiempo y la aparición de otros motores gráficos, se ha ido quedando atrasada en estos aspectos, por lo que fue necesario un cambio a Unity como explicaremos más adelante. Por este motivo, pasaremos a comentar las claves más importantes para el cambio mencionado.

- En primer lugar, Flash se ha encontrado con numerosas vulnerabilidades en repetidas ocasiones. En ocasiones, dichas vulnerabilidades son usadas por criminales, como por ejemplo, en febrero de 2013 [5], hubo informes de que atacantes podían tomar el control de un equipo si el usuario abría un archivo malicioso con el reproductor.
- En segundo lugar, el futuro de Flash ha resultado poco prometedor debido a la competencia y a la poca evolución mostrada. Desde 2012, Adobe no se ha mostrado interesado en el desarrollo posterior de Flash, dedicándose a la creación de Flash Next, otro motor gráfico que dejaría anticuado a este. El problema surge cuando se anunció que este nuevo desarrollo no sería compatible con las versiones anteriores de Flash, lo que significa una gran cantidad de trabajo para los desarrolladores y auguraba un futuro poco prometedor.
- En tercer lugar, se ha podido observar, que el personal dedicado a trabajar en el desarrollo de este motor se ha visto reducido con el paso del tiempo. Esto genera una menor atención por parte de los desarrolladores, así como una baja implicación, lo que produce que en la actualidad haya quedado en desuso.

- Por último, y uno de los factores más a tener en cuenta, es que el problema de la exportabilidad ha acabado deteriorando la vida de Flash de manera progresiva. Según explicaremos a continuación, tanto algunos sistemas operativos, como algunos dispositivos, no están diseñados para que este motor gráfico funcione correctamente en ellos. Primeramente, mencionaremos el conflicto con Apple, ya que se ha reportado que muchos dispositivos no son compatibles con Flash, lo cual resulta sorprendente debido a las relaciones comerciales entre ambas compañías. Dicho problema surge, por ejemplo, cuando insertas un archivo swf online, ya que se ha de tener en cuenta que muchos usuarios no podrán verlo, debido al uso de un producto de Apple. Seguidamente y en relación con lo comentado anteriormente, también surgen problemas con ciertos dispositivos, en particular con los móviles. Han llegado muchas críticas acerca del mal funcionamiento del mismo con el uso de pantallas táctiles. Esto ha supuesto que mucha gente viera que Flash no estaba dedicado a estos dispositivos, sino que se encontraba enfocado en ordenadores con el uso de ratón.

Debido a todos estos motivos explicados anteriormente, se ha buscado sustituir el uso de este motor gráfico por unos con mayor competencia, documentación o exportabilidad en la actualidad, ya que con el paso del tiempo, Flash, no ha brindado esta oportunidad como debía.

2.3. Ventajas y desventajas de Flash y Unity

Una vez vistos los motivos principales del deterioro de Flash, se han buscado soluciones ante este problema. Entre ellas, se ha encontrado Unity, el motor elegido y con diversas ventajas en frente del anterior. Para acercar el problema, haremos una pequeña comparación entre ambos para comparar las diferencias.

- En primer lugar, destaca el hecho de que Flash no sea gratuito o carezca de esa opción para los pequeños desarrolladores [10]. En cambio, Unity, aunque sea de pago para proyectos grandes, si dispone de una versión gratuita si los ingresos previstos para el desarrollo son menores de 100000€, lo cual es un motivo clave, sobre todo, a la hora de usarlo en los primeros proyectos de aprendizaje.
- En segundo lugar, Flash necesita una aplicación externa para ejecutar en escritorio (Adobe Air) [11]. Este, es un entorno de ejecución multiplataforma para la creación de aplicaciones, que utiliza Flash para usarse como aplicación de escritorio. Esto supone necesitar más herramientas, así como dedicar tiempo a su instalación y aprendizaje. En cambio, Unity, es un entorno multiplataforma y dispone de la posibilidad de ejecutar sus proyectos desde la misma aplicación de desarrollo, así como una previsualización de los mismos para observar directamente los cambios sin necesidad de exportar el juego.
- En tercer lugar, cabe destacar que Adobe, finalizó el soporte técnico para Flash a finales del 2020 [12], de esta forma ha sido eliminado por completo de todos los navegadores y ha quedado

obsoleto en todos ellos. A diferencia suya, Unity está en auge y actualmente está desarrollado para las últimas tres versiones principales de todos los navegadores [13].

- En cuarto lugar, como veremos en detalle más adelante, Flash utiliza ActionScript como lenguaje para generar los scripts. Este lenguaje posee varias desventajas respecto a los más demandados actualmente, ya que consta de un menor desarrollo y de pocas bibliotecas que permiten funcionalidades internas. En cambio, Unity, posee la opción de generar scripts mediante C#, un lenguaje conocido actualmente con innumerables bibliotecas, con soporte con otros lenguajes y una sintaxis sencilla entre otras ventajas.

Una vez comentadas las características técnicas que han motivado la migración de motor gráfico, también cabe mencionar el soporte y documentación que facilitan el uso de Unity y lo diferencian claramente de Flash, el cual se ha ido deteriorando con el tiempo. La forma de solucionar problemas emergentes en los proyectos es clave a la hora del desarrollo, y más todavía para desarrolladores inexpertos. Por ello, es necesario una documentación extensa y actualizada. Flash apenas cuenta con documentación actual existente, por lo que a la hora de intentar resolver un problema o documentarse sobre el mismo es más difícil hacerlo. En cambio, Unity cuenta con uno de los manuales más extensos de la actualidad, manteniendo las versiones más actualizadas y las anteriores para consultarlas en cualquier momento. Además, tener una comunidad amplia y resolutiva es otro punto importante que facilita nuestro desarrollo. En este aspecto, como en la mayoría de los vistos anteriormente, Flash se ha quedado obsoleto, contando con una comunidad mínima que se ha ido deteriorando con el paso del tiempo. Por su parte, Unity posee una amplia comunidad dispuesta a solucionar los problemas de una forma rápida y efectiva, ya que al ser tan inmensa, la mayoría de los problemas que nos puedan surgir ya han sido resueltos por otros usuarios.

Por último, en relación con los equipos de desarrollo pequeños, como en este caso, se busca facilitar la ampliación y desarrollo futuro del juego. De esta forma, está presente la motivación de generalizar las dinámicas y comportamientos de la aplicación, para que cualquier usuario sin casi experiencia previa, pueda seguir desarrollando niveles libremente. Por ello, se pretende que otros usuarios, puedan interactuar directamente con el panel de control que proporciona Unity directamente, arrastrando o introduciendo los elementos que necesiten en cada momento, sin necesidad de dedicar innumerables horas al apartado de scripting. De esta forma, se pueden añadir o reducir el número deseado de dinámicas a cualquier nivel, para poder ampliarlo, reducirlo o modificar su dificultad.

2.4. Entorno y herramientas tecnológicas

En esta sección vamos a exponer las tecnologías utilizadas para el desarrollo de este TFG, así como una descripción de cada una de ellas para profundizar más. Como breve introducción, cabe destacar el uso de Unity como motor gráfico para el desarrollo del videojuego, así como para la disposición de toda la interfaz y animaciones disponibles. Por otra parte, se ha utilizado Visual Studio como entorno de desarrollo para crear el código correspondiente. En cuanto al lenguaje elegido para ello, contamos con

C#, el cual ha servido para dar sentido a las animaciones y escenarios que han sido creados mediante el uso de scripts. Estos scripts, como veremos más detalladamente luego, se asignan a elementos y objetos del juego para darles el comportamiento que se desea. Para la gestión del proyecto, se ha confiado en el uso de Unity Hub, el cual permite gestionar de cualquier manera dichos proyectos. Como apoyo, se han utilizado otras tecnologías como Postman, para probar las llamadas a la API durante el desarrollo.

2.4.1. Unity

Unity [3] es un motor de videojuego multiplataforma, el cual dispone de una serie de rutinas de programación que permiten el diseño, la creación y el funcionamiento de un entorno interactivo (ver Figura 2.10). Este software centraliza todo lo necesario para la creación de juegos, ya sean en 2D o 3D. Su funcionamiento es simple, ya que permite crearlos para diversas plataformas, mediante un editor visual y una programación mediante scripting, obteniendo resultados profesionales. Por este motivo, pese a que sea clave para la iniciación de los desarrolladores novatos, también cabe remarcar que han surgido infinidad de juegos populares gracias al uso de esta herramienta. Por lo tanto, primero entraremos en detalles sobre su uso y funcionamiento, para más adelante comentar por que ha sido el motor que se ha decidido utilizar para el desarrollo del juego.

Para comenzar con la introducción de la herramienta, vamos a explicar sus rasgos generales que la caracterizan, así como su interfaz donde veremos la forma de trabajo directa, donde se permite interactuar con todos los elementos. Como se puede ver en la figura 2.7, la ventana de Unity 3D está formada por pequeñas ventanas individuales que se pueden reorganizar o agrupar en la ventana principal según se desee, permitiendo mostrar u ocultar opciones según nuestra forma de trabajo. Esto muestra que la apariencia del editor puede diferir de un proyecto a otro y de un desarrollador a otro.

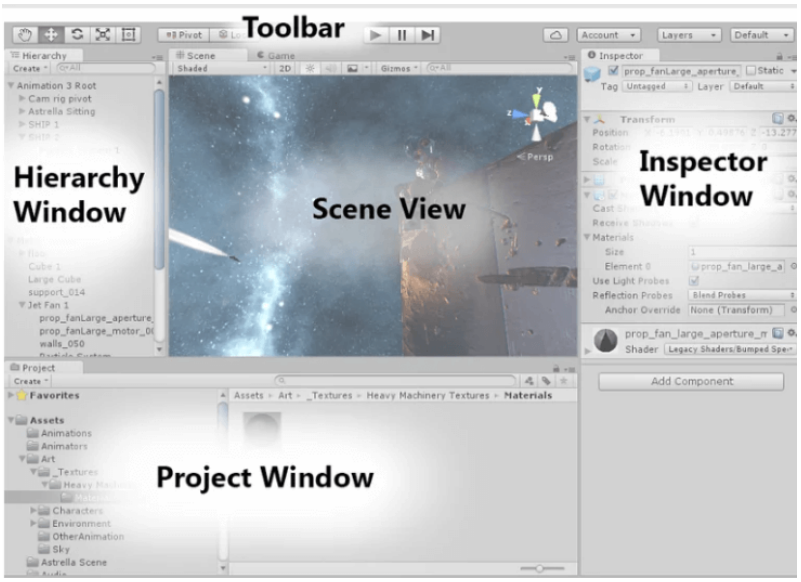

Figura 2.7: Distribución de la interfaz de Unity

Ahora vamos a proceder a explicar los principales elementos observados en la figura 2.7

- Project Window: Nos encontramos la biblioteca de elementos activos o Assets. Todos los elementos que se encuentran aquí están importados en el proyecto y por tanto están disponibles para su uso. Entre ellos, destacan los relacionados con animaciones, scripts o prefabs. Para poder usar de forma rápida los elementos que necesitamos con frecuencia, destaca un apartado de favoritos donde se pueden almacenar.
- Scene View: Permite a los desarrolladores tener una capacidad de edición y navegación visual para la escena que está creando, por lo que se tienen diferentes vistas según la opción que se elija. De esta manera, si se está desarrollando la escena, se fija una vista de la misma con las herramientas correspondientes para modificar sus elementos. Si por lo contrario se desea observar una prevista del juego, también se ofrece la posibilidad.
- Hierarchy Window: Muestra la representación jerárquica de cada uno de los objetos disponibles en la escena, mostrando la información de cómo los objetos se relacionan entre sí. Dado que toda la escena es el objeto principal, los objetos agregados se convierten en el objeto secundario de la misma. Este concepto también se conoce como paternidad en el mundo de Unity y ocurre cada vez que se agrega un elemento dentro de otro.
- Inspector Window: Permite a los desarrolladores inspeccionar y analizar todas las propiedades editables del objeto seleccionado. Ya que cada tipo de objeto puede tener diferentes conjuntos de propiedades con sus distintos diseños y contenidos, el inspector muestra una visualización acorde a cada tipo de objeto con sus elementos correspondientes.
- Toolbar: Es una ventana clave para el desarrollo, ya que contiene las herramientas principales para manipular la vista de la escena junto con los objetos contenidos dentro. Además, se encuentran los controles de reproducción y pausa para poder ejecutar el juego y ver la funcionalidad de los cambios realizados.

Una vez vistos los elementos principales de la interfaz de Unity, vamos a entrar más en detalle con los relacionados con la ventana del proyecto. Según se ha comentado, dicha ventana engloba los elementos que residen en el proyecto y se pueden utilizar, más conocidos como Assets. Cualquiera de estos recursos se ha podido importar desde fuera de Unity, aunque también se dispone de una colección de tipos Assets que se pueden producir dentro de Unity y utilizar en nuestro proyecto:

- GameObject: Cada objeto que se encuentra presente en el juego. Técnicamente, no agregan ninguna funcionalidad al proyecto, sino que simplemente actúan como soportes para componentes, incluyendo todos los necesarios dentro de un mismo GameObject. Como explicación, cabe destacar que los componentes son los bloques de construcción básicos, es decir, los elementos básicos de los objetos y sus actividades en un juego, actuando como piezas funcionales para cada GameObject.
- Escenas: Son la base, donde se pueden colocar los diferentes GameObjects y elementos para crear un nivel del juego. De esta manera, se pueden crear innumerables escenas, todas ellas

independientes entre sí y con su propia configuración y elementos que la componen. Para que estos niveles se incluyan en el juego, se deben añadir en la configuración de compilación. De este modo y mediante el scripting, se puede modelar el flujo de escenas por las que navegar en el juego.

- Prefabs: Son los componentes reutilizables de GameObject, es decir, instancias de un objeto con las mismas características para poder ser reutilizados en cualquier parte. De esta forma, los elementos que van a ser utilizados en distintas escenas pero que poseen la misma configuración, no tienen que ser creados cada vez que se quieran usar, sino que bastará con usar la misma instancia para todos.

Una vez vista la interfaz principal de Unity y los elementos cuyo uso es más común, vamos a adentrarnos en sus características más destacadas, las cuales avalan el uso de Unity como el software para la creación. Cabe destacar, que en la actualidad, este software es uno de los más importantes y utilizados. Creada por Unity Technologies, engloba motores para renderizar imágenes, motores de audio y motores de animación, los cuales son elementos claves para la creación de nuestro juego. Pese a todo esto, se disponen de muchas más características, así como de servicios que fundamentan el uso de dicho motor. En cuanto a las funcionalidades más típicas del motor, así como los servicios proporcionados, podemos destacar las siguientes:

- Analíticas: Para dicho software, se permite de una forma exhaustiva, obtener las analíticas sobre como es el juego del usuario. Este apartado es importante, ya que nos permite comprobar de primera mano cómo juegan los diferentes usuarios.
- Monetizar el videojuego: Para los pequeños usuarios que desarrollan por primera vez con este motor gráfico, es un gran incentivo que no pasa desapercibido. Unity tiene una política de uso clara, si eres un usuario que no estima obtener más de 100000€ de beneficios al año con el videojuego que ha desarrollado, se permite el uso totalmente gratis de este motor. Por este motivo, si se consigue exprimir al máximo el uso de publicidad o anuncios dentro de nuestro juego y dentro del límite permitido, se podrán conseguir una cantidad importante de ingresos en una primera instancia.
- Colaboración: Como comentaremos más adelante, es un factor clave, sobre todo para el desarrollo en equipos con un número elevado de integrantes. En este caso, la colaboración entre un gran número de personas no ha sido necesaria para este TFG, ya que solo ha sido desarrollado por un alumno. Pero la importancia de esto viene con la participación de los tutores, los cuales obviamente no han participado en el desarrollo del juego, pero si han tenido acceso a la comprobación del código y desarrollo de este. Esto supone una ventaja, ya que se puede comprobar de primera mano como está siendo el avance del alumno y así poder aportar las correcciones o comentarios necesarios en cada momento, así como acercar los problemas ocurridos en cualquier momento.
- Certificación de Unity: En el caso de obtener dicha certificación, hace que puedas validar de cara a otras personas el conocimiento en esta herramienta.
- Multijugador: Pese a que no ha sido integrado para este TFG, este motor permite implementar la función multijugador de una manera sencilla para poder enriquecer la experiencia de juego. En

nuestro caso, como ya se ha comentado, se busca que el usuario juegue de manera individual, pero esto no quita que sea una oportunidad importante para poder explotar en un futuro, ya que se puede pretender colaborar con más usuarios, cada uno desde un sitio diferente. Por tanto, se dispone la posibilidad de un futuro desarrollo con nuevas dinámicas o con la opción de modificar las ya existentes, para así poder hacer un juego más interactivo y participativo si cabe.

- Desarrollo de juegos 2D y 3D: Una de las características esenciales por la que se ha decidido utilizar Unity, ha sido por la facilidad ofrecida para desarrollar los dos tipos de juegos mencionados. Para nuestro caso, solo se ha necesitado el desarrollo en 2D, por lo que se ha permitido que este motor gráfico aporte su funcionalidad para renderizar los gráficos necesarios. Por otra parte y como es lógico, el renderizado de gráficos para un desarrollo 3D sería más costoso pero igualmente satisfactorio. Siguiendo con este tema, aunque no sea primordial para nuestro juego, cabe resalta la importancia del motor físico que simula las leyes de la física, permitiendo conseguir situaciones realistas. En nuestro caso, al ser un juego 2D y sin la necesidad de elementos como la gravedad, por ejemplo, no ha sido necesario incluir ningún scripting ni planteamiento relacionado. Aun así, es de gran importancia dicho apartado, ya que si en un futuro se desea modelar de esa forma, está abierto a dicha posibilidad.

- Animaciones: Otro apartado importante a resaltar es el de las animaciones. Una parte importante de cualquier videojuego es la interacción del usuario con los diversos elementos que se encuentran en él, así como la disponibilidad gráfica de observar los cambios que suceden a su alrededor. Entre las características más destacadas de la animación de Unity, podemos encontrar las siguiente:

  - Configuración de animaciones para todos los elementos de Unity
  - Flujo de trabajo sencillo y simplificado para alinear los clips de animación
  - Vista previa de clips de animación, transiciones e interacciones entre ellos
  - Funciones disponibles de capas y máscaras

  El sistema de animación de Unity se basa en el concepto de clips de animación, que contienen información sobre cómo ciertos objetos deberían cambiar su posición, rotación u otras propiedades a lo largo del tiempo. Cada clip se puede considerar como una sola grabación lineal. Por este motivo, como vemos en la figura 2.8, se pueden crear nuevas animaciones a partir de la modificación de las propiedades de cualquier sprite [14], es decir, objetos gráficos en 2D. Una vez creadas

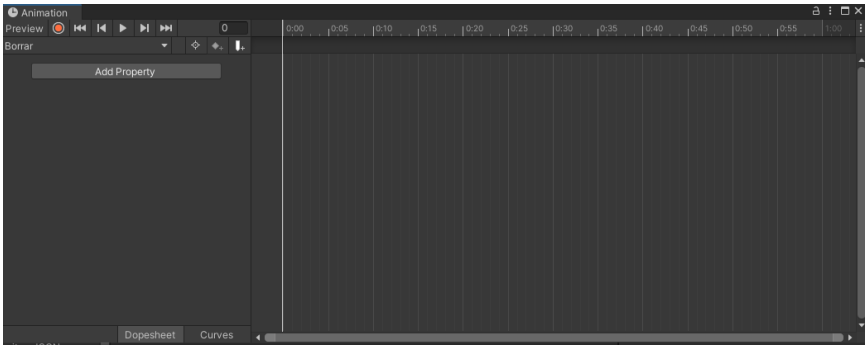

Figura 2.8: Línea temporal para crear las animaciones

las animaciones, se pueden tener varias para un mismo objeto, por lo que hay que administrar su flujo para poder transicionar entre ellas. La ordenación de las animaciones es idéntica a la de un diagrama de estado, es decir está compuesta por estados (que representas las diferentes animaciones) y transiciones que permiten cambiar de una a otra. Según vemos en la imagen 2.9, se observa el esquema principal del controlador, donde se pueden generar también triggers o variables para disparar el cambio entre los diferentes estados.

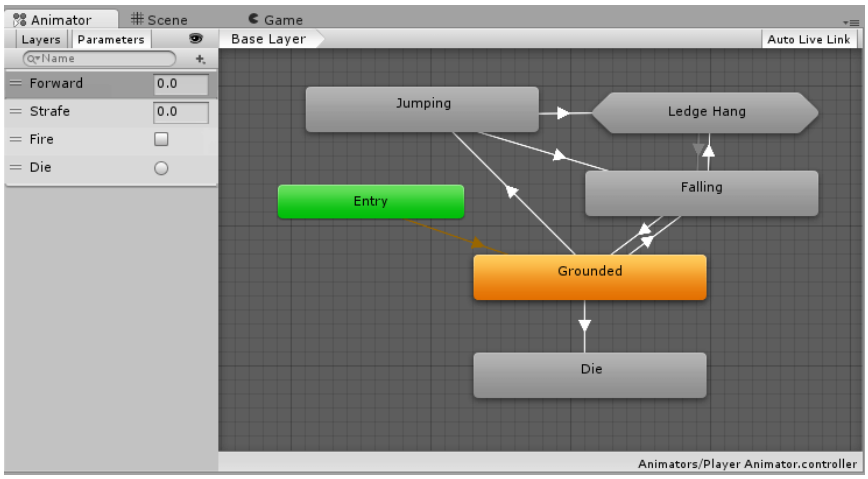

Figura 2.9: Controlador de animaciones para gestionar los estados

- Sonido: Este apartado constituye también una parte esencial del juego, ya que además de las animaciones mencionadas antes, uno de los objetivos del juego reside en la posibilidad de reproducir y grabar audios. De esta forma, que Unity permita el uso de una manera tan sencilla, es clave para su uso. En cuanto a la forma de implementar sonidos, se pueden destacar varias, pero en nuestro caso hemos optado principalmente por el uso del Audio Source. Este tipo de pistas disponibles para el sonido, permiten en primer lugar añadir cualquier tipo de sonido que deseemos al archivo. Así mismo, se cuenta con una amplia posibilidad de modificar los mismo o añadirles nuevas características. Suelen destacar los casos donde se usa la opción de reproducir directamente el sonido según se añade a la escena, como puede ser un sonido de fondo, o desactivar esta opción para activarla posteriormente mediante scripting, como puede ser la reproducción de un audio en un momento cualquiera. Así mismo la opción de repetir sonidos en bucle es importante, ya que se utiliza para escenarios en los que se dispone de un audio con una duración de unos pocos segundos, pero se estima que la estancia en dicha pantalla puede superar esa duración. En cuanto a detalles más técnicos, también hay diversas posibilidades para poder modificar los elementos, desde el pitch, pasando por el reverb o la reproducción en estéreo.

Una vez vistas las características principales que engloba Unity para el desarrollo de videojuegos, vamos a profundizar en detalles importantes que nos han hecho decidirnos por este motor para el desarrollo del TFG. Entre las cuales, destacan todas las relacionadas con el motivo de desuso de Flash, por ejemplo, las cuales suponen un punto fuerte en este caso y fueron una debacle para otros motores.

En primer lugar, destaca la documentación del software, siendo Unity el software con uno de los

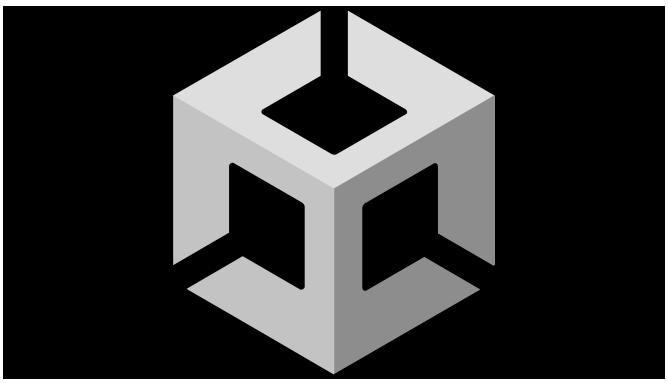

Figura 2.10: Logo de Unity [15]

mejores manuales que existen en el momento [16], los cuales se actualizan según se actualiza el propio software para que no vayan surgiendo diferencias, lo que produce un gran interés en su uso y un importante punto de referencia a la hora del desarrollo. Dichos manuales incluyen tanto información de uso, como de actualización a una versión concreta que queramos, así como guías para los usuarios más expertos para realizar tareas bastante avanzadas y que no todo el mundo necesita. Un punto clave de esta documentación, reside en el acceso a todo el historial de documentación disponible, es decir, el acceso a las versiones anteriores. Esta posibilidad de encontrar historiales pasados es enriquecedora, ya que, aunque se vayan actualizando junto con el software, las versiones anteriores no se pierden sino que se guardan para su consulta y no limita al usuario a comprobar solo la última versión más actualizada.
Por otra parte, también se dispone de la documentación referente a la API de programación o de scripting [17]. En esta parte, se dispone de todos los namespaces, así como sus clases, métodos y ejemplos de todos ellos para poder comprobar de primera mano el desarrollo referente a este apartado.

En segundo lugar, la comunidad de usuarios que dispone Unity es inmensa, por lo que facilita infinitos problemas emergentes con otros motores como puede ser Flash, ya que disponía de una comunidad en deterioro y con la que apenas se puede interactuar para obtener información. De esta manera, Unity cuenta tanto con un foro interno de usuarios, como un foro de usuarios por todo internet. De esta forma, se facilita la forma de ayudarse entre los usuarios cuando surgen dudas sobre el desarrollo, ya que con la inmensidad de personas que componen la comunidad, muchas de ellas ya han resuelto los mismos problemas que nos pueden estar sucediendo a nosotros por primera vez. Además, como punto a recalcar, al existir una comunidad tan grande, todas estas dudas originadas, son resueltas en un tiempo muy corto, ya que cualquier usuario puede responder sencillamente en los foros o páginas correspondientes, y al existir tantos, la probabilidad de que uno de ellos pueda solucionarnos el problema y de forma ágil es muy alta. Por último, cabe remarcar que en estos foros o comunidades mencionadas, no solo se explican los mejores métodos para resolver el problema de forma habitual, sino que también es fácil encontrar nuevas técnicas implementadas para estos problemas emergentes, las cuales nos pueden ayudar incluso de una forma más satisfactoria.

Por otra parte, y para finalizar con los elementos clave para la decisión de utilizar Unity, resalta la po-

sibilidad de ser exportado a diversas plataformas. Esta es una de las características más importantes, y a la par no compartida por otros motores gráficos como hemos visto anteriormente. De esta manera, es posible crear un videojuego para varias plataformas sin tener que limitarnos a una en concreto, lo que nos abre las diversas posibilidades de conocimiento y expansión del juego, ya que no solo se limita por ejemplo a un ordenador con ratón, sino que el mismo desarrollo se puede exportar en dispositivos móviles, y su funcionalidad y jugabilidad sigan siendo óptimas.
De esta manera, se puede elegir la plataforma con la que vamos a trabajar en el desarrollo de nuestro juego, teniendo en cuenta que aunque nuestro editor en ese momento pueda soportar Windows, MacOs o Linux, también se tiene la capacidad de crear el mismo para 25 plataformas diferentes. Cabe remarcar, que Unity, ofrece una ayuda para este proceso de exportación y no es un proceso totalmente automático, ya que por ejemplo, para la consolas, va a tener otro flujo de trabajo. Es se debe a que cada una de estas plataformas depende de los requisitos de la empresa, de unos procesos de certificación y de la implementación de ciertas características, pero de la misma forma, permite que el proceso no se realice de cero mediante la ayuda mencionada.

2.4.2. Unity Hub

A la hora de describir Unity Hub [18], podemos afirmar que es una herramienta para la gestión de proyectos Unity, además de poder gestionar todas las ediciones existentes de Unity que queramos, así como crear nuevos proyectos y abrir otros ya existentes. En cuanto a las funcionalidades que destacan para este editor, podemos encontrar las siguientes:

- Conectar y activar licencias: Se dispone de la opción de administrar tu propia cuenta y activar las licencias necesarias en ella. Para la gestión de proyectos, se puede optar por la creación de una cuenta para poder tener acceso directo a todos los proyectos de forma rápida. De la misma manera, y según lo mencionado en párrafos anteriores, se dispones de varias licencias para acceder a los servicios de Unity. Estas van desde licencias profesionales hasta gratuitas, accesible si se facturan menos de 100000€ al año con el proyecto desarrollado y enfocada a desarrolladores que están iniciando o no pertenecen a grandes empresas.
  Una característica que destaca a la hora del desarrollo es que no se permite gestionar proyectos en Unity Hub si no se dispone de una licencia, ya sea la gratuita o cualquier otra de pago. El proceso para añadirla es muy sencillo y se puede realizar hasta de tres maneras diferentes.

- Creación de proyectos: Se permite la creación de nuevos proyectos mediante plantillas o con archivos de proyectos de tutoriales. Entre las diversas plantillas, se encuentras las anteriormente comentadas, de 2D y 3D. Una vez creado el proyecto, uno de los puntos fuertes, es que se puede ejecutar con diversas versiones de Unity que se pueden descargar. El único inconveniente es el hecho de estar pendiente de las versiones con las que se desarrolla el proyecto y con la que posteriormente se desea abrir. Esto se debe a que proyectos nuevos desarrollados con versiones nuevas, pueden no ser compatibles con versiones mucho más antiguas y por tanto no permitir su apertura.

- Añadir componentes a las instalaciones: Cuando se descarga una versión del editor a través de

Unity Hub, se permite la opción de buscar y añadir nuevos componentes durante la instalación inicial o bien posteriormente. Entre los componentes adicionales que se permiten agregar, destacan algunos como la compatibilidad con plataformas específicas, documentos sin conexión, activos estándar o la descarga de Visual Studio como entorno de desarrollo para trabajar.

- Organizar y colaborar: Es importante mantener todos nuestros proyectos seguros en un solo lugar, ya que, si disponemos de distintas ubicaciones para cada proyecto o pruebas, será más tedioso encontrarlo. De cualquier modo, ya sea si trabajas junto con un equipo entero de desarrolladores, o por otra parte solo, como es el caso de este proyecto, es clave utilizar una herramienta como esta. Principalmente se puede administrar, actualizar y lanzar proyectos de Unity de forma fácil, por lo que es recomendable su uso.

- Descargar y administrar instalaciones: En cuanto a la diversas versiones que se permite instalar, destacan dos grupos. Primeramente, las versiones LTS, las cuales tienen un soporte a largo plazo y son recomendables usar, ya que resulta conveniente optar a grandes plazos de soporte. Seguidamente, destacan las versiones Tech Stream, las cuales son consideradas de prelanzamiento. De este modo se pueden instalar todas las versiones de Unity disponible que se deseen y administrar como se quiera, pudiendo eliminar y mantener las que se quiera. Además, y en relación a lo mencionado en el punto de creación de proyectos, se permite ejecutar cada proyecto en distintas versiones para aprovechar al máximo la plataforma.

- Aprendizaje: Unity Hub prima el hecho de poder seguir aprendiendo sobre cualquier tema relacionado con el editor o con la creación de proyectos. Por este motivo, brinda una ruta de aprendizaje diseñada para nuestras necesidades, mediante un apartado en el que se muestran documentos y proyectos los cuales enseñan a poder desarrollar o simplemente funcionalidades importantes. Este apartado cuenta con diversos videos explicativos, así como los propios proyectos para poder seguir el desarrollo de primera mano, lo que resulta conveniente usar en el inicio del desarrollo, es decir, en la etapa relacionada con el aprendizaje. Estos tutoriales y artículos están adaptados a los objetivos de aprendizaje del desarrollador, para que no se vea sobrepasado por la dificultad de estos. Como incentivo para premiar el uso de esta funcionalidad, se pueden lograr insignias, así como ascender de niveles hasta creador de Unity.

- Comunidad: Por último, y en relación con la importancia de la comunidad mencionada en el apartado de Unity, se pueden descubrir infinidad de recursos útiles y poder observar los que otros hacen mediante la opción de foros, blogs o respuestas proporcionadas sobre problemas, analizando los mismos y encontrando las soluciones pertinentes.

Una vez vistas las principales características del editor, cabe remarcar que en el manual de Unity Hub[19] aparecen listadas más funcionalidades, así como una lista de requerimientos para cualquier dispositivo en el que se desee instalar.

Figura 2.11: Logo de Unity Hub

2.4.3. Github Desktop

A la hora de poder trabajar en equipo, ya sea con otros desarrolladores o simplemente que otros usuarios que quieren acceder al código (como en este caso), es importante mantener nuestro proyecto actualizado y compartido. Para ello, la mejor opción y la utilizada en este TFG es mediante el uso de un repositorio. Para ello se permiten diversas formas, como por ejemplo desde línea de comandos, integrada en el propio editor o esta, que es la que se ha utilizado.
Por tanto, Github Desktop [20] es una herramienta que te permite interactuar con Git mediante el uso de una GUI, en vez de la línea de comandos que puede resultar menos intuitivo en algunos casos. Por eso, el uso de esta herramienta fomenta el uso de mejores prácticas, ya sea trabajando en equipo como solo. En cuanto a la principal funcionalidad, como cualquier gestor de código, se puede crear, agregar o clonar un repositorio para empezar a trabajar. Seguidamente, se permite la creación de ramas para poder trabajar de forma paralela en las funcionalidades con la seguridad de no afectar al proyecto final y poder integrarlo cuando se desee. También se observa la pestaña de cambios realizados, donde se muestran todas las modificaciones hechas en los diversos ficheros. En algunos casos, como la cantidad de ficheros modificados es muy elevada, pero solo nos interesan unos pocos, se procede a la creación de un fichero llamado gitignore. Este archivo se coloca en el directorio raíz del proyecto e indica a Git que archivos o carpetas debe ignorar en el proyecto, para poder tener una visión más clara y rápida de los cambios que hemos realizado y nos interesan.
Otra funcionalidad imprescindible, una vez vistos y comprobados los cambios, es la de integrarlos en la rama. Para ello, se permite la opción de hacer un commit con ellos. Además, una vez realizado, se puede observar el historial, para tener un mejor seguimiento de todos los cambios que se han ido produciendo.
Por último, cabe resaltar que se puede usar GitHub Desktop para crear informes de problemas o solicitudes de extracción para colaborar en proyectos con otras personas. Estos informes, ayudan a llevar un seguimiento de las ideas referentes al proyecto, así como la posibilidad de realizar un debate acerca del desarrollo del mismo. Estas solicitudes, permiten compartir tus cambios propuestos con los demás, además de recibir retroalimentación y fusionar los cambios en un proyecto.

Figura 2.12: Logo de GitHub Desktop

2.4.4. C#

C# [21] es un lenguaje de programación multiparadigma desarrollado y estandarizado por Microsoft, con el objetivo de permitir a los desarrolladores crear una infinidad de aplicaciones ejecutadas en .NET Framework. Si nos centramos en una explicación general, podemos afirmar que es un lenguaje sencillo, fácil de aprender por tanto para nuevos desarrolladores que se enfrentan a nuevos retos. Esto es debido a que su sintaxis es fácil de entender y aprender, por lo que si se está familiarizado con otro tipo de lenguaje previo resultará muy intuitivo. En nuestro caso, se han encontrado diversas características similares al lenguaje de programación Java, por lo que ha sido muy sencillo adaptarse a este nuevo lenguaje, ya que no se ha necesitado casi tiempo previo en entender su sintaxis o estructuras de control, por ejemplo.
Además, cabe resaltar, que C# es un lenguaje orientado a objetos, ya que proporciona construcciones del lenguaje para admitir directamente estos conceptos, por lo que se trata de un lenguaje natural en el que crear y usar componentes de software. Como última característica principal, se puede afirmar que también está basado en seguridad de tipos.

En cuanto a sus orígenes, se empezó a gestar la creación de este en 1999, cuando Andrés Hejlsberg decidió dar el paso. Sus orígenes residen en la familia de lenguajes C, aunque su primera versión, como hemos comentado antes, se parece mucho a Java, ya que se buscaba como una alternativa viable. De esta forma, y con el paso del tiempo, se ha conseguido que este lenguaje admita los conceptos de encapsulación, herencia y polimorfismo (compartidos con Java), así como la opción de facilitar el desarrollo de componentes software.
En el párrafo anterior se han visto las características generales del lenguaje, pese a ello, vamos a indagar un poco más profundo en ellas y en otras, las cuales han sido importantes a la hora de desarrollar nuestro proyecto:

- Sintaxis sencilla: Según lo comentado anteriormente, su sintaxis carece de mucha complicación. Esto sumado a su similitud con un lenguaje visto en nuestro caso como es Java, hace que su aprendizaje haya sido fácil.

- Sistema de tipo unificado: De esta forma, se permite el uso de operaciones comunes, así como los valores de todos los tipos se puedan almacenar, transportar y usar de forma coherente.
- Orientación a componentes: Antes, hemos comentado que este lenguaje estaba orientado a objetos, pero a su vez, también está orientado a componentes. Esto se debe a que permite definir propiedades sin la necesidad de crear métodos sin tratar con punteros a funciones.
- Espacio de nombres.
- Bibliotecas: Cualquier compilador de C# contiene un número mínimo de bibliotecas disponibles para usar. En nuestro caso hemos utilizado varias. Algunas de ellas necesarias para realizar cualquier operación con nuestros eventos en 2D, otras simplemente para la creación o manipulación de los elementos de sonido, y otras como opción previa al reconocedor de voz implementado, es decir, mediante el uso de una biblioteca con un reconocedor y un sistema de diccionarios que detectaba las palabras clave introducidas.
- Integración con otros lenguajes de programación.
- Multihilo: En este lenguaje, se permite dividir el código en múltiples hilos de ejecución, para poder trabajar en paralelo y sincronizarlos al final. En nuestro proyecto se ha trabajado de forma secuencial, pero esto no descarta el hecho de poder trabajar con hilos para obtener un mayor rendimiento en la aplicación, sobre todo si se sigue desarrollando a gran escala y alcanza unos niveles de soporte más grandes.

Figura 2.13: Logo de C#

2.5. Comparación con proyectos similares

En esta sección se procede a analizar las aplicaciones o proyectos similares que se encuentran en el mercado. Para ello, se realiza una búsqueda a favor de detectar las fortalezas y debilidades en el trabajo que otros ya han realizado, para tratar de evitar los mismos errores e impulsar los puntos positivos.

2.5.1. Desarrollo de videojuegos en Unity para educación

Este proyecto [22] aborda el aprendizaje de valores, tales como la educación, el respeto o el altruismo, mediante una aventura gráfica que cuenta con una serie de escenarios éticos.
En cuanto a las similitudes que se pueden apreciar con nuestro proyecto, se observan en las mecánicas o comportamientos del mismo, ya que podemos observar la aparición de diálogos con otros personajes, lo cual nos muestra como interactuar con ellos y las posibles respuestas. Esto genera varios dilemas éticos, los cuales deben resolverse mediante la toma de decisiones. Podemos apreciar los siguientes escenarios:

- Generosidad
- Compañerismo
- Educación
- Altruismo

Para apreciarlo mejor, se muestra la imagen 2.14 donde se ve la toma de decisiones, las cuales serán evaluadas para fijar el comportamiento que se ha tenido.

Figura 2.14: Toma de decisiones para evaluar nuestro comportamiento

Además de esto, también se pueden tomar referencias del uso del inventario. Este proyecto también dispone de uno, donde se almacenan los elementos posibles y se guardan para su posible uso cuando se necesite. De esta manera, como se ve en la imagen 2.15, disponemos de un apartado para guardar e interactuar con los objetos de la escena.

En cuanto al resto de puntos del juego, cabe destacar que también hay diferencias con el nuestro a la hora de tomarlo como referencia. Por ejemplo, en nuestro caso, la perspectiva será 2D, mientras que en este, el desarrollo del videojuego es en 3D, lo que influye en los movimientos y posicionamiento de los personajes y animaciones. Además, el almacenamiento de datos se produce en una BD, lo cual dista de nuestro interés.

Figura 2.15: Inventario del juego

2.5.2. Desarrollo un juego para el aprendizaje de matemáticas en educación primaria

En el TFG [23], se utiliza Unity para fomentar el aprendizaje de las matemáticas en algunos cursos de la educación primaria, por lo que también se aprecia el carácter educativo y la intención por enseñar en este desarrollo. Por ello, se pretende desarrollar la flexibilidad de pensamiento en los niños de tal modo que les permita comprender las matemáticas de una manera fácil y mediante el uso de técnicas para favorecer el aprendizaje, en las cuales nos podemos fijar:

- Acumulación de puntos: Si se refuerza positivamente un comportamiento, conseguimos mayor interés por parte de los usuarios. De esta manera, según se aciertan las pruebas se obtienen puntos que se van acumulando.
- Escalado de niveles: Se proponen una serie de niveles que el jugador escala según los va realizando. Además, se favorece la dificultad de los mismos, escalando progresivamente en ella.
- Obtención de premios: Al igual que antes, según se consiguen objetivos se van otorgando premios a modo de colección.
- Misiones: Se proponen misiones a resolver, acompañadas de retos. De esta manera, se busca superar los objetivos marcados en los diferentes niveles.

De esta manera, se busca conseguir un aprendizaje más efectivo por parte de los alumnos. En la imagen 2.16, se puede ver una escena del juego, donde mediante una pantalla, se propone un reto para realizar una suma concreta. El procedimiento del resto de operaciones es similar, aunque varían los niveles y pruebas según su progresión.

2.5.3. CirQuizz

Este desarrollo [24], corresponde a la finalización de un TFG. Su premisa se basa en el aprendizaje de las nociones básicas de los circuitos eléctricos para las personas sin experiencia. De esta manera, se consigue acercar a los alumnos a la introducción de esta enseñanza debido a la posible falta de

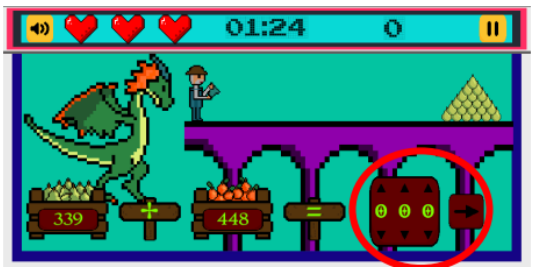

Figura 2.16: Capturas de un nivel con la operación de suma

espacio o recursos en un laboratorio convencional. La forma de aprendizaje se divide en 4 juegos que dinamizan la enseñanza. Al igual que en nuestro desarrollo, varios de ellos tienen la posibilidad de elegir una de las múltiples opciones, siendo solo una de ellas la correcta. Entre las formas de aprendizaje que facilita, son las siguientes:

- Buscaminas: Se divide la placa en tramos, enseñando solo una parte de la placa cada vez. El usuario, deberá resolver un enigma para encontrar la solución del sistema. Posee varios niveles, los cuales incrementan su dificultad según se avanza por ellos.
- Ruleta: Consiste en tirar de una ruleta la cual está llena de colores correspondientes al valor de una resistencia. Una vez obtenido el color en la tirada, se muestran 3 opciones donde solo 1 es la correcta. De esta manera se pretende calcular el valor de una resistencia mediante las opciones emergentes.
- Conexiones: En esta prueba se busca afianzar la noción del esquema interno de una placa de montaje. Para ello, se busca construir una placa en la que hay una conexión hecha y varios pines disponibles. Entre todos, se tiene que señalar el correcto, es decir, el que está interconectado con los otros dos.
- Gira Gira: En la pantalla aparecen una serie de resistencias giradas y desordenadas, el jugador debe interactuar con ellas para girarlas y posicionarlas de forma que se pueda leer el código y calcular el valor de la resistencia.

Según vemos en la imagen 2.17 y en la imagen 2.18, se pueden apreciar los distintos niveles educativos que hemos explicado anteriormente.

2.5.4. Recycle

Este proyecto [25], consiste en otro juego educativo basado en el reciclaje. Para ello, se presentan elementos en pantalla que el usuario deberá reconocer y clasificar correctamente según los materiales con los que está hecho. Esto ayudará a los jugadores con el uso cotidiano de estas situaciones para saber dónde debe reciclarse cada tipo de desecho.

De esta manera, presentan cuatro niveles de dificultad. Según se aumenta la dificultad, se dispone de menos tiempo para responder, así como de menos intentos para conseguir la opción correcta. Según

Figura 2.17: Capturas de los niveles 1 y 2

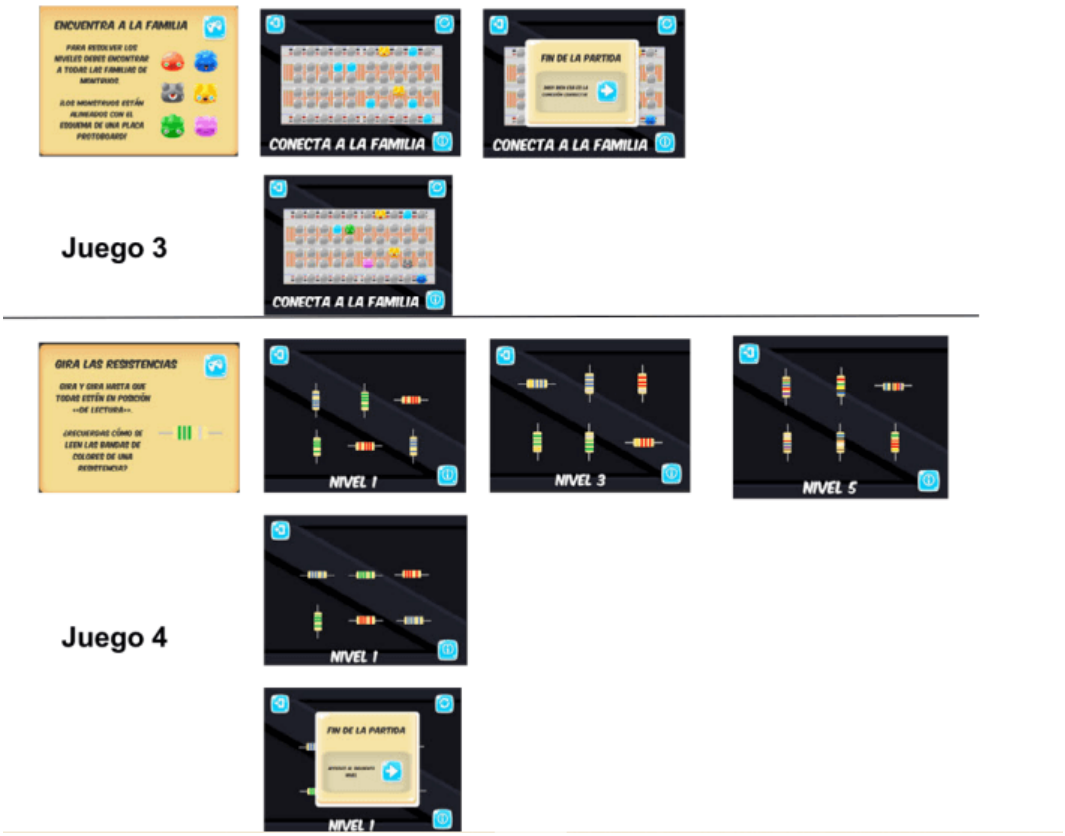

Figura 2.18: Capturas de los niveles 3 y 4

se ve en la imagen 2.19, se muestra un nivel del juego, en el cual se debe seleccionar la opción correcta entre todas las mostradas. Además, se presenta un menú de pausa, el cual puede ser tomado en cuenta

para nuestro desarrollo. Como última apreciación, cabe remarcar el uso de los audios para este juego. En forma de música y efectos de sonido, el usuario podrá escuchar un sonido de fondo mientras supera los niveles, o escuchar sonidos cuando interactúa con los elementos.

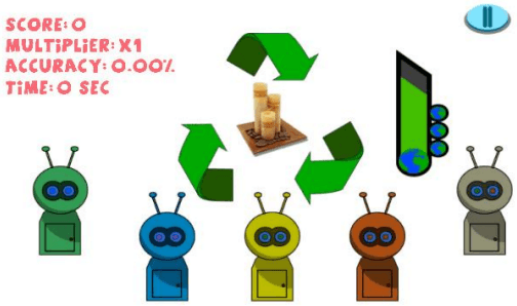

Figura 2.19: Nivel del juego con una opción para reciclar

Capítulo 3

Planificación

3.1. Planificación inicial

Esta sección de centrará principalmente en la organización del trabajo para cumplir con los requisitos fundamentales del proyecto. Para ello se realizará una metodología ágil basada en iteraciones incrementales, según como vimos en la asignatura de Planificación y Gestión de Proyectos del grado en Ingeniería Informática en la UVa. Por tanto, se aproximarán unas horas de trabajo adecuadas para cumplir con todas las necesidades.

3.1.1. Distribución temporal

La realización de este TFG se va a hacer entre el primer y segundo cuatrimestre del curso 2021-2022. La primera iteración comienza el día 28 de octubre, con la primera reunión con los tutores, mientras que la última iteración finalizará el 6 de junio (coincidiendo con la fecha de inicio de solicitud de las defensas). De este modo se tiene de margen 18 días, es decir, hasta la fecha límite para solicitar la defensa. De esta manera se cuentan con 222 días para poder realizar el proyecto. Los días de trabajo previstos serán de lunes a viernes, contando con los fines de semana libres de carga de trabajo, pero disponibles para poder realizar horas extras si así fuera necesario.

Ya que el TFG se va a comenzar en el primer cuatrimestre, cabe destacar que se compagina con 3 asignaturas más en este período, mientras que en el segundo cuatrimestre solo se realizarán las prácticas de empresa. Además, se cuenta con 2 exámenes en la convocatoria ordinaria de enero, por lo que se dejará un descanso para su estudio intensivo y realización de los mismos entre los días 6 y 17 de enero. Por tanto y como se observará más adelante, la carga de horas de trabajo hasta enero (pasados los exámenes), será menor que en todo el segundo cuatrimestre. Por lo que, la primera iteración contaría con 1 hora de trabajo las 4 primeras semanas y 1 hora y 30 minutos de trabajo las otras 6 semanas. Por otra parte, el resto de iteraciones hasta terminar el proyecto serán de 3 horas diarias. Si realizamos los cálculos obtenemos 380 horas para todo el proyecto. La normativa actual

del TFG contempla 300 horas de trabajo para el mismo, por lo que se dispone de 80 horas extra para posibles retrasos, inconvenientes o mejoras en el proyecto.

Como se ha comentado antes, el tiempo de trabajo está dividido en iteraciones. A continuación se muestra la propuesta inicial:

- **1ª Iteración (10 semanas, 80 horas)**
  Se realizará una primera reunión con los tutores del proyecto para presentarlo y exponer las ideas más generales del mismo. Se explicarán a grandes rasgos las tareas a realizar en el proyecto, herramientas utilizadas y dudas iniciales por parte del alumno. A lo largo de esta iteración, el principal trabajo será familiarizarse con la tecnología que se va a manejar, mediante documentación y realización de tutoriales (como los de Unity3D) o cursos como "La hora del código". Estas semanas claramente serán de iniciación y familiarización con el proyecto, ya que como se ha explicado antes, al tener asignaturas de cuarto curso, no se puede hacer un trabajo exhaustivo todavía.

- **2ª Iteración (5 semanas, 75 horas)**
  Esta segunda iteración se empezará a realizar una vez terminada la convocatoria ordinaria de exámenes. Por tanto, ya se dispone de tiempo suficiente como para centrarse en el TFG de forma completa, ya que hasta que se empiecen las prácticas de empresa dispondremos de todo el tiempo libre necesario. Por tanto, en esta iteración y una vez familiarizado con las herramientas y lenguaje necesario, se entrará de lleno en la realización del juego. Por tanto, los objetivos para este apartado son tanto empezar con la migración del juego, como desplegar el proyecto para comprobar que las implementaciones son correctas, así como realizar las pruebas y test correspondientes al mismo.

- **3ª Iteración (4 semanas, 60 horas)**
  Para la tercera iteración contaremos con otras cuatro semanas, aunque realizando las mismas horas de trabajo, se entiende que se estará compartiendo el desarrollo del proyecto con las prácticas de empresa. Aun así, se dispone de tiempo suficiente para el correcto desarrollo del mismo. Por tanto, la tarea principal para este apartado será la de completar una de las dinámicas, así como el nivel que la incorpore.

- **4ª Iteración (8 semanas, 120 horas)**
  Para la cuarta iteración, como se puede observar la más larga, contamos con 8 semanas para desarrollar el grueso del proyecto. Aun así, el trabajo diario será el mismo que para las dos anteriores iteraciones, es decir 3 horas. La tarea principal de esta fase será terminar de migrar el proyecto como tal, es decir, integrar todo el diseño y funcionalidad del mismo. Para ello se buscará realizar la dinámica que falta, así como todos los niveles correspondientes.

- **5ª Iteración (3 semanas, 45 horas)**
  Para la última iteración, como se puede observar la más corta de todas, contamos con solo 3 semanas, aunque tiempo suficiente como para terminar todo satisfactoriamente. Para este apartado se realizarán reuniones más asiduas con los tutores para conseguir mayor retroalimentación, así como solucionar posibles fallos de manera más rápida. También se finalizará con la memoria dándola por cerrada.

En la Figura 3.1 se puede apreciar la distribución gráfica de las semanas para cada iteración. Se puede observar como la iteración que más semanas va a llevar de todo el proyecto, en principio, es la 1ª iteración. Esto no se debe a que sea la iteración con más trabajo, al contrario, sino se debe a que es la parte de documentación y familiarización, por lo que es más extensa. Debido a la compaginación del TFG con otras asignaturas en el primer cuatrimestre, se ha decidido distribuir de esta forma la planificación, con una primera parte más extensa pero menos laboriosa.

En lo que respecta al reparto de horas, y sabiendo que a partir de la 2ª iteración el número de horas diarias aumenta a 3, se puede ver en la Figura 3.2 como queda el reparto final.

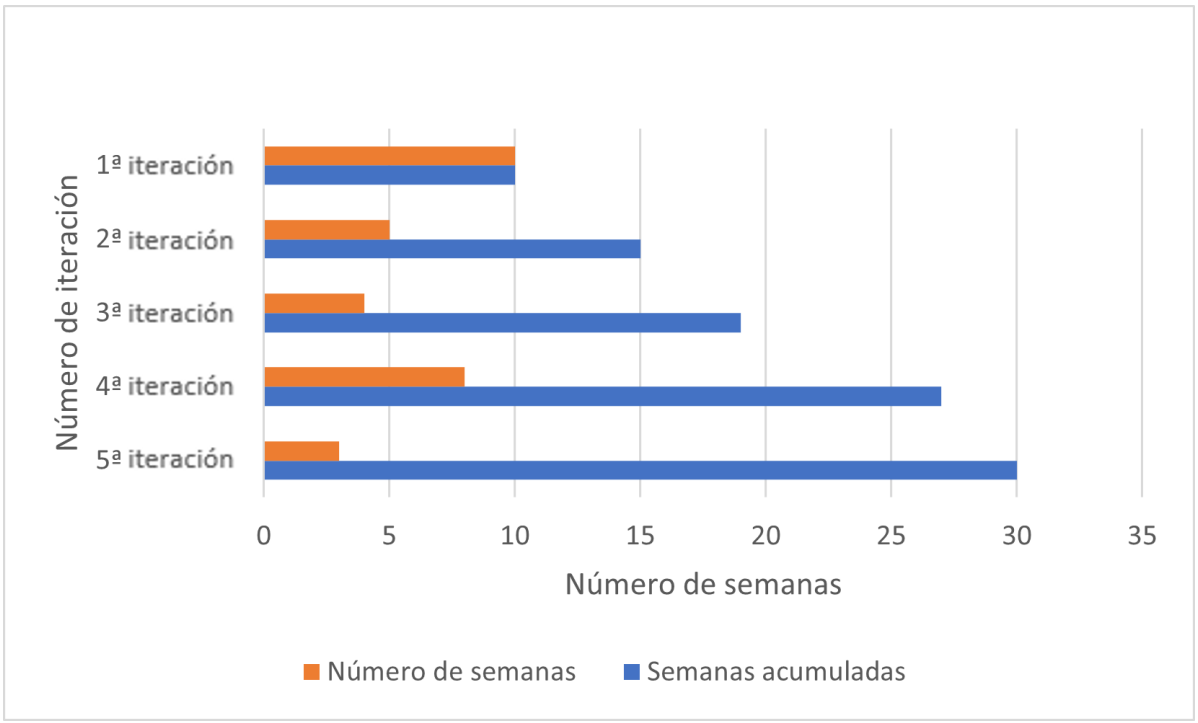

Figura 3.1: Distribución del número de semanas de desarrollo en cada iteración

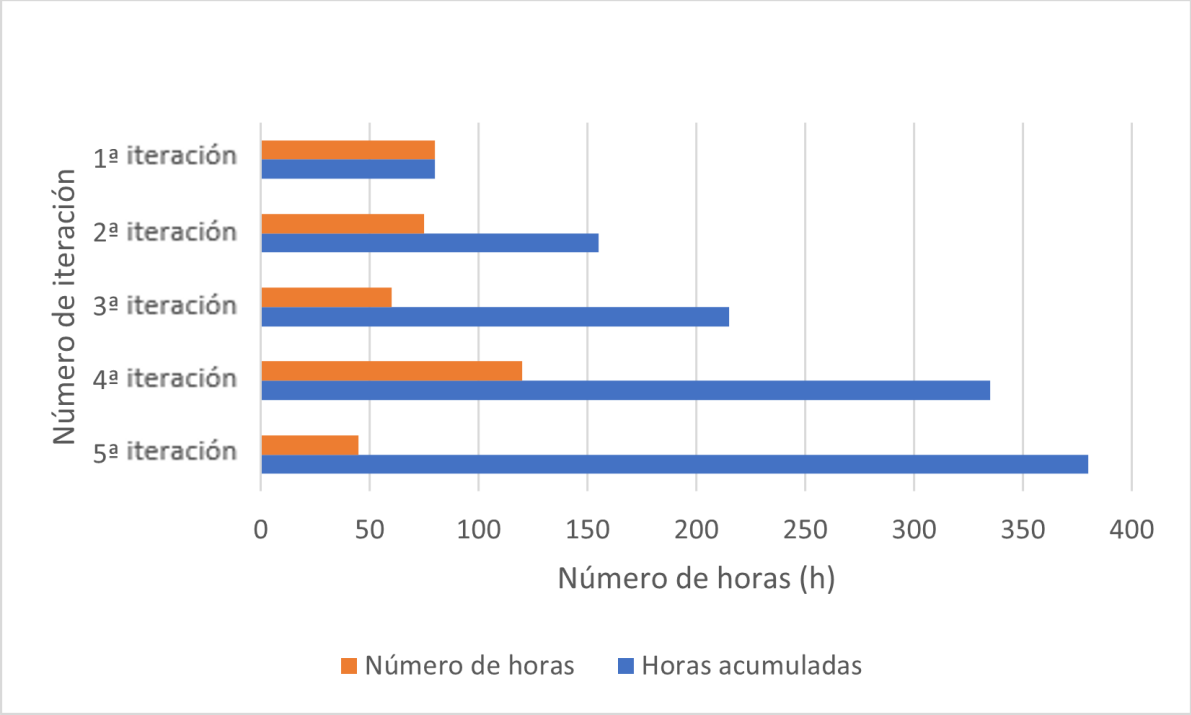

Figura 3.2: Distribución del número de horas de desarrollo en cada iteración

3.1.2. Análisis de riesgos

En la siguiente sección se incluirán los riesgos que se aprecian probables de suceder y cuya aparición podría provocar retrasos a la hora de realizar el proyecto. Los riesgos que se van a describir aparecen junto a la estimación de su probabilidad, una breve descripción para comprenderlo correctamente y un impacto que supondría sufrir el riesgo mencionado, es decir, un retraso estimado que podría darse. En cuanto a este último punto, cabe destacar que cada día de retraso incluido en esta columna, corresponden a 3h/hombre de trabajo. Todo lo descrito anteriormente puede observarse en la Tabla 3.1. Como apunte en esta tabla, la probabilidad está indicada en tanto por uno, mientras que el retraso estimado lo está en días.

Como se puede observar, al haber unos riesgos existentes explicados anteriormente, también existen una serie de acciones para reducir la posibilidad de que ocurra dicho riesgo, es decir, un plan de reducción de riesgo. Por otra parte, se observa la posibilidad de una vez ocurrido el riesgo sea lo más leve posible, es decir acciones llevadas a cabo una vez producido el riesgo, o lo que es lo mismo un plan de mitigación. La tabla 3.2 muestra con detalle, según lo explicado anteriormente, tanto las medidas de reducción de cada riesgo, como las medidas de contingencia en caso de que se produzca el mismo. Como se puede ver en esta tabla se detalla el plan de acción para cada uno de los riesgos posibles que ocurren en el discurso del proyecto. En cuanto al último campo que faltaba por aclarar, es decir, la exposición al riesgo, es el tiempo promedio que se perdería debido a cada uno de los riesgos. La forma de calcularlo se obtiene multiplicando las columnas de probabilidad de que el riesgo suceda y retraso estimado, las cuales se encuentran en la Tabla 3.1.

ID	Riesgo	Probabilidad	Descripción	Retraso estimado
1	Enfermedad	0.05	El desarrollador contrae una enfermedad de media duración	7
2	Contagio por COVID	0.05	El desarrollador contrae el COVID con síntomas e impedimento de trabajo	7
3	Falta de experiencia con Unity3D	0.1	Se emplea más tiempo del debido al inicio al realizar las tareas	10
4	Falta de experiencia con C#	0.05	Se emplea más tiempo del debido al inicio al realizar las tareas	7
5	Planificación incorrecta	0.5	Se provocan retrasos debido a la falta de experiencia en planificar y recortar plazos continuamente	15
6	Pérdida de datos en el desarrollo	0.25	Se tienen problemas con los entornos de desarrollo o las plataformas donde se almacenan y trabajan los datos	2
7	El ordenador de trabajo sufre algún problema	0.1	El ordenador con el que se desarrolla el proyecto sufre algún impedimento para seguir trabajando con él	5
8	Problemas con herramientas del desarrollo	0.25	Alguna herramienta o aplicación de trabajo sufre problemas e impide el desarrollo habitual	2
9	Cambio en los requisitos	0.2	Se añade, elimina o modifica algún requisito en el desarrollo del proyecto	15

Tabla 3.1: Riesgos durante el desarrollo del proyecto

ID	Riesgo	Exposición	Plan de reducción	Plan de mitigación
1	Enfermedad	0.35	Tener cuidado con el frío en los meses más helados	Se usarán las horas del fin de semana para el retraso
2	Contagio por COVID	0.35	Cumplir las medidas de prevención del estado, como mascarilla o vacunación	Se usarán las horas del fin de semana para el retraso
3	Falta de experiencia con Unity3D	1	Realizar tutoriales y documentación necesaria antes de empezar con el proyecto como tal	Se usarán documentación oficial para las dudas, así como las horas del fin de semana para el retraso
4	Falta de experiencia con C#	0.35	Realizar tutoriales y documentación necesaria antes de empezar con el proyecto como tal	Se usarán documentación oficial para las dudas, así como las horas del fin de semana para el retraso
5	Planificación incorrecta	7.5	Ser realista con los plazos y tareas a realizar en los mismos	Se usarán horas de los fines de semana y días posteriores a la fecha de finalización para mantener la entrega final del proyecto antes del final de la primera convocatoria
6	Pérdida de datos en el desarrollo	0.5	Uso de git para mantener guardada cada versión del proyecto, así como auto guardado y uso de la nube para las copias del documento	Se usarán las copias del último día de trabajo previo a la pérdida
7	El ordenador de trabajo sufre algún problema	0.5	Mantener el ordenador actualizado, cuidado y en un lugar seguro	Trabajar con otro ordenador de sobremesa que dispongo, hasta la compra de uno nuevo o reparación
8	Problemas con herramientas del desarrollo	0.5	Mantener las aplicaciones actualizadas, y tener una segunda herramienta de apoyo por si falla la otra	Documentarse para poder solucionar el problema, si es irreversible, buscar otra herramienta similar
9	Cambio en los requisitos	3	Aplicar la metodología ágil en el desarrollo para reducir el impacto	Realizar reuniones para fijar el nuevo requisito, planificando una depuración e implementación de los cambios

Tabla 3.2: Plan de acción de los riesgos del proyecto

3.2. Entorno tecnológico

3.2.1. Herramientas utilizadas

En esta sección se procede a describir todas las herramientas utilizadas en el proyecto junto con las funcionalidades más destacadas e importantes de cada una de ellas.

- **Astah [26]**: Creación de diagramas UML.

- **C# [21]**: Lenguaje de programación multiparadigma, orientado a objetos y componentes, además de contar con una sintaxis sencilla y similar a Java en diversos aspectos.
- **Chrome [27]**: Navegador usado para acceder a overleaf y redactar la memoria, así como buscar información correspondiente y documentación.
- **Excel 365 [28]**: Herramienta usada para la realización de tablas y figuras del proyecto

- **Git [29]**: control de versiones para el código del proyecto.

- **GitHub Desktop [20]**: Permite interactuar con Git mediante el uso de una GUI, en vez de por línea de comandos, lo que resulta más intuitivo. Permite gestionar cualquier proyecto mediante la creación de ramas, modificación e integración del código, consulta del historial de cambios realizados...
- **Gitlab [30]**: Repositorio remoto para almacenar todo el código fuente y datos del proyecto.
- **Google Drive [31]**: Almacenamiento de ficheros en la nube relacionados con la memoria.

- **Jitsi Meet [32]**: Software de videoconferencia y mensajería instantánea, mediante el cual se realizan las tutorías necesarias para comprobar el desarrollo del proyecto y resolver dudas más extensas.
- **One Drive [33]**: Almacenamiento complementario de ficheros en la nube, donde se encontrarán los elementos necesarios para desarrollar el juego, así como las dinámicas o explicaciones pertinentes.

- **Overleaf [34]**: Editor colaborativo de LaTeX basado en la nube que se utiliza para escribir, editar y publicar documentos, en este caso para realizar la memoria del TFG.

Ordenador	
Asus X555LD	
Hardware	
Procesador	Intel Core i7-4510U
Memoria	8 GB
Disco SSD	1 TB
Tarjeta gráfica	Nvidia Geforce 820M
Software	
Sistema operativo	Windows 10
Arquitectura del sistema	64 bits

Tabla 3.3: Ordenador utilizado para llevar a cabo el desarrollo

Monitor	
LG 27GL850	
Resolución	2560 x 1440
Tamaño	25 pulgadas
Ratio de aspecto	16/9

Tabla 3.4: Monitor utilizado en el desarrollo

- **Telegram [35]**: Comunicación con los tutores del proyecto sobre el avance del mismo, consulta y resolución de dudas rápidas.

- **Unity 2021.2.9f1 [36]**: Motor gráfico de videojuego multiplataforma creado por Unity Technologies. Está disponible como plataforma de desarrollo para Microsoft Windows, Mac OS, Linux.

- **Unity Hub [18]**: Herramienta para la gestión de proyectos Unity, además se pueden gestionar múltiples. instalaciones del editor Unity junto con sus componentes, crear nuevos proyectos y abrir proyectos existentes. La herramienta Unity Hub se puede utilizar tanto para administrar la cuenta de Unity como para licencias de editor.

- **Visual Studio [37]**: Entorno de desarrollo integrado desarrollado por Microsoft. Consta de las herramientas y tecnologías necesarias para realizar nuestro código y scripts necesarios.

3.2.2. Entorno de desarrollo

En este apartado encontraremos lo referente a la máquina y entorno donde se ha desarrollado el proyecto, tanto lo que nos incumbe en el desarrollo como en las pruebas realizadas.

3.3. Estimación de costes

Para el apartado presente, se va a detallar una estimación inicial sobre los costes del proyecto, es decir, tanto los humanos como los técnicos. Desde el salario bruto del desarrollador hasta los de los equipos, licencias y servicios, todo ello se procede a contabilizar con IVA.
El enfoque para calcular el mismo, se basa en la ley referente al teletrabajo en España, la cual establece que la empresa debe remunerar al trabajador por los gastos derivados del trabajo en casa, mediante el convenio colectivo del sector. Por lo que los gastos serán imputables, referidos al domicilio actual desde el que se ha realizado el TFG, además, la referencia de estos datos para obtener el cómputo global, hace que sean tomados según la media geométrica del último año. Por todo ello, pasaremos a comentar en los distintos subapartados todos los costes que infieren en el desarrollo de este proyecto, según lo comentado en la introducción.

3.3.1. Salario del trabajador

Para este TFG, el alumno Gonzalo Fernández González, no ha sido contratado por la Universidad de Valladolid, ni ha realizado el proyecto en colaboración con ninguna empresa como extensión de las prácticas. Por este motivo el sueldo recibido han sido 0€, pero para hacerlo realista y estimar los costes reales, vamos a proceder a calcularlo. En primer lugar, se atribuye el desarrollo de este proyecto a un programador Junior. Según la referencia de sueldo medio en España por un programador de esas características [38], se han recopilado los datos del presente año para los desarrolladores con una menor experiencia. Dicho sueldo varía entre los 16000 y 22000€, por lo que la media anual se encontraría en unos 19000€. Una vez tomado este valor para realizar los cálculos pertinentes a este apartado, y teniendo en cuentas que se trabajan 1800 horas al año [39], quedaría un salario bruto de 10.55€/hora. En nuestro caso, con un trabajo estimado de 380h, el coste total del salario bruto del desarrollador es de 4009€.

3.3.2. Espacio de trabajo

El proyecto se ha realizado en el lugar de residencia del desarrollador, por ese motivo vamos a recabar los costes que supondrían el alquiler y gastos complementarios. En cuanto al primero, incluyendo los gastos de comunidad, seguro del hogar y el pago del IBI, la cantidad correspondiente al alquiler por la zona de residencia y mercado actual asciende a 450€/mes. De este modo, el coste del mismo por hora serían 0.625€. Así mismo, como el trabajo estimado asciende a 380 horas, el precio correspondiente a las horas trabajadas sería de 237.5€ en total.

Además de los costes correspondientes a los comentados, también existen los servicios básicos relacionados, es decir, los utilizados para poder desarrollar el proyecto de forma satisfactoria, entre los

que se encuentran la luz e internet.
La factura de internet asciende a 50€/mes. Contando que el desarrollo efectivo del TFG, ha sido de 380 horas de trabajo, nos quedaría un coste de 26.38€, lo que supone 0.069€/hora.
En cuanto a la luz, según la información contrastada [40], el uso de ordenador 6h al día y 180h/mes supone 1,8kWh, como en nuestro caso el uso será de unas 63h/mes a lo largo de los 6 meses, se estima un gasto de 3.8 kWh, que con un precio de 0.13€/kWh supondría un coste total de 0.494€.

3.3.3. Material físico

Para comprobar la cuantía invertida en el material físico, tendremos en cuenta su coste en el momento de compra, para poder realizar los cálculos de todos los dispositivos utilizados para realizar el presente TFG. Entre las herramientas usadas, disponemos del ordenador con el que se ha desarrollado el proyecto y la pantalla complementaria que ha servido de apoyo al mismo. En cuanto al ordenador, su precio de compra fue de 800€, por lo que será el precio que consideremos para calcular los costes. Por otro lado, se ha utilizado también un monitor, cuyo coste de adquisición fue de 400€. Sumando los precios de ambos dispositivos, podemos determinar que los costes correspondientes al hardware utilizado para el desarrollo de esta práctica ascienden a 1200€.

3.3.4. Costes del software

Para este apartado mencionaremos el coste real que hubiera supuesto comprar el software o licencias necesarias para el correcto desarrollo del presente TFG. Para ello, cabe destacar que en nuestro caso, todo el software ha sido gratuito, ya sea por su valor actual o por las licencias recibidas por parte de la universidad. Aun así, vamos a calcular el valor real que habría supuesto sin todo ello.
Como hemos dicho, el uso de Unity es totalmente gratuito para un particular que no tiene previsto facturar más de 100000€ con su proyecto, como es nuestro caso, por lo que esto seguiría siendo gratuito. Por otra parte, el uso de Astah, necesario para realizar los diagramas correspondientes, si necesita de una licencia, la cual es de 8€/mes. Como solo hemos necesitado de ella el último mes, calculamos que las horas dedicadas a la licencia han sido 63 aproximadamente. De esta forma, el precio por hora corresponde a 0.011€ y el precio total según las horas de uso es de 0.7€.

3.3.5. Costes totales del proyecto

Una vez recabados todos los datos correspondientes a los costes originados en el desarrollo del TFG, podemos hacer una suma de todos ellos para obtener la suma global. De esta forma, podremos comprobar el precio que costaría cada hora de desarrollo y el precio total del mismo, según podemos comprobar en la tabla siguiente 3.5.

Material	Coste/hora	Coste total (380h)
Espacio de trabajo	0.625€	237.5€
Internet	0.069€	26.38€
Luz	0.0013€	0.494€
Ordenador	-	800€
Monitor	-	400€
Astah	0.011€	0.7€
Sueldo trabajador	11.25€	4009€
Total	**14.40€**	**5474.07€**

Tabla 3.5: Costes totales del proyecto

3.4. Planificación final

Esta última sección de la planificación se ha realizado una vez terminado el proyecto. En ella se van a comparar los datos de la planificación inicial con los datos obtenidos al final del desarrollo, tanto en horas de trabajo, como en seguimiento de las iteraciones.

Para comenzar, debemos comentar que la estimación de horas de trabajo se ha quedado corta, ya que en un inicio se estimaron 380h y al final del proyecto las horas totales han sido 400h. Esto se ha debido principalmente a dos factores, los cuales son la realización de un mes más de prácticas extraordinarias, el cual no estaba previsto y ha supuesto menos dedicación al TFG de la esperada, y la adición de dos nuevos requisitos importantes a mitad del desarrollo, lo cual ha supuesto más horas de planteamiento y resolución de las previstas. Por suerte, en la planificación inicial sí se tuvo en cuenta el riesgo de una mala planificación (ver en la Tabla 3.1 el riesgo ID-5), que tenía como plan de mitigación usar horas extra tanto de los fines de semana (ver Tabla 3.2 riesgo ID-5), como de los días posteriores al 6 de junio.

Una vez comentado lo anterior, vamos a entrar a hablar más detalladamente de los problemas ocurridos y el impacto que ha tenido en la desviación de horas y semanas respecto a la planificación inicial. Todo ello, se puede apreciar de forma visual en la figura correspondiente a la comparación del número de semanas 3.3 y a la comparación del número de horas 3.4.

En primer lugar, el hecho de realizar las prácticas extracurriculares ha supuesto un mes más de trabajo en las mismas que no se tenía planeado. Esto, sumado al hecho de realizarlas de forma presencial algunos días, también ha supuesto un mayor número de horas perdidas en transporte y estancia. Todo ello, se ha visto reflejado en la planificación final, ya que durante la cuarta iteración, se han tenido que añadir dos semanas más para poder realizar los objetivos marcados en ese tiempo. El número de horas realizadas para dicha iteración no ha variado, simplemente se ha repartido en el tiempo, ya que debido a la carga de trabajo externa de las prácticas, se ha dispuesto de menos tiempo efectivo así como motivación para realizarlas. En segundo lugar, se han añadido dos nuevos requisitos durante la cuarta iteración, lo que ha supuesto un incremento del número de tareas a realizar así como del tiempo. Las

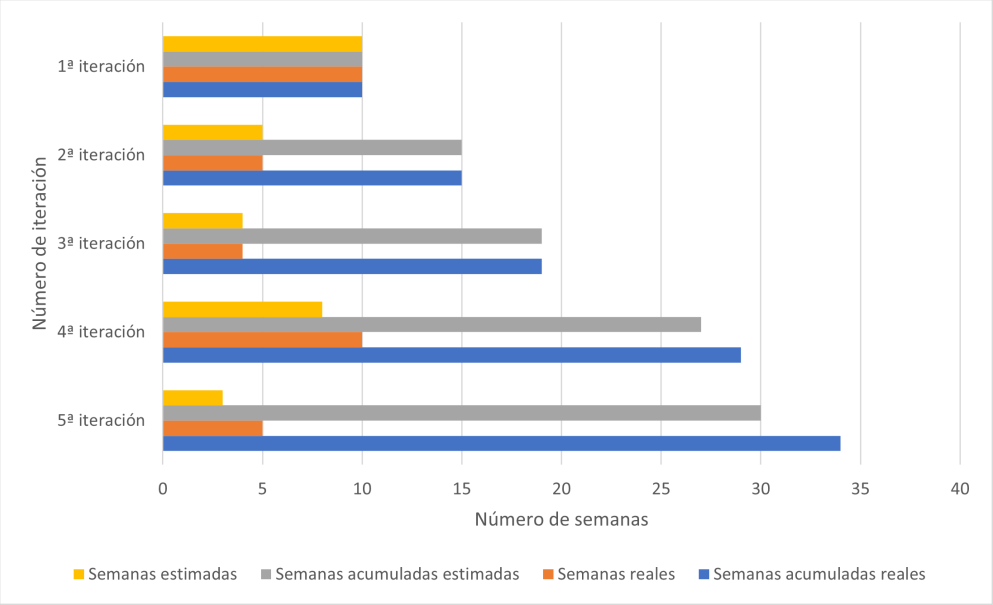

Figura 3.3: Comparación del número de semanas de desarrollo en cada iteración estimadas y reales

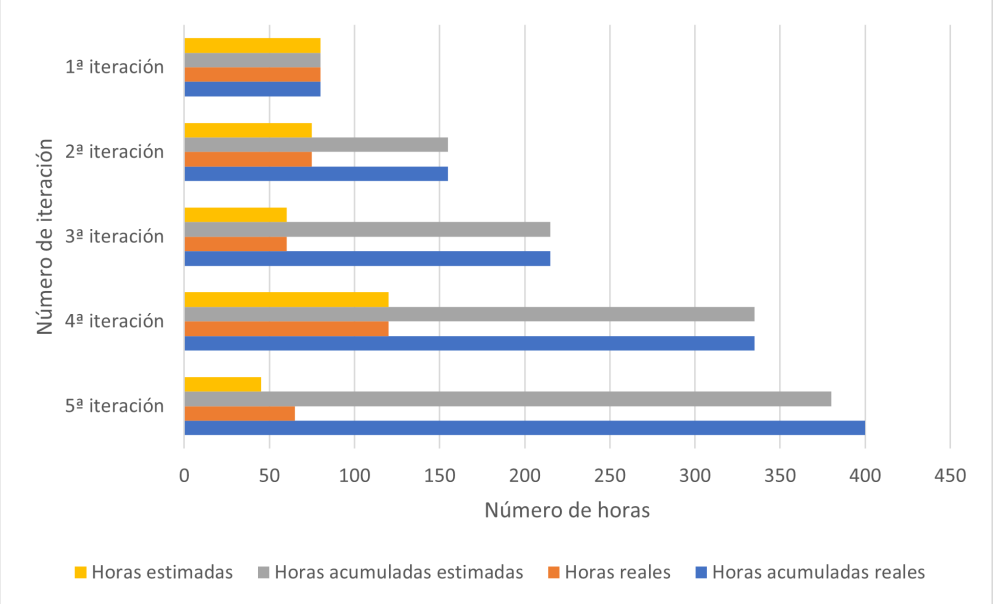

Figura 3.4: Comparación del número de horas de desarrollo en cada iteración estimadas y reales

dos tareas a remarcar son las siguientes:

- Bloqueo de elementos: Se debían bloquear los elementos tanto visual como funcionalmente, cuando se encontraban en medio de una animación, audio o dinámica que les impidiera estar activos. El hecho de plantear su implementación y de realizarla, ha supuesto un incremento en las horas planificadas, las cuales se han visto reflejadas principalmente en la quinta iteración. Parte de las 20 horas extra que se han realizado, han sido para terminar de implementar dicho requisito que había quedado a medias en la iteración anterior.
- Uso de un reconocedor de voz externo: Este nuevo requisito también ha supuesto problemas, ya que a parte de su planificación para implementarlo, también se ha tenido que refactorizar

su implementación inicial. En un primer momento, en la cuarta iteración, se habían usado las 2 semanas extra para poder utilizarlo mediante las bibliotecas de C#. De esta forma, se utilizaban clases del propio lenguaje, las cuales utilizaban un reconocedor de palabras y un diccionario, para detectar las frases que se estaban grabando y disparar una acción si eran las correctas. Como se vio posteriormente, esta solución no era la más adecuada, sobre todo pensando en desarrollos futuros. Por lo que se decidió utilizar las peticiones al servidor como se ha explicado anteriormente. Este proceso, se ha terminado de realizar en la última iteración, por lo que también se han necesitado parte de esas 20 horas extra para poder cumplirlo.

Además de los nuevos requisitos, la planificación de la redacción de la memoria tampoco fue correcta. En un primer momento, se pensó en realizar las partes correspondientes según se desarrollaba el proyecto, y terminar de realizar las partes necesarias una vez se iba avanzando. Debido a los capítulos que han tardado más en desarrollarse y a ciertas refactorizaciones de la memoria que han llevado más tiempo del previsto, también se han terminado de utilizar las 20 horas extra comentadas en la última iteración.

Durante el desarrollo del proyecto se han llevado a cabo un total de 17 reuniones con los tutores, es decir, una cada dos semanas. Siempre se ha realizado una reunión al inicio de cada iteración, para poder hablar en profundidad de las ideas y requisitos a realizar. La cuarta iteración ha sido la que más reuniones ha tenido, ya que ha sido la más extensa y en la que se ha desarrollado un mayor trabajo. Además, a la par que las reuniones, se ha utilizado Telegram de manera más continua para plantear dudas o dar correcciones por parte de los tutores de una forma más rápida y cercana.

Capítulo 4

Descripción de las iteraciones

En este apartado, se va a proceder a explicar el desarrollo completo de todas las iteraciones llevadas a cabo para la realización del proyecto. Entre otros motivos, se va a hablar de los requisitos del proyecto, el diseño de las pantallas, la implementación y generalización de las dinámicas, así como todos los problemas que se han ido encontrando y su solución.

4.1. Iteración 1

Para empezar con la realización del TFG y según lo explicado anteriormente en el apartado de planificación, se puede observar como la primera iteración corresponde a un período de iniciación, sin llegar a producirse una inmersión de lleno en el proyecto. Esto se debe a que se ha empezado a desarrollar el TFG a mediados de primer cuatrimestre del 2021. Pese a ello, no se disponía del tiempo completo para dedicarse al mismo, ya que se estaban cursando tres asignaturas en el grado de Ingeniería Informática en la UVa en paralelo. De esta manera, se han utilizado las horas correspondientes a esta iteración como horas complementarias para posteriormente poder entrar directamente en el desarrollo.
Así pues, el inicio consta del día 28 de octubre de 2021, donde se realizó la primera reunión con los tutores del TFG tras haber acordado la realización del mismo. En ella se expusieron las ideas principales a llevar a cabo, pero de una manera generalizada, por lo que no se fijaron los requisitos en un inicio, sino que se dejarían para la segunda iteración, una vez adquiridos los conocimientos previos necesarios para el desarrollo y una vez superada la convocatoria ordinaria para optar de más tiempo disponible. Por ello, a continuación, se procede a explicar las tareas realizadas en este período.

4.1.1. Tareas realizadas

En primer lugar, tras las primeras reuniones efectuadas, se diseñó y escribió el apartado correspondiente a la planificación inicial. De esta manera se pretendía fijar las horas correspondientes a cada iteración y el trabajo futuro que se iba a realizar. Como es lógico, esta planificación no se ha llevado

a cabo de una manera estricta, ya que debido a que no se habían fijado los requisitos necesarios, se escribió con la idea general que se tenía de desarrollo del proyecto. No fue hasta la siguiente iteración, como se explicará más adelante, cuando se fijaron definitivamente los requisitos y se enseñó el proyecto de Pradia para poder tener un punto de partida a la hora de realizar el juego.

Por todo ello y según lo comentado en el apartado anterior, las horas dedicadas a esta iteración fueron principalmente utilizadas para la documentación, aprendizaje e introducción en el desarrollo de los videojuegos con Unity, así como para la investigación y descarga de herramientas que se iban a utilizar en el proceso. Para ello, en primer lugar, se proporcionó de parte de los tutores un tutorial [41] para tener acceso a una gran cantidad de horas de contenido en el que poder fijarse para mejorar con Unity. Pese a que el tutorial proporcionado, explicaba la creación de un juego en 3D y, por tanto, ciertas dinámicas como las físicas o posicionamiento espacial de los jugadores no se iba a tener en cuenta, si permitía observar diversas funcionalidades implementadas posteriormente en nuestro juego 2D. Entre ellas, destacan la gestión de eventos y creación de items y objetos para un posible inventario. De esta forma se consiguió avanzar de forma notoria con el aprendizaje tanto de C# como del funcionamiento completo de Unity, es decir, como trabajar con su interfaz y proporcionar desde animaciones hasta darles sentido mediante el scripting.

A parte de ello, también se ha utilizado otro tipo de documentación para poder ampliar los conocimientos y emprender el juego en un futuro de una forma más sencilla y directa. Entre la documentación básica utilizada, cabe mencionar la proporcionada por la web oficial de Unity para el manejo de la misma [16], así como la necesaria para poder realizar los scripts [17]. Además, como se ha comentado anteriormente, una de las ventajas que se observan en el desarrollo con este motor gráfico, es la inmensa comunidad que posee. Por tanto, se ha podido aprender de forma complementaria a solucionar problemas mediante foros o personas las cuales ya habían resuelto los mismos problemas particulares que se estaban teniendo en un momento puntual.

Además de los mencionada, cabe destacar el uso de tutoriales externos y gratuitos, encontrados principalmente en forma de video [42] o también como pequeños blogs [43] para complementar el aprendizaje y la documentación. Entre ellos, destacan los que aplicaban conocimientos extra a los aprendidos en el tutorial proporcionado por los tutores. Entre esos conocimientos, se pueden mencionar el aprendizaje para guardar información en ficheros log, la creación de un fichero *gitignore* a la hora de subir documentos al repositorio, la capacidad de generar mejores animaciones, así como un aprendizaje exhaustivo de la distribución de los elementos en la interfaz gráfica de Unity.

4.2. Iteración 2

Para esta iteración, según lo indicado anteriormente, pese a tener ya un contexto general de la aplicación, se ha profundizado completamente en las funcionalidades y detalles del proyecto. Por ello, para comenzar con el desarrollo de esta segunda iteración, se procede a realizar una reunión con el cliente, es decir, en este caso, los tutores del TFG. En ella, se exponen completamente las características del proyecto de manera que se puedan identificar los diferentes requisitos y las correspondientes historias de usuario como veremos a continuación. Para ello se explicó todo detalladamente y se hizo una demo para visualizar el juego y sus funcionalidades, siendo así posible visualizar por primera vez como iba a ser el desarrollo de este. Por ello, vamos a comentar los requisitos vistos a continuación.

4.2.1. Requisitos Funcionales

Para entender los requisitos funcionales que se han determinado, cabe remarcar que son la definición de los servicios que el sistema debe proporcionar, cómo debe reaccionar a una entrada particular, y cómo debe comportarse ante situaciones particulares. Es decir, las funcionalidades que debe ser capaz de realizar la aplicación, las cuales aparecen en tabla 4.1.

ID	Nombre	Descripción
RF01	Obtención de objetos	El sistema deberá permitir al jugador conseguir objetos para guardarlos en el inventario
RF02	Uso de objetos del inventario	El sistema deberá permitir al jugador usar los objetos del inventario clicando sobre ellos
RF03	Interacción con objetos del escenario	El sistema deberá permitir al usuario interactuar con los objetos del escenario clicando sobre ellos
RF04	Salir del juego	El sistema deberá permitir al usuario salir del juego
RF05	Pausar el juego	El sistema deberá permitir al jugador pausar el juego fuera de una dinámica
RF06	Cambiar de escenario	El sistema deberá permitir al jugador salir de un escenario para entrar en otro.
RF07	Dinámica de producción	El sistema deberá tener dinámicas de producción lingüística
RF08	Dinámica de comprensión	El sistema deberá tener dinámicas de comprensión lingüística
RF09	Asistente del jugador	El sistema deberá tener un asistente para guiar al jugador durante el juego
RF10	Refuerzo positivo y negativo	El sistema deberá mostrar refuerzos diferentes si la respuesta es correcta o no
RF11	Interacción con el ratón	El sistema deberá permitir usar el ratón del ordenador para interactuar con el juego
RF12	Reproducción de audio	El sistema deberá poder reproducir sonidos
RF13	Almacenar grabaciones	El sistema debe almacenar las grabaciones realizadas por el usuario
RF14	Animaciones	El sistema deberá tener animaciones e imágenes para ilustrar el juego
RF15	Elementos del juego	El sistema deberá tener elementos gráficos como paneles o botones
RF16	Corrección manual	El sistema deberá permitir al profesor o supervisor evaluar las dinámicas mediante el teclado
RF17	Registro log	El sistema deberá tener un almacenamiento log para registrar los eventos del usuario y las grabaciones
RF18	Lenguaje comprensible	El sistema deberá usar un lenguaje sencillo para textos y audios, facilitando así su comprensión
RF19	Reconocedor	El sistema debe permitir la opción de utilizar un reconocedor para validar las grabaciones
RF20	Desactivar elementos	El sistema debe desactivar los botones y elementos que no se puedan utilizar en las dinámicas, tanto funcional como visualmente

Tabla 4.1: Requisitos funcionales

4.2.2. Requisitos No Funcionales

Para entender los requisitos no funcionales, cabe destacar que son los cuales hacen referencia a las características técnicas que el sistema debe cumplir. Por este motivo, veremos los siguientes requisitos analizados a continuación en la tabla 4.2.

ID	Nombre	Descripción
RNF01	Entorno de desarrollo	El sistema tendrá que ser desarrollado usando Unity 2021.2.9.f1 y C# 9.0
RNF02	Formatos de guardado	El sistema debe permitir que el registro de eventos esté almacenado en ficheros JSON y los audios en formato WAV, stereo
RNF03	Reproducción audios	El sistema deberá usar archivos MP3 o WAV pregrabados para la reproducción de sonido
RNF04	Resolución	El sistema deberá permitir que la aplicación se visualice correctamente en cualquier pantalla de proporciones 16:9
RNF05	Exportación	El sistema deberá permitir usar la aplicación por lo menos en Windows 10 y Linux 16.04 en adelante
RNF06	Conectividad	Una vez descargada la aplicación no hará uso de internet, excepto al utilizar la funcionalidad del reconocimiento de voz

Tabla 4.2: Requisitos no funcionales

4.2.3. Requisitos de Información

Estos requisitos detallan la información que se debe guardar en el sistema para cumplimentar su funcionamiento. Por tanto, en la tabla 4.3, veremos los necesarios:

ID	Nombre	Descripción
RI01	Logs del juego	El sistema debe almacenar en ficheros json los eventos del juego con la fecha, hora, el tipo de evento producido y una posible información adicional
RI02	Logs de dinámicas	El sistema debe almacenar en ficheros json las dinámicas del juego con la fecha, hora, el tipo de evento producido y una posible información adicional
RI03	Logs de grabaciones	El sistema debe almacenar en ficheros json las grabaciones realizadas en las dinámicas
RI04	Tipos de dinámicas	El sistema debe contener las dinámicas de juego de producción y comprensión

Tabla 4.3: Requisitos de información

4.2.4. Funcionalidad realizada

Una vez visualizados los requisitos y con los conocimientos previos de las herramientas adquiridos gracias a la iteración anterior, se empezó a analizar y diseñar el juego, guiándose por la planificación

estimada en un inicio, los requisitos fijados y las competencias propuestas en la reunión realizada con los tutores, para poder posteriormente implementar todo ello.

En primer lugar, se fija el objetivo de empezar a desarrollar las dos primeras pantallas del juego, así como la primera dinámica, es decir, la dinámica de producción. Bajo dos premisas clave, como son la generalización de todos los elementos posibles para hacer una plantilla reutilizable, y el correcto funcionamiento de la dinámica, tanto en reproducción como grabación de sonido, para cualquier escena en la que fuera necesario utilizarse. De esta forma, según se observa en la imagen 4.1, vemos el escenario del primer nivel a realizar en el juego.

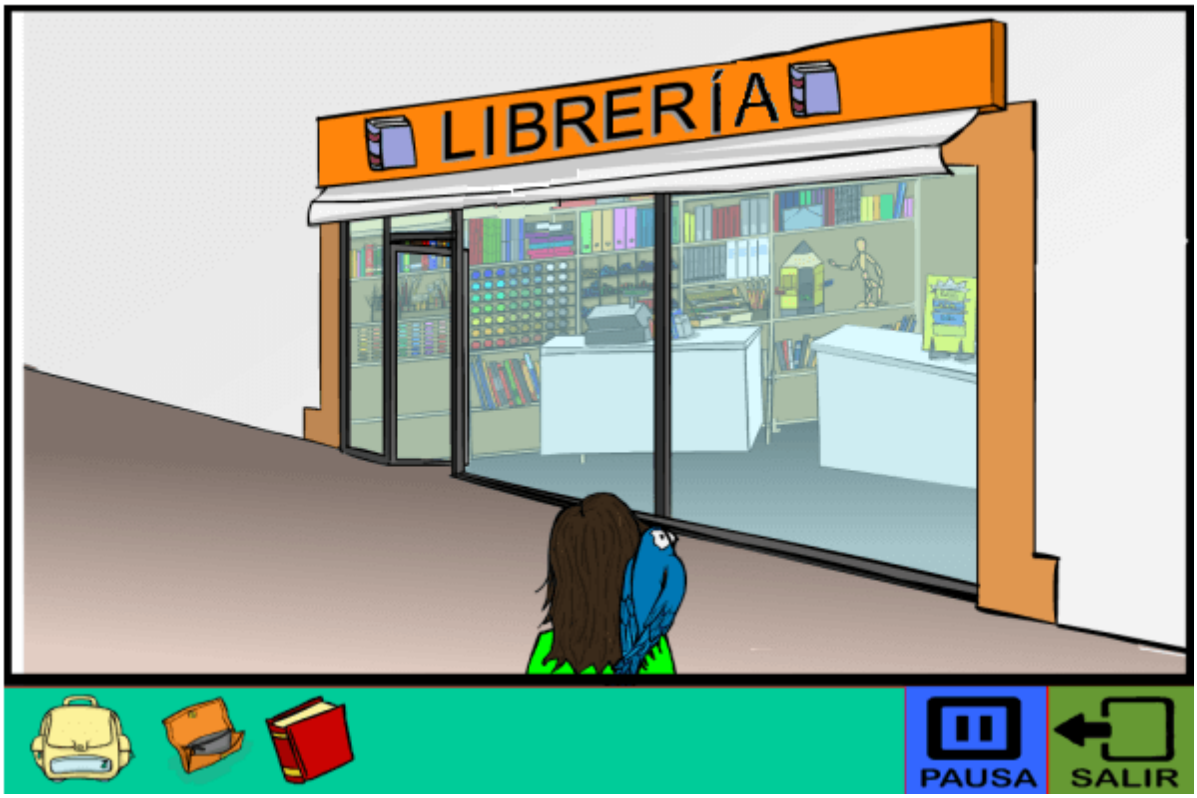

Figura 4.1: Escenario exterior correspondiente al primer nivel

Para esta primera pantalla exterior no ha sido necesario implementar la dinámica comentada anteriormente, ya que esta fue incluida para la pantalla siguiente. Por este motivo, los eventos relacionados con este primer nivel fueron principalmente relacionados con las animaciones y, en menor medida, con la gestión de eventos.

Principalmente se han desarrollado tres tipos de animaciones. En primer lugar, la referente al movimiento del personaje lateralmente para dar una sensación de dinamismo y movilidad, e indicar al usuario que el juego está en funcionamiento y tiene diferentes posibilidades para realizar. En segundo lugar, y a colación del dinamismo mencionado anteriormente, destaca el resaltado de la puerta mediante un cambio de color en su entorno. De esta forma, se consigue llamar la atención del jugador para que interactúe con dicho elemento y no se quede bloqueado en el nivel sin saber que hacer. Por último, destaca la animación de transición, la cual se dispara una vez se selecciona la puerta. En ese

momento, el personaje realiza un movimiento vertical y el escenario se acerca al usuario, consiguiendo así una sensación de movimiento. Para estas animaciones, gracias a la documentación y la preparación previa realizada, se pudieron afrontar de una forma sencilla y eficiente. Por ello, vamos a explicar las funcionalidades básicas realizadas para implementar dichas animaciones desde cero en Unity.

Primeramente, se necesitan añadir los sprites correspondientes al sistema de ficheros de Unity para poder tener una referencia. Una vez añadidos, los sprites proporcionados venían dados, en ocasiones, como una secuencia de imágenes en un solo archivo. Por ello, mediante el editor de sprites de Unity, se necesita separar esos sprites como se ve en la imagen 4.2.

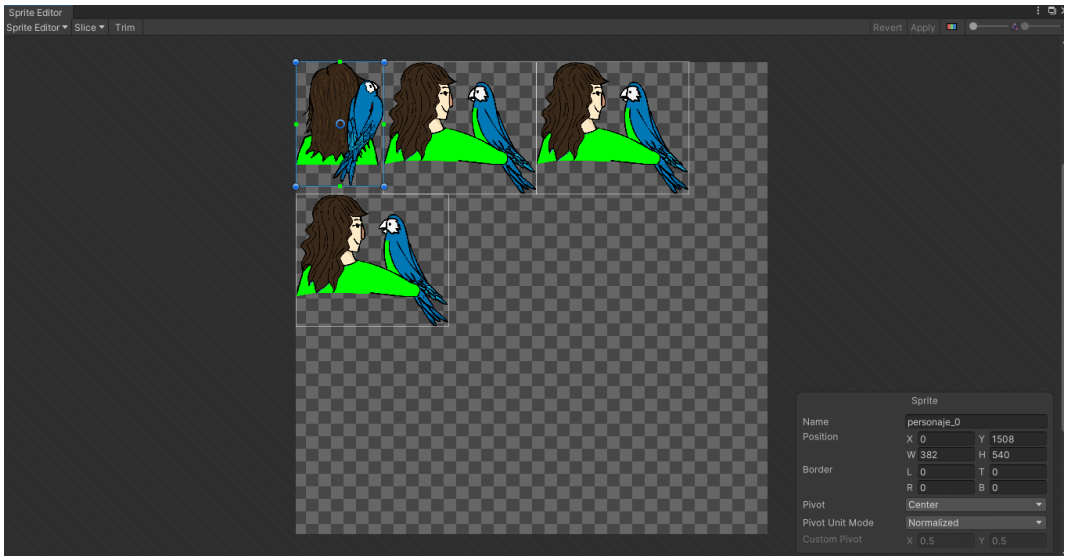

Figura 4.2: Sprite editor

Una vez, separados los elementos y obtenido los sprites por separado si fuera necesario, se procede a acceder al editor de animaciones de Unity para poder animar dichos elementos. Una vez dentro, se crean los estados necesarios para las diferentes animaciones referentes al nivel, en este caso, fueron necesarias la del movimiento lateral del personaje y la del movimiento vertical para cambiar de escenario. Según se ve en la imagen 4.3, se permiten modificar las propiedades que nos interesen en una línea temporal. Para ello, se ha visto necesario modificar la posición del sprite en el espacio según el tiempo, es decir, para obtener una sensación de movimiento según transcurre el paso de los segundos.

Una vez realizados los pasos previos, ya se tienen creadas las animaciones, pero hacía falta darle sentido al flujo y la transición entre ellas. Para eso, Unity facilita un animator donde se visualizan las distintas animaciones creadas como un diagrama de estados, con la posibilidad de transicionar entre ellos como vemos en la imagen mostrada en la Figura 4.4.

Para disparar las transiciones entre los estados y poder cambiar así de animaciones, se realizó la asignación de trigger [44] o de un boolean el cual se disparase en un momento dado, permitiendo así la transición. De esta forma, se asignaron los disparadores a las transiciones entre los estados, para así permitir el cambio de animaciones desde el código cuando se disparase alguno de ellos. Por ello, se completó de esta forma uno de los puntos principales a tratar en este nivel.

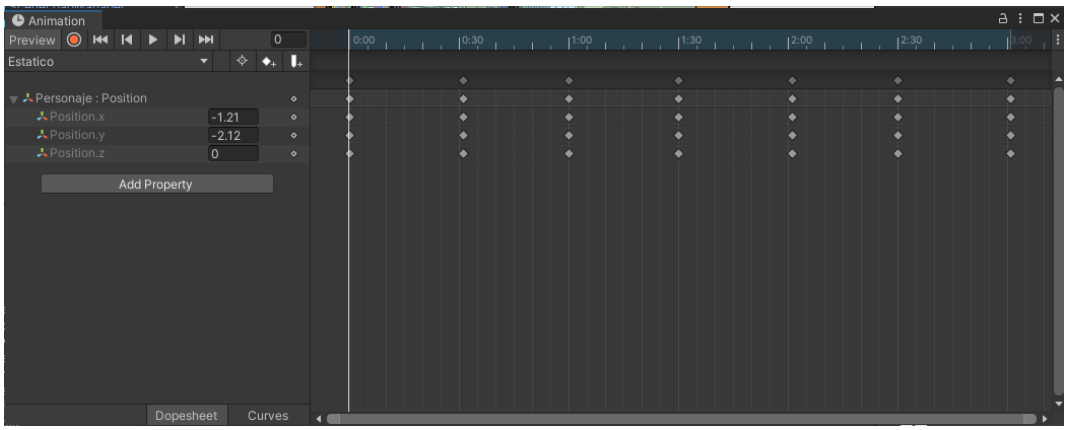

Figura 4.3: Animación del sprite

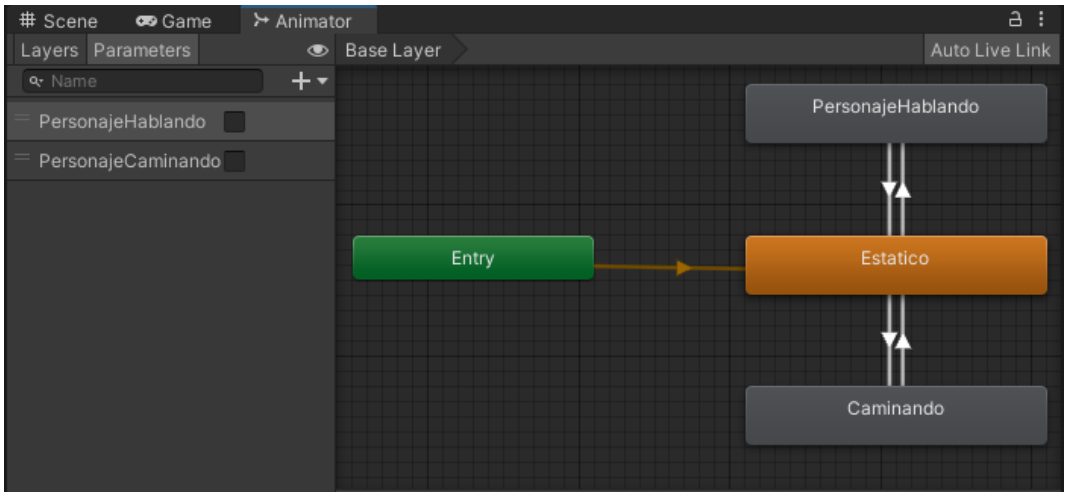

Figura 4.4: Animator Unity

Como se ha comentado antes, también ha sido necesario administrar la gestión de eventos para esta pantalla, como por ejemplo a la hora de realizar una pausa o pulsar sobre la puerta para poder cambiar de escena. Mediante el scripting, se ha conseguido realizar de forma satisfactoria ambas acciones, para dar por finalizado este nivel.

Después de esto, según se ha mencionado antes, se buscaba también empezar a desarrollar la segunda pantalla, la cual contaba con el desarrollo de la dinámica de producción y la necesidad de implementar una plantilla para poder generalizar los eventos realizados en el nivel, para poder ser utilizados indistintamente en cualquier otro. Por tanto, según vemos en la Figura 4.5, se empezó a desarrollar dicha escena a partir de los visualizado y las dinámicas y eventos presentados.

Figura 4.5: Escenario interior correspondiente al segundo nivel

Una vez vista la escena a desarrollar se empezó por la construcción de la misma, añadiendo los elementos necesarios a la interfaz gráfica de Unity, para crear así el escenario y elementos complementarios. Al igual que en el nivel anterior, se realizaron las animaciones que faltaban, como la de simulación de hablar cuando se reproduce un audio, y se reutilizaron las creadas anteriormente, como por ejemplo las que hemos visto para el personaje principal.

Según hemos comentado, se han añadido las animaciones referentes al habla, por lo que también se crearon los primeros scripts en referencia a ellos. Para ello, mediante el editor de Unity y con la ayuda de C#, se ha conseguido integrar dicha funcionalidad. Como vemos en la imagen 4.6, desde el script se permite serializar [45] elementos y poder así arrastrarlos directamente desde la interfaz. De esta manera, mediante el código, se obtienen, por ejemplo, los elementos de audios vistos para poderse reproducir o detener cuando se desee, el sprite del personaje para poder modificar su animación, o las variables que necesitemos para tenerlas al alcance de una forma sencilla y cercana.

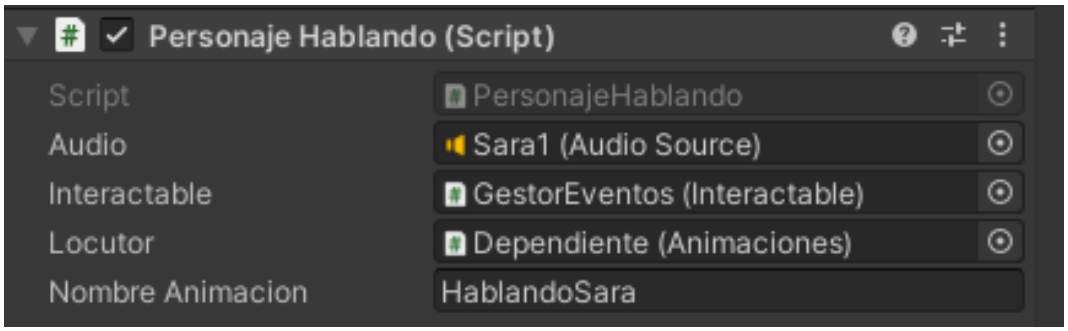

Figura 4.6: Elementos del script modificables directamente desde Unity

Por todo ello, una vez implementados todos los cambios de la iteración, se procedió a realizar el despliegue de la misma. En este caso, mediante el modo que ofrece Unity de correr los proyectos y previsualizarlos. De este modo, y para completar esta iteración siguiendo la metodología propuesta, se realizaron las pruebas pertinentes. Estas, consistían en comprobar que las nuevas implementaciones y cambios realizados fueran satisfactorios, como por ejemplo, los cambios de escenario, cambios de

estado en las animaciones o gestión de eventos correctamente. Así mismo, se empezó a pensar en el desarrollo del script de la dinámica de producción, con las posibles pruebas necesarias para observar su comportamiento correcto. Todo ello no fue implementado en esta iteración, ya que como veremos a continuación, tuvimos que realizarla en la siguiente, debido a la complejidad inicial de la misma, ya que no nos habíamos enfrentado a otra similar con anterioridad.

4.3. Iteración 3

Para esta iteración, se ha planteado integrado todos los cambios necesarios para la creación de los niveles propuestos anteriormente. Por ello, según vamos a ver, se termina de completar la dinámica de producción, así como de generalizarla para que se puede utilizar en cualquier pantalla o escena de forma correcta sin tener que modificar nada del script, únicamente cambiando los elementos correspondientes como pueden ser audios o textos. También, se han concebido ideas necesarias para poder generalizar correctamente el juego, así como diversas técnicas para hacerlo más eficiente y obtener un código más limpio.

4.3.1. Realización de la Dinámica de Producción

Según las ideas preconcebidas en la iteración sobre cómo afrontar el desarrollo de este punto, se puede afirmar que en esta iteración se ha terminado de desarrollar la plantilla de la dinámica requerida. Se busca hacer una generalización de la misma, ya que no en todas las escenas se tiene que comportar de la misma manera. Como se puede observar en la imagen 4.7, nos podemos hacer una idea de cómo es la interfaz de la dinámica a realizar, para poder así realizarla de manera correcta.

Figura 4.7: Ejemplo de escena que contiene dinámica de producción

Así mismo, para tener una idea correcta del funcionamiento de esta dinámica y de su flujo se ha utilizado el documento explicativo ya proporcionado con anterioridad. Esto se debe a que no todas las dinámicas de producción tienen el mismo comportamiento, pero si la misma estructura. Por ejemplo,

algunas no necesitan refuerzos positivos o negativos según las respuestas del usuario, otras no contemplan la necesidad de añadir un altavoz para poder volver a escuchar la frase. Por ese motivo, se muestra en la Figura 4.8 un pequeño trozo del documento con el flujo del comportamiento, para tener un ejemplo de referencia.


```
{
    "nombreDinamica": "Dinamica3.1",
    "audioIntro": {
      "nombreAudio": "Dinamica3.1_intro",
      "locutor": "lolo"
    },
    "audioAcierto": {
      "nombreAudio": "",
      "locutor": ""
    },
    "numRepeticiones": 2,
    "mostrarRefuerzoNegativo": true,
    "mostrarRefuerzoPositivo": true,
    "reproducirFrase": false,
    "mostrarFrase": false,
    "frasesDificultad": [
      {
        "dificultad": 1,
        "textoAGrabar": "Hola, ¿tienen lupas? Me gustaría comprar una.",
        "audio": {
          "nombreAudio": "Dinamica3.1_1",
          "locutor": ""
        },
        "audioLolo": {
          "nombreAudio": "Dinamica3.1_1L",
          "locutor": "lolo"
        }
      },
      {
        "dificultad": 2,
        "textoAGrabar": "Hola, ¿tienen lupas? Quería comprar una.",
        "audio": {
          "nombreAudio": "Dinamica3.1_2",
          "locutor": ""
        },
```


Figura 4.8: Fragmento del comportamiento de la dinámica de producción

En primer lugar, para desarrollar esta dinámica, se ha echado mano del documento con el comportamiento ya mencionado. Para ello, se han obtenido los elementos claves que debía tener y por tanto han sido implementados. Para ello, mediante el scripting, se han declarado como serializables los elementos clave, para ser así modificados desde la interfaz gráfica de Unity. De esta manera se han podido añadir de forma sencilla audios y elementos importantes como la dificultad, el número de repeticiones de la dinámica o el nombre de la propia dinámica

Una vez obtenidos los elementos con los que íbamos a trabajar, se ha procedido a crear el flujo. Dicho comportamiento consiste en la reproducción de un audio, para posteriormente ofrecer la posibilidad de

grabar un audio referente a una frase que debe ser dicha. Para ello, en este momento, solo se contemplaba la posibilidad de jugar acompañado de un supervisor que validara si la grabación era correcta o no mediante el teclado. Por eso mismo, se mantenía una comprobación por si se disparaba por teclado la tecla correspondiente a la afirmación o negación de la frase grabada. Una vez confirmada la frase o agotados los intentos necesarios para hacerla bien, se podía o no lanzar un refuerzo, ya fuera positivo o negativo según la necesidad.

A la hora de desarrollar el flujo de la dinámica, se tuvo en cuenta el problema que podían suponer la reproducción de los audios para el comportamiento correcto, ya que dependiendo de la duración de estos se debía esperar un tiempo u otro. Para ello, en una primera instancia, se comprobó que suponía una dificultad no prevista, ya que para mostrar paneles o animaciones, se debía esperar un tiempo fijo que variaba dependiendo de cada situación, y que obviamente, no se podía introducir como variable por parte del usuario. De esta forma, para solucionar dicho problema, se hizo uso de las corrutinas [46].

Con esta solución, se consiguió arreglar dicho problema, ya que como se citó antes, una corrutina es una forma especial de hacer que la lógica suceda con el tiempo. De esta forma, se dispara el uso de una corrutina y se espera el tiempo de duración del audio a reproducir. De esta manera, mientras está sonando el audio no se realizan más acciones, ya que no retorna el flujo normal hasta que termina. Una vez finalizada la corrutina, se continua con el flujo y, por tanto, se disparan las acciones necesarias que debían hacerlo tras los sonidos.

4.3.2. Otras funcionalidades realizadas

Una vez realizada la dinámica de producción, la cual se puede ya reutilizar en cualquier momento en la misma escena o en otra totalmente distinta, procedemos a finalizar con las tareas necesarias en la escena actual. Para ello, las funciones en las que un personaje aparece hablando con un audio de fondo se han podido reutilizar. Esto es gracias a la generalización que se ha ido haciendo, como veremos más adelante solo es una pequeña parte de la misma, ya que está compuesta por muchos más factores. Dicha generalización permite que el script del personaje permita recibir la función genérica del mismo, asociada al sprite, así como el audio que va a reproducir, según hemos visto anteriormente en la Figura 4.5. De esta forma se permite modificar la animación del personaje correspondiente y reproducir el audio necesario, todo ello usando un único script para cualquier tipo de personaje o sprite que necesite cambiar de estado.

A parte de estas tareas, se han realizado alguna más innovadora y no vista con anterioridad, como ha sido la gestión de eventos del monedero y la lupa. De esta forma, mediante un procedimiento similar a los previos, se han creado las animaciones correspondientes para dichos elementos, posteriormente se ajustaron los estados y las transiciones entre animaciones, y se generó un script para dar sentido a todo el comportamiento.

Una vez implementado el planteamiento de esta iteración, se ha procedido a desplegar el proyecto, para comprobar que todos los cambios funcionaban correctamente. De igual manera, se han realizado las pruebas pertinentes para comprobar toda la funcionalidad. Podemos encontrar entre ellas principalmente la validación del funcionamiento de la dinámica, comprobando su funcionamiento de manera positiva y negativa, así como los refuerzos necesarios en cada caso.

4.3.3. Ideas para generalizar el código

Por último, se dejaron preparadas para la siguiente iteración una serie de ideas para seguir con el proceso de generalizar el proyecto al máximo, consiguiendo una especie de plantilla del mismo. Como ya se ha comentado, la parte de animaciones se ha podido reutilizar, ya que si los personajes poseen el mismo comportamiento en escenas diferentes, pueden reutilizar dichas animaciones y las transiciones que permiten cambiar entre ellas. También se ha visto como generalizar la forma de diálogo de los personajes, así como conseguir una dinámica de producción válida para cualquier escenario sin modificar el código, solo elementos distintos desde la interfaz gráfica de Unity. Pese a todos estos avances, todavía quedaban varios por realizar, por lo que se explicarán a continuación las ideas concebidas para ello, pero no desarrolladas.

En primer lugar, se ha pensado utilizar un script para poder generalizar la lista de operaciones en una pantalla. De esta manera, poder añadir los distintos scripts a esa lista de tareas para una ejecución en secuencia de estos. La idea es sencilla, poder modificar la posición de ejecución de las tareas desde la interfaz de Unity, permitiendo así modificar el orden de estas, añadir unas nuevas o eliminar otras con un solo clic. Por ello, se ha pensado utilizar el tutorial de Unity visto [41], ya que se visualizó una idea similar a la hora de iniciar con la primera iteración.

En segundo lugar, se ha pensado en la utilización de Prefabs[47], el cual actúa como una plantilla a partir de la cual se pueden crean innumerables instancias a partir de él. De esta manera, se busca reutilizar elementos comunes en muchas escenas, los cuales se deberían crear a mano de nuevo para cada nivel. Esto, como es lógico, supondría un esfuerzo y tiempo innecesario si se lleva a cabo dicha idea, como veremos en un futuro. Esta idea parece eficiente, ya que se observa la repetición de diversos elementos, como pueden ser los paneles de las dinámicas o los inventarios. Además, se plantea la idea de como despejar la interfaz gráfica de la escena en Unity, para tener solo los elementos necesarios y no elementos sobrantes que pueden ser creados en un futuro.

Por tanto, con estas ideas para poder realizar una plantilla del proyecto, y las que ya se han implementado, se pretende conseguir un desarrollo más eficiente del mismo. Su función es poder reutilizar diversos elementos o dinámicas para poder crear niveles nuevos sin casi necesidad de esfuerzo ni conocimientos de programación, ya que todo se realizaría desde el entorno gráfico.

4.4. Iteración 4

En esta iteración, según hemos visto en la planificación, disponemos de mucho más tiempo que las otras. Por eso, ha sido la que más trabajo se ha desarrollado, centrándonos en los objetivos fijados, implementando las ideas que habían surgido en las iteraciones anteriores y refactorizando los errores que han ido apareciendo a lo largo del proyecto.

4.4.1. Requisitos funcionales

A lo largo del desarrollo, y gracias a la metodología que estamos usando, pese a que no sufra apenas modificaciones, si se permite cierta flexibilidad a la hora de modificar los requisitos. De este modo, en la reunión que se tuvo con los tutores, aparecieron dos nuevos requisitos los cuales no estaban fijados con anterioridad. En primer lugar, el referente al reconocimiento de voz. Para ello, solo se había mencionado la opción de poder jugar con un supervisor al lado del usuario para poder validar las grabaciones, siguiendo así el mismo patrón que el juego anterior de Pradia. Pero ahora, se busca la posibilidad de mantener esa opción y complementarla con un reconocedor de voz para las ocasiones que se desee.

Así mismo, en segundo lugar, surge la necesidad de poder bloquear los botones y elementos de las dinámicas cuando no se puede realizar ninguna acción con ellos. Esto es para que el usuario no piense que tiene la opción de interactuar con ellos cuando no es así, evitando confusiones o malos usos de la aplicación. De este modo, cuando no puedan utilizarse, estarán bloqueados tanto funcional como visualmente. Los requisitos comentados, se encuentran en la tabla 4.1.

4.4.2. Nuevas escenas implementadas

En esta iteración, se implementa la idea de realizar dos nuevas escenas en el juego. Según veremos a continuación la estructura es similar a la vista para los niveles anteriores. Consta de una primera escena exterior y otra interior, con la particularidad de que esta última, aparte de utilizar la dinámica de producción comentada anteriormente, debe crear e implementar una nueva dinámica de comprensión.

En primer lugar, la primera escena se va a construir siguiendo la idea de la interfaz que tenemos en la imagen 4.9. Como se puede observar, la estructura es similar a la del nivel exterior visto anteriormente, es decir, solo será necesario incluir o reutilizar las animaciones correspondientes y gestión de eventos mínimo, como la de pausar el juego o cambiar de escena clicando la puerta.

Para ello, la construcción de la pantalla fue más sencilla que la anterior, ya que además de tener la experiencia previa, se han podido reutilizar múltiples elementos. Esto facilita su creación y nos ayuda a comprender la necesidad de generalizar el juego para que funcione con plantillas, de esta forma pudien-

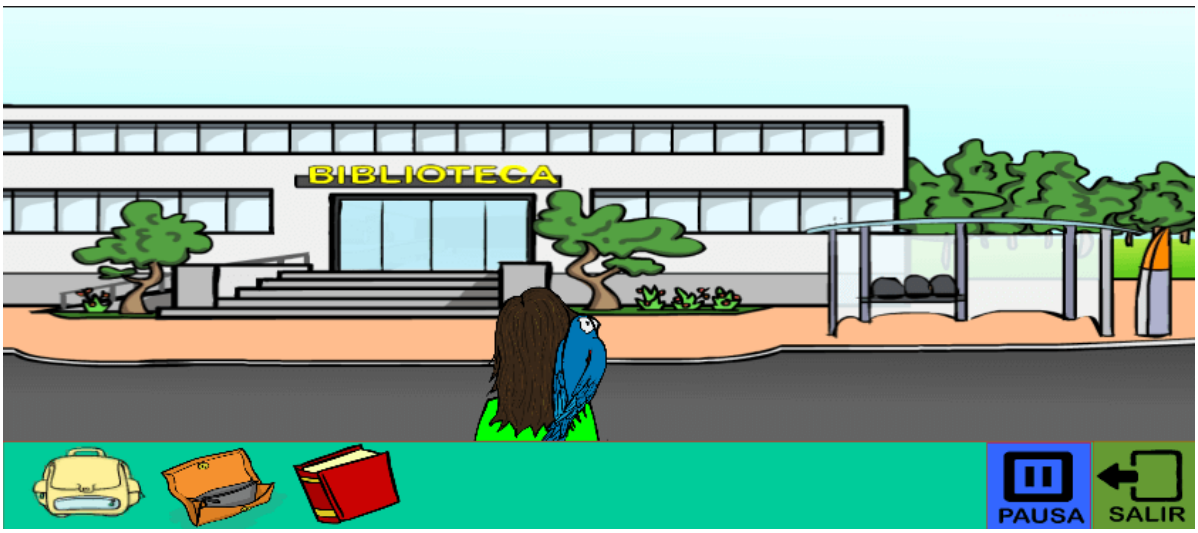

Figura 4.9: Interfaz del tercer nivel, correspondiente al exterior de la biblioteca

do crear niveles como este de forma rápida y fácil. De este modo, se han reutilizado las animaciones del personaje principal para dar sensación de movimiento, y, por otra parte, se ha aprovechado la ya existente de transición del fondo para crear la nueva transición idéntica, pero modificando el sprite del fondo.
En cuanto a la gestión de eventos necesarios, tanto la pausa como el cambio de escena, ya estaban creados de la pantalla anterior, por lo que también se han reutilizado para facilitar la creación de esta. Todo ello, como hemos comentado antes, avala el uso de la reutilización que buscamos.

En segundo lugar, la segunda escena nueva que se busca implementar también es similar a la escena interior vista antes. Para construirla hemos realizado la interfaz que podemos ver en la figura 4.10. Como nos ha pasado antes, para crearla, se han podido reutilizar animaciones y scripts de su escena homónima.

Figura 4.10: Interfaz del cuarto nivel, correspondiente al interior de la biblioteca

Según se observa hay una estructura reutilizable, por lo que al igual que con la pantalla anterior, se han podido reutilizar todas las animaciones del personaje principal, siendo solo necesario animar los

nuevos sprites del otro personaje. Así mismo, esta escena repite comportamientos de su homónima, como por ejemplo cuando un personaje mantiene un diálogo o la propia dinámica de producción. Para ello, solo ha sido necesario añadir los scripts directamente y modificar los elementos de audio o importantes necesarios, pero todo ello desde la interfaz gráfica de Unity. Todo esto, sigue confirmando un correcto uso de la generalización que se buscaba, ya que la realización de todo este proceso ha sido ínfimamente menos costosa que la anterior, gracias a la reutilización de todos estos elementos. Como nuevo apunte, según veremos a continuación, la dificultad de esta escena residía en la creación de la nueva dinámica de comprensión.

4.4.3. Realización de la dinámica de Comprensión

Para la realización de esta dinámica, se tiene en cuenta como se afrontó la dinámica de producción, ya que aunque no tiene rasgos específicos comunes, si compartes ciertas resoluciones a la hora de afrontar los problemas. Por ello, se busca realizar una generalización de la misma, para que pueda ser utilizada indistintamente en cualquier nivel, modificando simplemente elementos desde el visor de Unity, y no modificando el script. Para ello, podemos comprobar a continuación la semejanza que debe tener dicha dinámica (ver en la figura 4.11).

Figura 4.11: Dinámica de comprensión

De esta forma, se pueden apreciar a simple vista las diferencias con la otra dinámica desarrolladas. El funcionamiento principal de esta consiste en escuchar un audio para realizar una acción. Dicha acción se refleja en dos paneles, los cuales contienen una misma frase, pero con contextos diferentes según como se emplee. Por ello, se permite seleccionar uno de los paneles con la opción que se intuya correcta. Por ejemplo, para esta dinámica, no se incluyen en una primera instancia los refuerzos, ya que, si se responde una serie de veces mal, se bloquea la opción incorrecta para poder seleccionar la adecuada. A diferencia de la dinámica de producción, esta no tiene la opción de grabar un audio y comprobar si se ha realizado correctamente, ya que solo dispone de una opción interactiva mediante el clic.

Así mismo, para tener una idea correcta del funcionamiento de esta dinámica y de su flujo se ha utilizado el documento explicativo ya proporcionado con anterioridad. Esto se debe a que no todas las dinámicas de comprensión tienen por qué tener el mismo comportamiento, pero si la misma estructura, para mantener su funcionalidad característica, pero pudiendo modificar ciertos elementos. Por ese motivo, se muestra en la figura 4.12 un pequeño trozo del documento con el flujo del comportamiento, para tener un ejemplo de referencia.

En primer lugar, para desarrollar esta dinámica, se ha utilizado el documento con el comportamiento ya mencionado. Se han obtenido los elementos claves que debía tener y, por tanto, han sido implementados. Para ello, mediante el scripting, se han declarado como serializables los elementos clave, para ser así modificados desde la interfaz gráfica de Unity. De esta manera, se han añadido de forma sencilla audios y elementos importantes como dificultad, numero de repeticiones...
Una vez obtenidos los elementos con los que vamos a trabajar, se ha procedido a la creación del flujo. Para ello, su comportamiento ya explicado antes, consiste en seleccionar la opción correcta entre dos posibilidades según un audio de referencia. Además, para completar una explicación visual que resulte más sencilla para los usuarios, los elementos se resaltan o apagan según se mencionan en la explicación. Esto ha supuesto un pequeño problema, ya que no es posible utilizar un tiempo determinado de los audios mediante corrutinas como se hizo anteriormente, ya que para la duración de un solo audio se han de activar y desactivar varios elementos, por lo que no se puede tomar la duración como referencia. Debido a que los elementos se activan en puntos concretos de la explicación, se ha optado por reutilizar la idea de las corrutinas, pero en este caso, siguiendo un tiempo fijo de espera entre el resaltado de los elementos. Esto se debe a que dicho audio de explicación será común para todas las escenas que usen esta dinámica, por lo que no supone un problema ni habrá que modificarlo para los distintos casos particulares que la usen.

Por tanto, una vez explicado la realización de la parte de ayuda auditiva, queda comentar como se ha realizado la interacción con las respuestas. Para ello, es el botón que contiene el panel quien avisa a la dinámica cuando se ha pulsado, y es esta quien gestiona si es correcta o no, a diferencia de la dinámica anterior, quien comprobaba de forma constante la entrada por teclado para validar la respuesta. Dependiendo del número de intentos que llevemos, se desactiva la opción correcta o no, dejando la posibilidad de marcar solo la correcta si se ha superado el límite de intentos, según se observa en la imagen 4.13.

De esta forma se ha conseguido implementar la idea de esta nueva dinámica, pero reutilizando ciertos elementos o funcionalidades para los mismos problemas que surgieron con la anterior, como serializar los elementos, usar corrutinas...

4.4.4. Implementación de la generalización pendiente

Según comentamos en la sección anterior, se busca conseguir una generalización para las dinámicas y diversos comportamientos del juego, siendo así muy sencillo crear nuevos niveles. Se busca por

```
{
  "nombreDinamica": "Dinamica2.1",
  "audioIntro": {
    "nombreAudio": "Dinamica2.1_intro",
    "locutor": "lolo"
  },
  "audioAcierto": {
    "nombreAudio": "Dinamica2.1_acierto",
    "locutor": "lolo"
  },
  "opcionCorrecta": 1,
  "bloquearOpcionesIncorrectas": true,
  "mostrarRefuerzoNegativo": false,
  "mostrarRefuerzoPositivo": false,
  "opciones": [
    {
      "opcion": 1,
      "nivelesDificultad": [
        {
          "dificultad": 1,
          "frase": "Busco libros de historia",
          "audio": {
            "nombreAudio": "Dinamica2.1_opcion1.2",
            "locutor": ""
          }
        },
        {
          "dificultad": 2,
          "frase": "Busco libros de historia",
          "audio": {
            "nombreAudio": "Dinamica2.1_opcion1.2",
            "locutor": ""
          }
        },
        {
          "dificultad": 3,
          "frase": "Busco libros de historia",
          "audio": {
```


Figura 4.12: Fragmento del comportamiento de la dinámica de comprensión

tanto crear una especie de plantilla para poder reutilizar todos los elementos necesarios y en las nuevas escenas crear todo el comportamiento a partir de la interfaz gráfica de Unity, arrastrando y añadiendo los elementos necesarios al panel de control o al inspector correspondiente. De esta forma, ya se han

Figura 4.13: Límite de intentos superado

comentado ciertos avances ya implementados, aunque han quedado ideas en el aire por realizar, las cuales han sido hechas en esta interacción.

En primer lugar, se ha implementado el script para poder generalizar la lista de operaciones en la pantalla. De esta manera y como se aprecia en la imagen posterior 4.14, disponemos de un script para poder añadir nuestras dinámicas y acciones en el orden que queramos de forma manual y directa desde Unity. De esta manera, si deseamos añadir nuevas dinámicas a nuestra escena o eliminar las que sobren, solo debemos modificarlo desde dicho panel, ya que se ejecutan de forma secuencial.

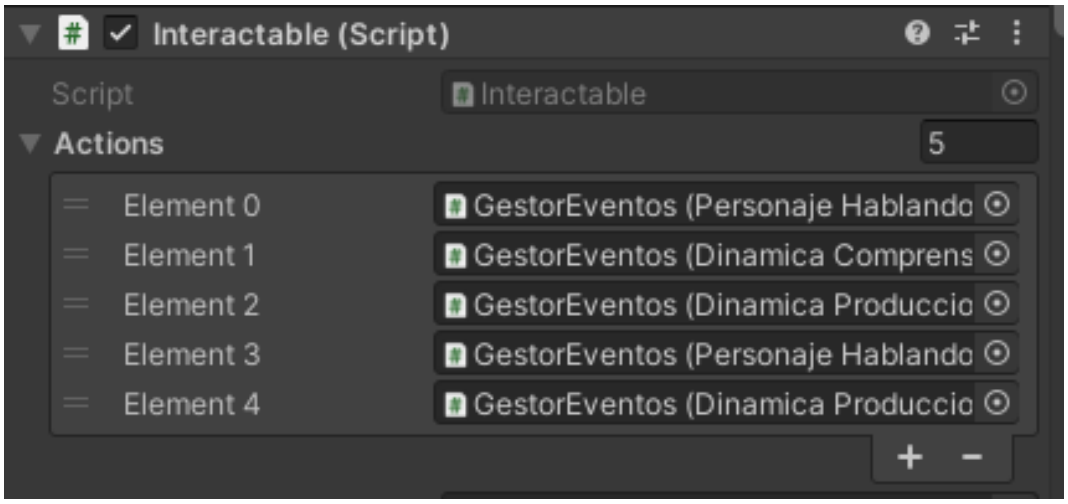

Figura 4.14: Función para ejecutar tareas en orden

El comportamiento de este script es sencillo. Todas las acciones que se introducen deben implementar una clase la cual contiene el método Act(). De este modo, el script recorre la lista de tareas que debe hacer y las inicia ejecutando dicha función. Una vez ejecutada, empieza el flujo principal de esa tarea o dinámica, y una vez finaliza, ella misma avisa que ha terminado para poder seguir ejecutando el resto de las iteraciones de la lista. Así pues, según se observa en el fragmento de código posterior 4.15, se observa cómo se ha generalizado dicho comportamiento para poder ser reutilizable en cualquier escena.

```
private void Update()
{
    while (i < actions.Length && siguienteAccion)
    {
        siguienteAccion = false;
        actions[i].Act();
        i++;
    }
}
```

Figura 4.15: Método para el flujo de tareas

En segundo lugar, se ha implementado completamente el uso de Prefabs comentado con anterioridad. Esta plantilla, permite instanciar los elementos que se desee de la misma generalización en cualquier escena, todos ellos con las mismas características. Así pues, se ha conseguido reutilizar varios elementos comunes en distintas pantallas ahorrando así esfuerzo innecesario en volver a crearlos de cero. Por ejemplo, los elementos más claros pueden ser los inventarios de las diferentes escenas, los cuales tienen una serie de elementos comunes, por lo que resulta eficiente hacerlo, ya que simplemente con arrastrar el Prefab a la escena, conseguimos toda su funcionalidad. Otra plantilla posible para crear es la de los personajes, sobre todo el protagonista, el cual se repite en todos los niveles. Si creamos una plantilla del mismo, podemos reutilizarlo de manera muy simple y rápida.

Todo esto supone una ventaja, pero además soluciona un problema existente. Antes, cuando se ejecutaban varias dinámicas en una escena, tenían elementos comunes como paneles, por ejemplo. Al utilizar estos elementos directamente de la escena y no creándolos dinámicamente, se originaban problemas de funcionamiento en la dinámica, ya que se mezclaban los paneles de ambas dinámicas o se solapaban comportamientos. La solución más factible que se ha visto oportuna realizar, fue la de crear Prefabs para estos elementos comunes de las dinámicas y crearlos o eliminarlos según se utilizaban o se acababa la dinámica, es decir, mediante la creación dinámica de instancias [48]. De esta forma, se consigue eliminar dicho problema. Mediante los Prefabs se crean los elementos comunes a estas dinámicas que pueden crear inconvenientes, como los son los paneles con todos sus elementos o los refuerzo. De esta forma, una dinámica instancia dichos Prefabs de forma dinámica cuando necesita utilizarlos, pero cuando ha acabado con su uso pasa a destruirlos. Esto se puede apreciar directamente desde el panel gráfico de Unity, donde se ven los objetos aparecer y desaparecer según instanciación. Esto, además, permite una mejor visualización de los elementos que tenemos en la escena, ya que al limpiar todos ellos que serán instanciados posteriormente, solo nos quedamos con los que importan para la creación de la propia escena.

Por todo ello, con la implementación de las nuevas generalizaciones preconcebidas y las ya implementadas, se consigue crear una generalización casi completa del juego, realizando de esta forma una plantilla reutilizable entre las diferentes escenas. Esto, nos da la razón en cuanto a uno de los objetivos principales que se buscaba, es decir, conseguir ahorrar tiempo y mantener la funcionalidad mediante la

reutilización de los elementos o scripts. Así pues, a partir de este momento, se pueden crear los niveles que se deseen, añadiendo las dinámicas o comportamientos ya existentes sin necesidad de modificar nada. Todo ello con la facilidad de poder realizarlo desde el entorno gráfico de Unity, por lo que no sería necesario programar comportamientos ya existentes, solo nuevas funcionalidades pendientes o animaciones que no hayan aparecido todavía.

4.4.5. Implementación correcta en el guardado en Logs

En las secciones anteriores, se había comentado, que dos requisitos importantes eran guardar tanto las grabaciones realizadas por el usuario, como los distintos eventos que se debían recoger. Pero ninguno de los dos estaba implementado de forma correcta como veremos a continuación, por lo que se ha visto necesario en esta iteración implementarlos, para cumplir así los requisitos necesarios.

Para el guardado de las grabaciones en logs, se ha implementado de cero, ya que en la dinámica de producción, que es donde se realiza, solo se hacía grabaciones ficticias ya que no se almacenaban en ningún fichero. De esta forma, cuando se activa la opción de grabar el audio para la dinámica, se detecta el dispositivo con el que se está intentando realizar dicha operación y se almacena el contenido que se está grabando en un clip de audio. Cuando se termina de grabar, nos apoyamos de una clase complementaria a la que se le pasa el clip y lo guarda en la ruta indicada con el formato indicado. Para ello, el nombre de la ruta debe seguir un patrón, para ser más fácil de manipular, estando formado de esta manera por la fecha y la dinámica desde la que se ha realizado la grabación.

Para el correcto almacenamiento de los eventos importantes ocurridos en las escenas, se ha procedido a mejorar su implementación y calidad del código. Para ello se dispone de una clase GameData, la cual almacena los datos principales que se han de guardar en cada registro de eventos, como son la fecha, el tipo de evento o datos extras. De esta manera, otra clase complementaria que hace de escritor, almacena todo el comportamiento para guardar los elementos en el formato correcto y con el nombre adecuado. De esta forma, cada vez que se tenga que almacenar un evento desde cualquier sitio, se llama a dicha clase y se le proporcionan los elementos que necesita, como son el tipo de evento y desde que dinámica se produce. A continuación, se puede observar una imagen 4.16 sobre el guardado de estos eventos en el fichero log.

4.4.6. Uso de un reconocedor de voz

La posibilidad de integrar un sistema de interacción con un reconocedor automático de voz mediante una API estaba contemplada en unos de los requisitos funcionales de este TFG. En una primera instancia se vio correcto usar la API de Google para esto, pero debido a que era de pago ofreciendo solo un período de pruebas gratuito, se buscaron otras soluciones.

```
"fraseAGrabar": "Sí, la necesito. ¿Cuánto vale?",
"dinamica": "Dinamica3.2",
"dificultad": "Medio",
"eventos": [
    {
        "fecha": "10/2/2022",
        "evento": "Entrada en la dinamica",
        "hora": "21:7:4:733"
    },
    {
        "fecha": "10/2/2022",
        "evento": "Comienzo grabación",
        "hora": "21:7:11:393"
    },
    {
        "fecha": "10/2/2022",
        "evento": "Grabacion correcta ",
        "hora": "21:7:12:756",
        "infoAdicional": "0;0;0;0;0;0;1;"
    },
    {
        "fecha": "10/2/2022",
        "evento": "Fin grabación",
        "hora": "21:7:12:821",
        "infoAdicional": "10-2-2022_21-7-11-393_Dinamica3.2_1.wav"
    },
    {
        "fecha": "10/2/2022",
        "evento": "Salida de la dinamica ",
        "hora": "21:7:15:842",
        "infoAdicional": "Superada"
    }
```


Figura 4.16: Fragmento de guardado en JSON

Para esta iteración, se ha utilizado un KeywordRecognizer [49], una clase proporcionada por Unity, en vez de una API externa que compruebe las grabaciones, lo cual veremos en un futuro que es la opción utilizada. Para ello, dispone de un diccionario al cual se le pueden añadir todas las palabras o frases que se deseen. De esta forma, se añaden los textos necesarios para grabar en cada dinámica y se almacenan en dicho diccionario. Por otra parte, posee un reconocedor de palabras el cual empieza a funcionar según se indique. Cuando este reconocedor detecta una frase que corresponde al diccionario que tiene, procede a disparar la función correspondiente, en nuestro caso, la de validación del texto bien pronunciado. Como veremos más adelante, esta propuesta fue sustituida por una más eficiente, basadas en el uso de un servidor y un API externa.

Una vez implementados todos los requerimientos vistos en esta iteración, se ha procedido a desplegar el proyecto, para comprobar que todos los cambios funcionaban correctamente. De igual manera, se han realizado las pruebas necesarias para comprobar toda la funcionalidad. Podemos encontrar

entre ellas la validación del funcionamiento de la dinámica de comprensión, comprobando a fondo su funcionamiento. De igual manera, se ha probado el correcto guardado en los ficheros, el funcionamiento del reconocedor o la correcta generalización del proyecto.

4.5. Iteración 5

Para esta iteración, según la planificación, se había reservado el tiempo necesario para corregir los posibles fallos que han podido surgir, así como finalizar la memoria de este TFG. Esto ha llevado más tiempo del previsto, ya que la memoria se ha rellenado de forma más intensiva una vez terminado el desarrollo del juego, por lo que el tiempo para hacerla no ha sido medido correctamente. Por otra parte, también han surgido más problemas y refactorizaciones de lo esperado, como la nueva implementación del reconocedor o funcionalidades incompletas del código.

4.5.1. Corrección de errores

Según se ha comentado, hemos utilizado esta iteración para refactorizar todos los errores que se habían divisado, así como implementar alguna funcionalidad que no estuviera completa. En nuestro caso, destacan algunas que comentaremos a continuación.

En primer lugar, para el correcto guardado de los eventos en los ficheros log, ha sido necesario hacer algunas modificaciones. Por ejemplo, a la hora de guardar los ficheros con el nombre indicado, se ha tenido que modificar la forma de hacerlo, ya que no se estaban guardando según indicaba el estándar. Además, para realizar buenas prácticas a la hora de realizar el código, se ha necesitado la creación de la clase que contuviera un diccionario [50]. Este diccionario, tiene la funcionalidad de un HashMap, es decir contiene un par clave valor, el cual almacena dos cadenas de texto, una de ellas con la descripción que se guardará en el log, y otra con el texto sintetizado que se usará en las distintas clases a la hora de referencia para guardar.

En segundo lugar, uno de los nuevos requisitos no estaba implementado todavía, por lo que ha sido necesario desactivar los elementos necesarios tanto funcional como visualmente. De esta manera, en las dos dinámicas o incluso los botones para pausar el juego o salir de él, deberían carecer de funcionalidad y resaltado cuando no se pueden utilizar. Mediante dos funciones de activación y desactivación de los mismos generadas en las dinámicas, se ha conseguido el objetivo que estaba fijado. Además, se buscaba que la apariencia del ratón tampoco se modificara por la de una mano cuando no estaban activos, para evitar así confusiones de un posible uso. De esta forma y desactivando también esta posibilidad, se ha completado el correcto funcionamiento del requisito.

En tercer lugar, ha sido necesario implementar las pruebas unitarias correspondientes para la realización del punto de pruebas posterior. En un primer momento, no se habían visto convenientes realizar dichos test, ya que con los de usabilidad se creía suficiente. Pese a ello, y como este tipo de test ya estaban desarrollados en el TFM realizado por Mario Corrales sobre Pradia, no se ha visto conveniente volver a realizar. Para ello, gracias a la ayuda de la documentación necesaria [51] se ha visto como crear dichos test y como implementarlos. Este apartado se comentará más detalladamente en el capítulo 6, por lo que aquí cabe mencionar simplemente su uso, para probar tanto su correcto funcionamiento, como para poder demostrarlo en esta memoria.

En cuarto lugar, ha sido necesario completar la funcionalidad del reconocedor. Como vimos anteriormente, se había realizado mediante las bibliotecas internas de C#, pero este método no ha convencido, ya que si se migra a otro lenguaje es normal que se pierda dicha funcionalidad. Por ello, se ha optado por utilizar el reconocedor de voz online privado ofrecido por el CLST, de Radboud University [52]. Para ello, se ha migrado el código desarrollado en python a C#, para poder implementar las mismas funcionalidades. De este modo, mediante llamadas http, se han enviado peticiones a dicho servidor para poder cumplir con las llamadas necesarias. En este caso son las siguientes:

- Login: Para poder validar el usuario, obtener el token y realizar el resto de las operaciones necesarias.
- Subir un archivo de audio: De este modo, se proporciona el audio que queremos comprobar con el reconocedor.
- Decode: Decodificamos el audio para poder verificar si la frase ha sido grabada correctamente o no.

Una vez implementado el planteamiento visto en la iteración, se ha procedido a desplegar el proyecto, para comprobar que todos los cambios funcionaban correctamente y se habían conseguido solucionar los problemas previos. De igual manera, se han realizado las pruebas necesarias para comprobar toda la funcionalidad. Podemos encontrar entre ellas la validación del requisito referente a la desactivación de elementos, así como del correcto funcionamiento del reconocedor. De igual manera, se ha probado que los datos que se almacenan en los ficheros log fueran correctos.

4.5.2. Finalización de la memoria

Por otra parte, según lo comentado, se ha reservado un especial enfocado a la redacción de la memoria, para poder finalizarla. En las anteriores iteraciones se ha ido avanzando con los apartados correspondientes a la introducción, estado de la cuestión, planificación y desarrollo de las iteraciones según se iban completando. Por otra parte, los apartados más avanzados, como las últimas iteraciones, el estado final de la aplicación, pruebas o conclusiones no podían realizarse hasta haber completado el juego o tenerlo en un estado suficientemente avanzado. Por ello, en este período se han completado

los apartados que restaban y se han remodelado algunos apartados o ideas a partir de las correcciones propuestas. De esta forma y tras varias reuniones con los tutores, se completa la memoria del presente TFG, complementando la finalización del código requerido con la redacción de la memoria.

Capítulo 5

Estado final de la aplicación

A lo largo de este capítulo, se tratarán los temas relacionados con el análisis y diseño de la aplicación, mostrando tanto diagramas como patrones o técnicas utilizadas.

5.1. Estado final de la aplicación

En esta sección se va a trata el flujo de las llamadas en la aplicación, las historias de usuario, el modelo de dominio y los diagramas de actividad de las historias de usuario obtenidas.

5.1.1. Historias de usuario

Una de las opciones para representar los requisitos son las historias de usuario. Una historia de usuario es una pequeña descripción de una necesidad, una característica o un deseo desde el punto de vista de un rol de usuario específico del software, además de contener una explicación de su importancia, su beneficio para el usuario y sus criterios de aceptación. Al estar redactados en un lenguaje coloquial, se hacen más fáciles de entender por parte de los clientes y desarrolladores del software y son más flexibles ante los cambios de requisitos que se vayan generando durante el desarrollo del proyecto. A continuación, se muestran las historias de usuario generadas:

ID	HU01
ID RF relacionado	RF01
Nombre	Conseguir objetos
Prioridad	Alta
Riesgo	Bajo
Descripción	Como usuario, quiero poder conseguir objetos en el juego para guardarlos en el inventario
Validación	- No debo poder guardar ningún objeto que no sea almacenable en el inventario para poder usarlo posteriormente

Tabla 5.1: Historia de usuario HU01

ID	HU02
ID RF relacionado	RF02
Nombre	Utilizar objetos del inventario
Prioridad	Alta
Riesgo	Medio
Descripción	Como usuario, quiero poder usar los objetos de mi inventario para interactuar con el juego.
Validación	- Quiero poder ver resaltados los objetos que tengo que utilizar en el momento de su uso - No debo poder interactuar con un objeto si el juego se encuentra en medio de una animación, audio o dinámica

Tabla 5.2: Historia de usuario HU02

ID	HU03
ID RF relacionado	RF03
Nombre	Interactuar con los objetos del escenario
Prioridad	Alta
Riesgo	Medio
Descripción	Como usuario, quiero interactuar con los objetos del escenario para producir resultados en el juego.
Validación	- Quiero que los objetos se resalten cuando pueda interactuar con ellos - No debo poder interactuar con un objeto si el juego se encuentra en medio de una animación, audio o dinámica

Tabla 5.3: Historia de usuario HU03

ID	HU04
ID RF relacionado	RF04
Nombre	Salir del juego
Prioridad	Medio
Riesgo	Bajo
Descripción	Como usuario, quiero poder salir del juego cuando sea posible.
Validación	- No debo poder salir del juego un escenario si el juego se encuentra en medio de una dinámica, audio o animación

Tabla 5.4: Historia de usuario HU04

ID	HU05
ID RF relacionado	RF06
Nombre	Cambiar de escenario
Prioridad	Alta
Riesgo	Medio
Descripción	Como usuario, quiero pasar de un escenario a otro para avanzar en la historia del juego.
Validación	- Quiero que se resalten las salidas de un escenario cuando pueda salir de él. - No debo poder salir de un escenario si el juego se encuentra en medio de una animación, audio o dinámica

Tabla 5.5: Historia de usuario HU05

ID	HU06
ID RF relacionado	RF10, RF13, RF12
Nombre	Dinámica de producción
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder realizar la dinámica, a partir de la grabación de frases, reproducción de las mismas y observar los resultados obtenidos
Validación	- Quiero poder iniciar la grabación cuando esté preparado para hablar - Quiero que la frase a grabar varíe en función del nivel de dificultad del juego - No debo poder iniciar la grabación si se está reproduciendo un audio - Quiero poder ver directamente cuando puedo escuchar una frase. - Quiero ver un resultado diferente para la realización de una dinámica correcta o incorrectamente

Tabla 5.6: Historia de usuario HU06

ID	HU07
ID RF relacionado	RF08, RF10, RF12
Nombre	Dinámica de comprensión
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder realizar la dinámica, a partir de la selección de frases, reproducción de las mismas y observar los resultados obtenidos
Validación	- Quiero que solo una de las frases disponibles sea la correcta. - Quiero poder saber cuándo puedo o no puedo seleccionar o escuchar una frase. - Quiero que las frases disponibles varíen en función de la dificultad del juego. - No debo poder elegir una frase si se está reproduciendo un audio. - Quiero ver un resultado diferente para la realización de una dinámica correcta o incorrectamente

Tabla 5.7: Historia de usuario HU07

5.1.2. Modelo de dominio

En la figura 5.1, se puede observar el modelo de dominio del sistema, donde se aprecian las clases conceptuales significativas extraídas de los requisitos y las historias de usuario. De esta manera, se describen las entidades que las conforman, así como sus atributos y las relaciones con otras entidades.

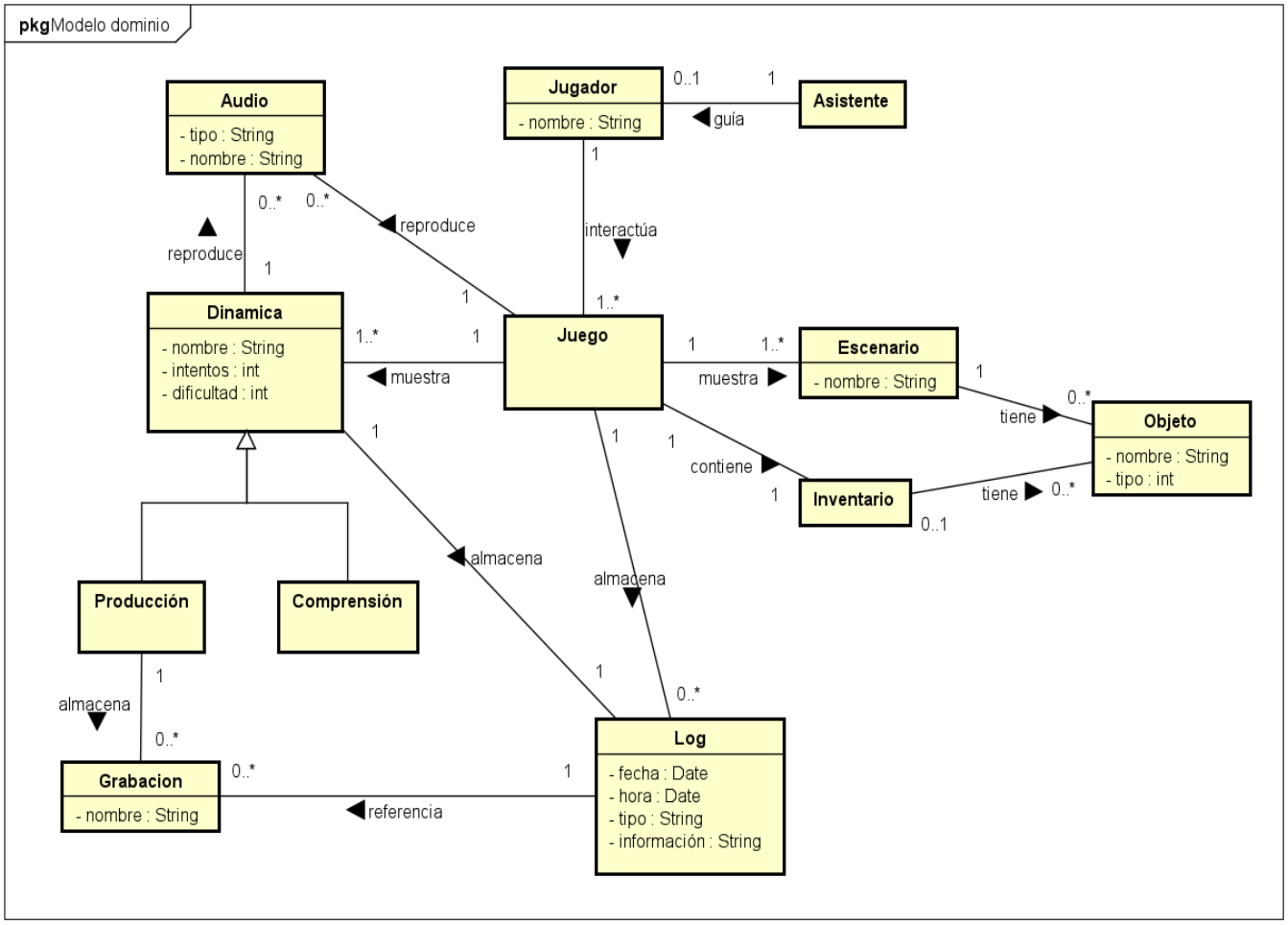

Figura 5.1: Modelo de dominio

Según se observa en el modelo conceptual realizado, se han extraído las entidades y relaciones a partir de los requisitos vistos con anterioridad. De esta manera, podemos resaltar la importancia del jugador, el cual interactúa directamente con el juego mientras es guiado en todo momento por un asistente. El juego muestra varios escenarios, dependiendo de los niveles que haya, además de un inventario. Ambos elementos pueden contener objetos pertenecientes e interactuables por ambas partes. Por otro lado, el juego muestra una serie de dinámicas, las cuales pueden reproducir audios al igual que el hace el propio juego. Dentro de las dinámicas, podemos ver la herencia de los dos tipos existentes, es decir, entre producción y comprensión. En cuanto a la dinámica de producción, se aprecia la opción de grabación, la cual es almacena en un fichero log, al igual que los eventos que suceden en todas las dinámicas, lo cual es esencial en el propio juego para almacenar toda la información de importancia.

5.1.3. Diagramas de actividad

En esta sección se exponen todos los diagramas de actividad de las historias de usuario definidas anteriormente.

Conseguir objetos (HU01)

El usuario puede seleccionar un objeto para poder añadirlo a su inventario. Se comprueba que el objeto se pueda almacenar, si es así, se registra el evento y se guarda el objeto en el inventario, mostrándolo visualmente.

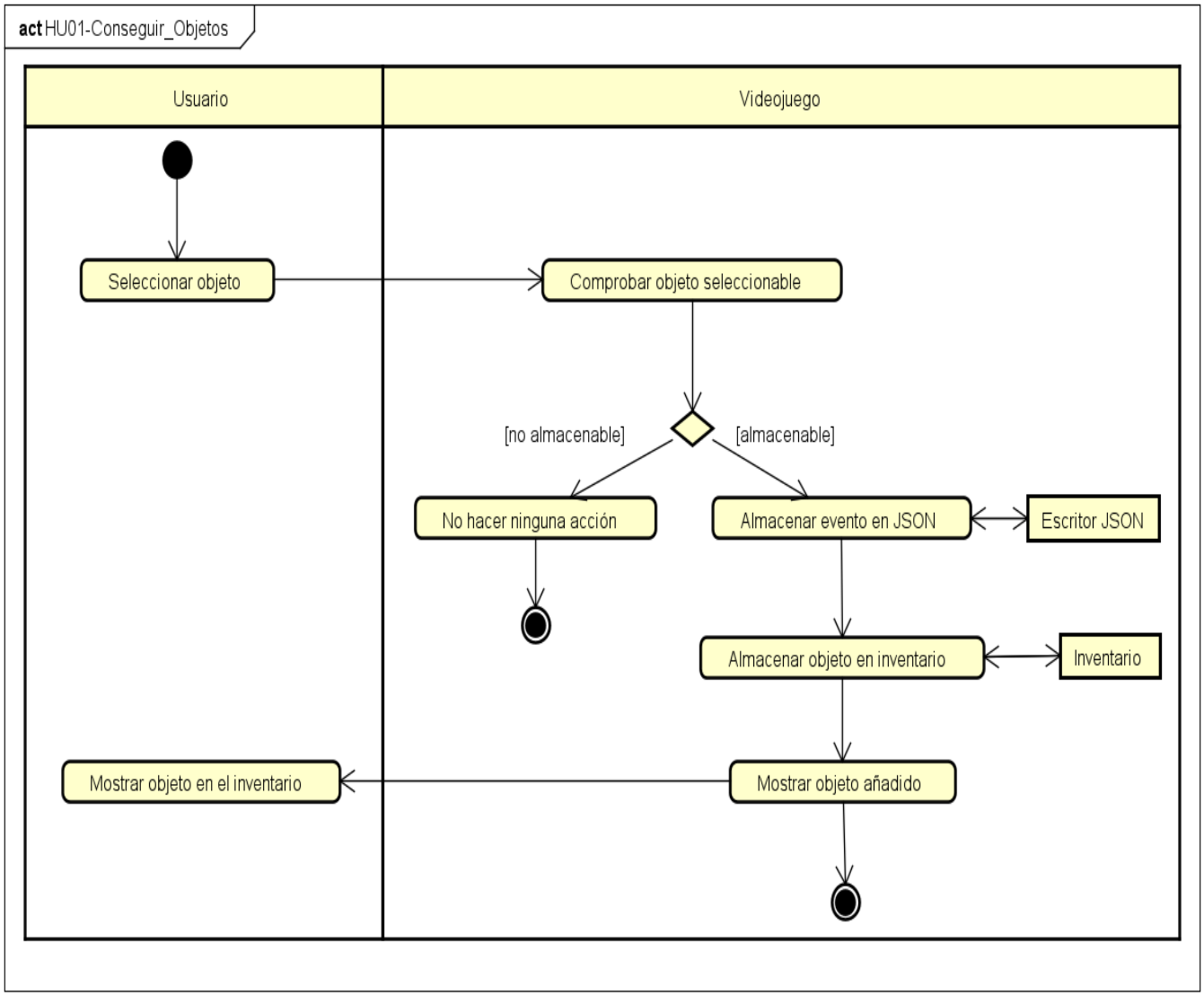

Figura 5.2: Diagrama de la historia de usuario de conseguir objetos (HU01)

Utilizar objetos del inventario (HU02)

El usuario puede seleccionar un objeto del inventario. Se comprueba que se pueda utilizar el objeto, es decir, que no estemos en medio de un audio, dinámica o animación y posteriormente se realiza la acción del mismo si la tiene. Mostramos los cambios visualmente.

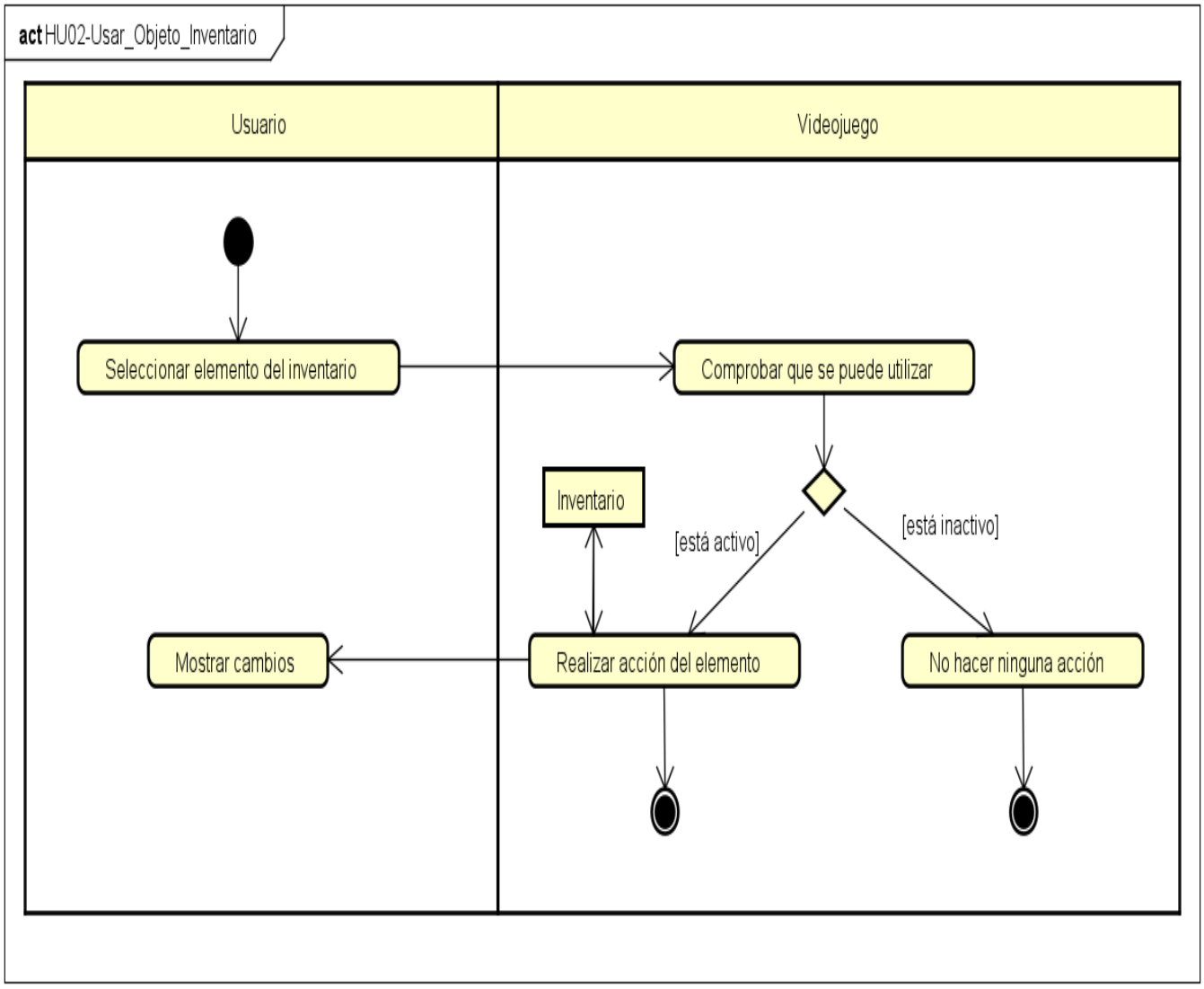

Figura 5.3: Diagrama de la historia de usuario de utilizar objetos del inventario (HU02)

Interactuar con los objetos del escenario (HU03)

El usuario puede seleccionar un objeto del escenario. Se comprueba que se puede utilizar, es decir, que no estamos en medio de un audio, dinámica o animación. Se comprueba que el objeto desencadena alguna acción y procede a realizarse si es así. Se muestran los cambios en la interfaz.

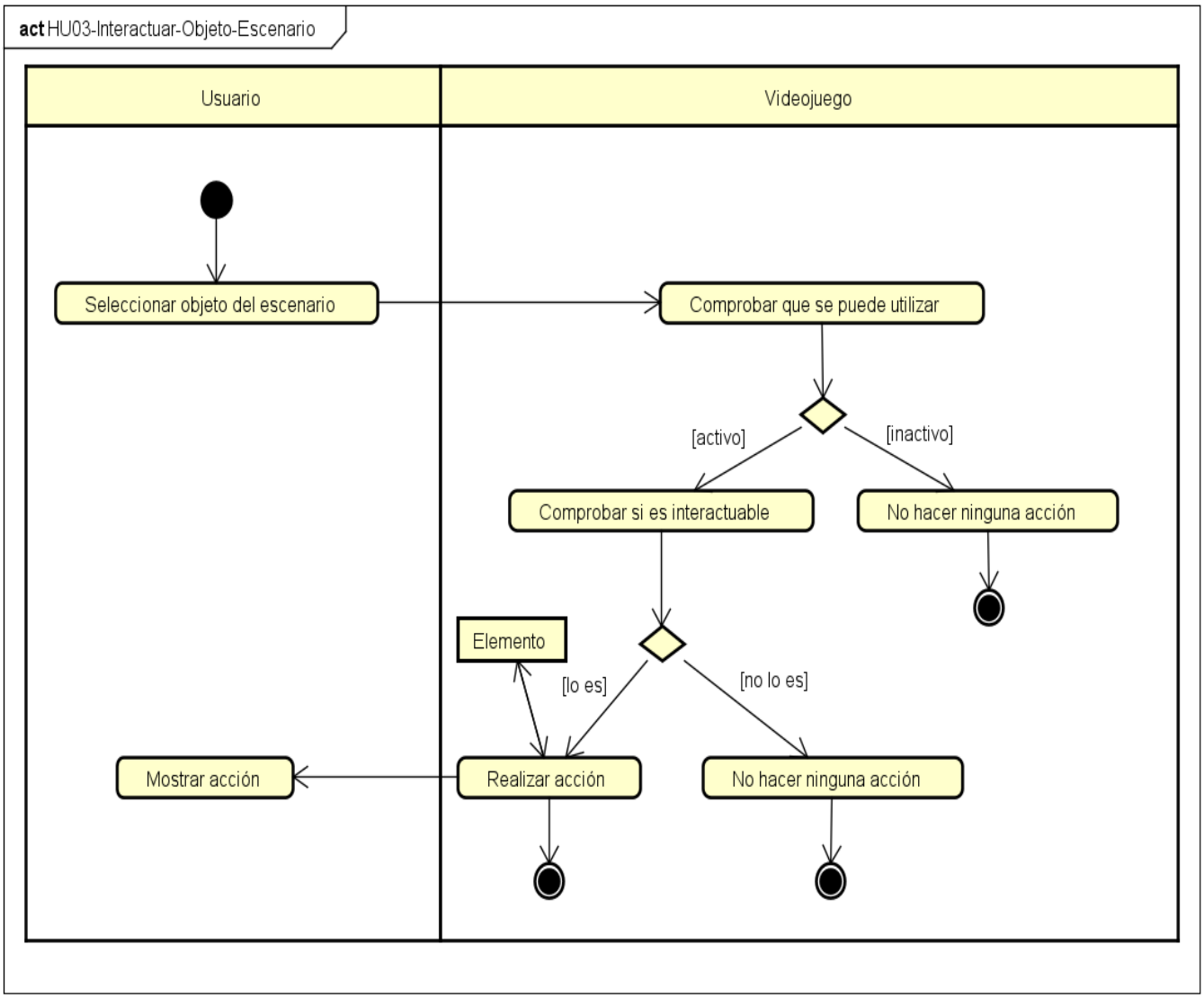

Figura 5.4: Diagrama de la historia de usuario de interactuar con los objetos del escenario (HU03)

Salir del juego (HU04)

El usuario decide salir del juego pulsando el botón. Se comprueba que no estemos en medio de una dinámica, audio o animación, si es así se procede a salir del mismo.

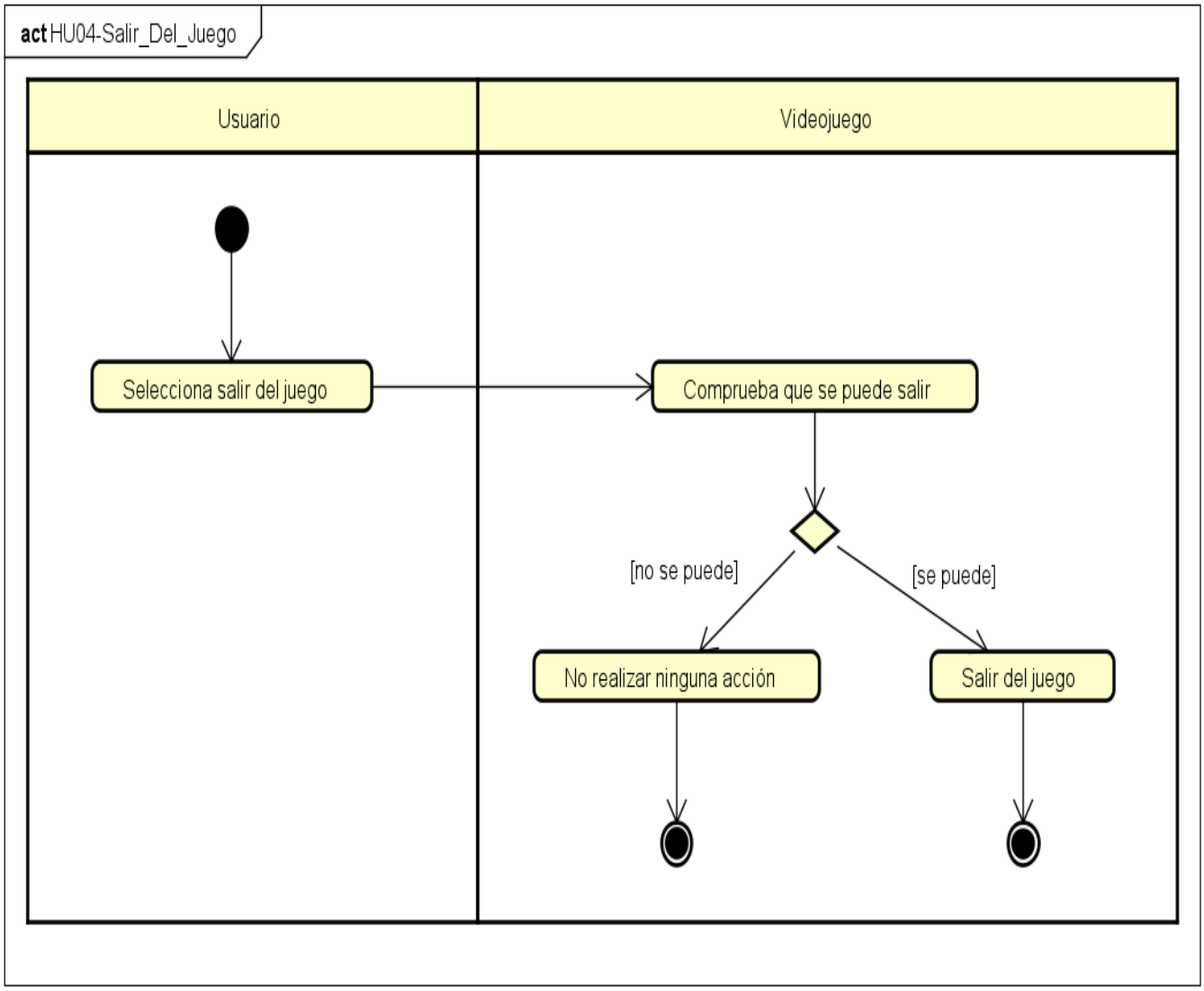

Figura 5.5: Diagrama de la historia de usuario de salir del juego (HU04)

Cambiar de escenario (HU05)

El usuario selecciona un objeto para cambiar de escena y se comprueba si ese elemento es el indicado. Si es así, se detiene el resaltado que poseía el objeto y se inicia la animación del personaje y del escenario para la transición. Finalmente, cuando terminen dichas animaciones se cambia de escena y se muestra.

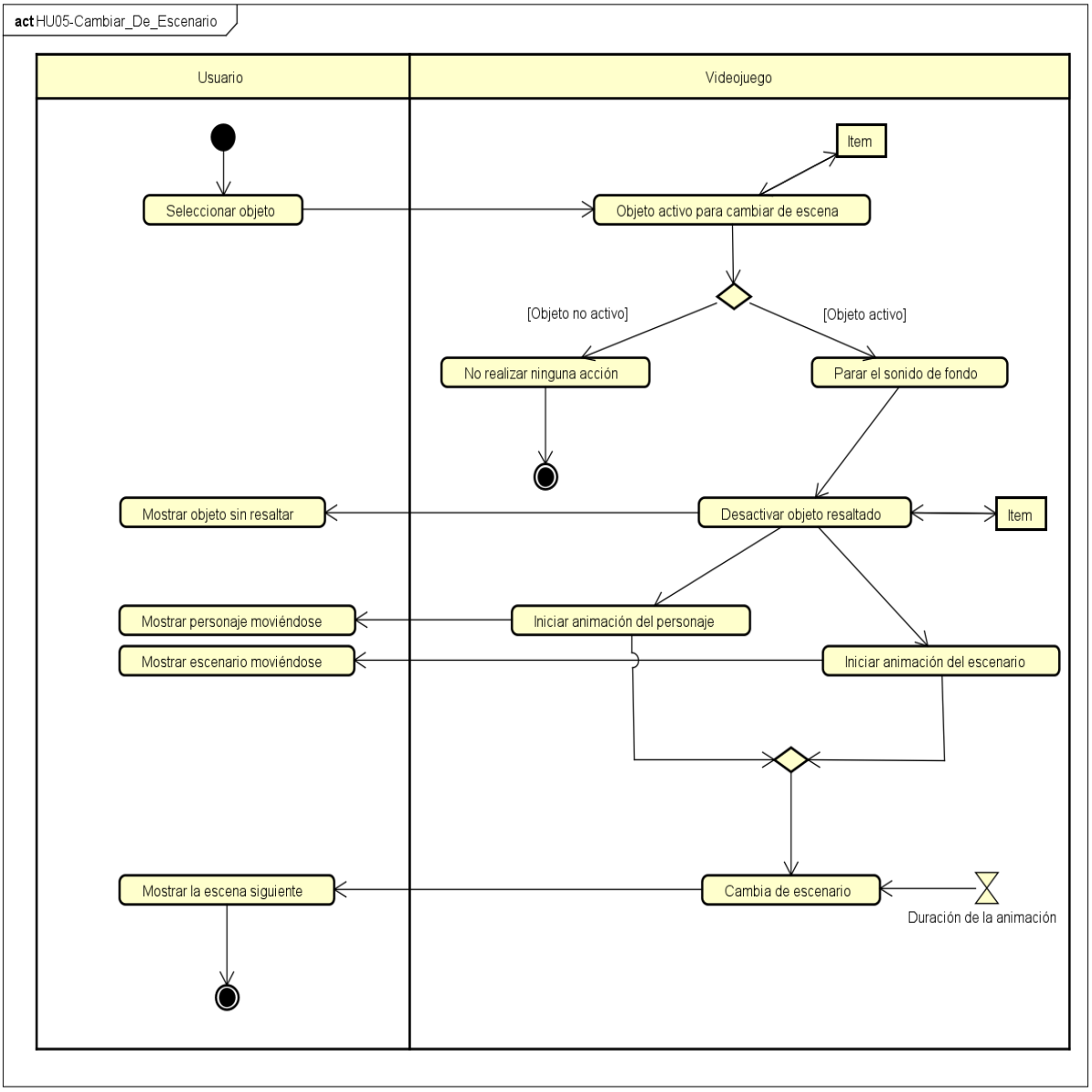

Figura 5.6: Diagrama de la historia de usuario del cambio de escenario (HU05)

Dinámica de producción (HU06)

Se inicia la dinámica de producción. Si se ha de mostrar la frase se enseña en el panel, sino se reproduce directamente la frase. Si el usuario lo desea, puede escuchar de nueva la frase, ejecutándose así el diagrama de escuchar la frase, contenido en la figura 5.8. Cuando el usuario está listo procede a grabar la frase, se ejecuta el diagrama de grabar frase, contenido en la figura 5.9.

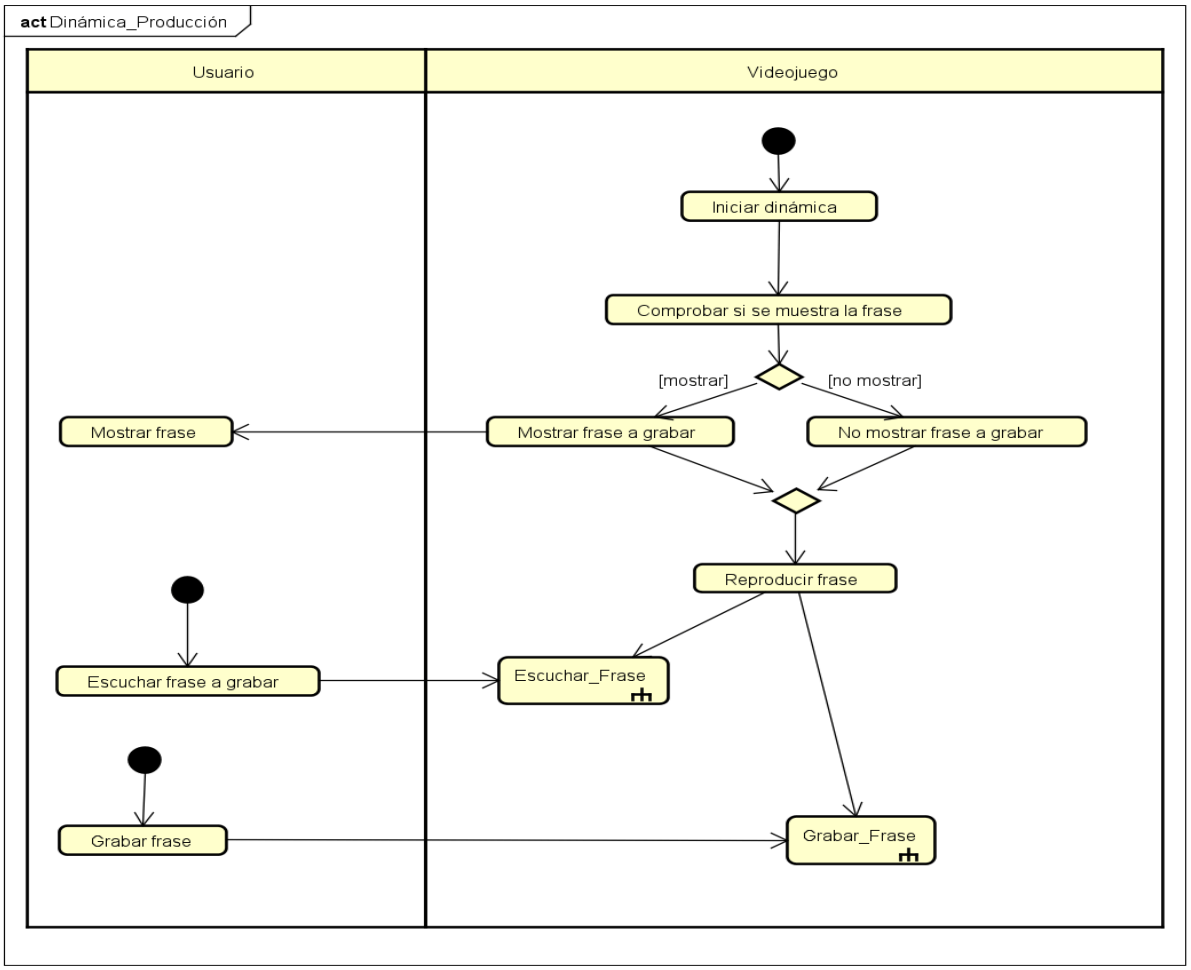

Figura 5.7: Diagrama de la historia de usuario de la dinámica de producción (HU06)

Escuchar frase (Diagrama de apoyo)

El usuario había seleccionado el altavoz y se comprueba si está activo para poder reproducir la frase. Si es así, se guarda el evento y se reproduce la frase correspondiente. Durante el audio, se desactivan los elementos de la dinámica y una vez termina el audio, se vuelven a activar todos.

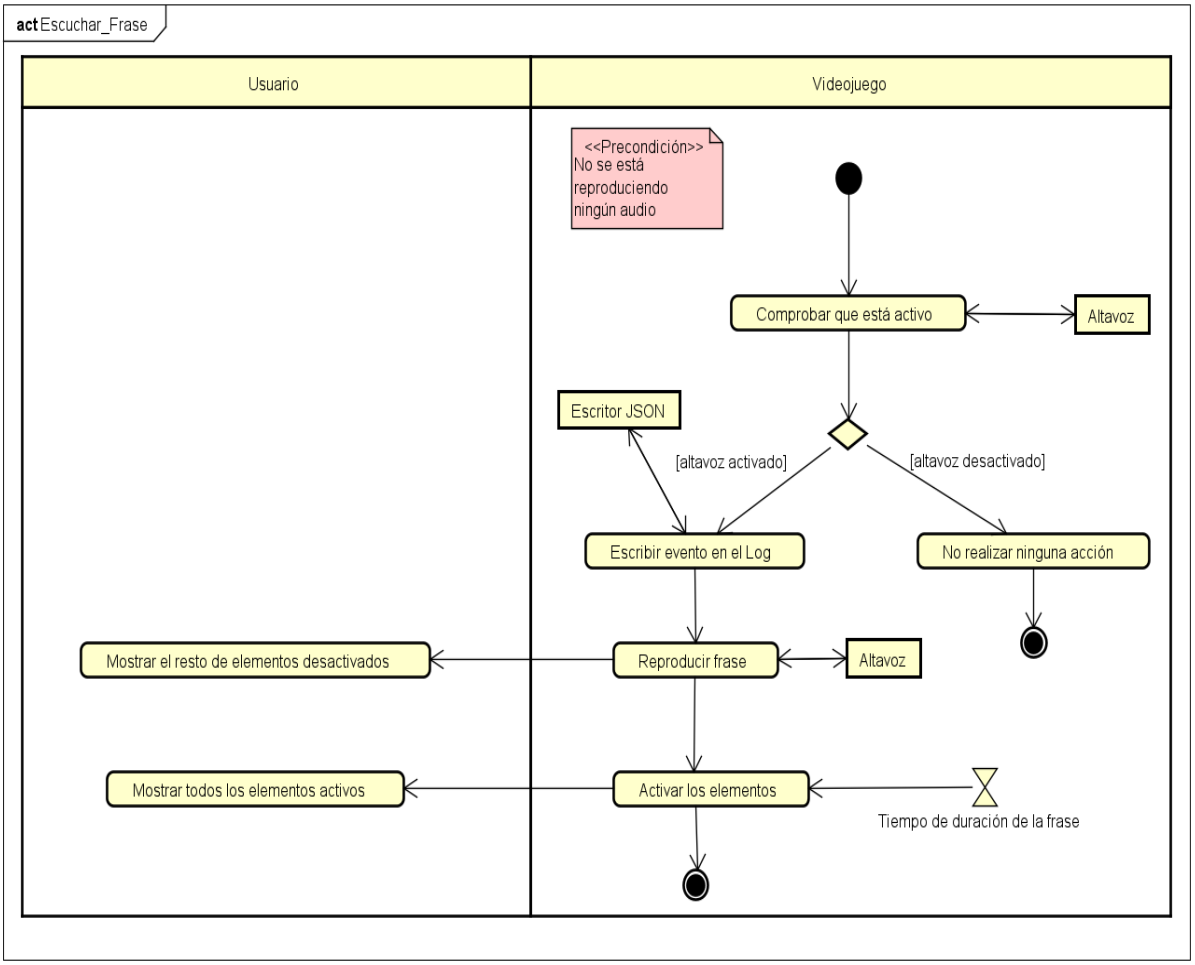

Figura 5.8: Diagrama de apoyo de la historia de usuario de producción (HU06)- Escuchar frase

Grabar frase (Diagrama de apoyo)

El usuario había seleccionado el micrófono para grabar la frase y se comprueba si está activo para poder empezar con la grabación. Una vez iniciada la grabación, el logopeda en cualquier momento puede considerar oportuno evaluar la frase, por lo que se ejecutaría el diagrama de evaluación de la grabación contenido en la figura 5.10.

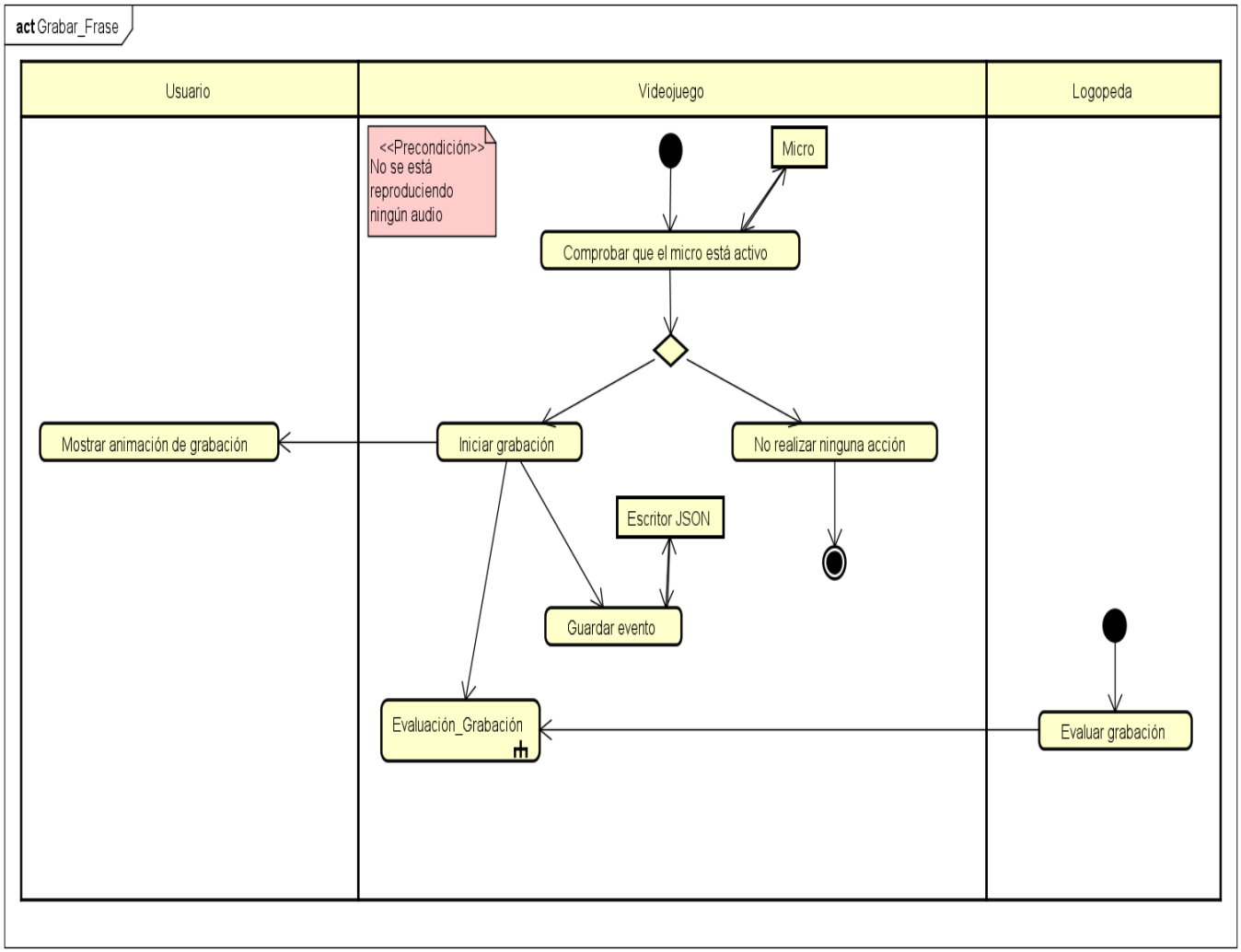

Figura 5.9: Diagrama de apoyo de la historia de usuario de producción (HU06)- Grabar frase

Evaluar grabación (Diagrama de apoyo)

El logopeda había decidido evaluar la frase, por lo que se comprueba la entrada por teclado introducida, si e correcta, se finaliza la grabación y se comprueba la evaluación. Si la evaluación es positiva, se comprueba si hay refuerzo, en caso afirmativo se acompaña al audio positivo con dicho refuerzo. Si la evaluación es negativa, se reproduce el audio de error y se comprueban los intentos restantes. Si quedan intentos se sigue intentando la dinámica, si no, se comprueba si hay refuerzo negativo y en caso de que lo hubiera se muestra.

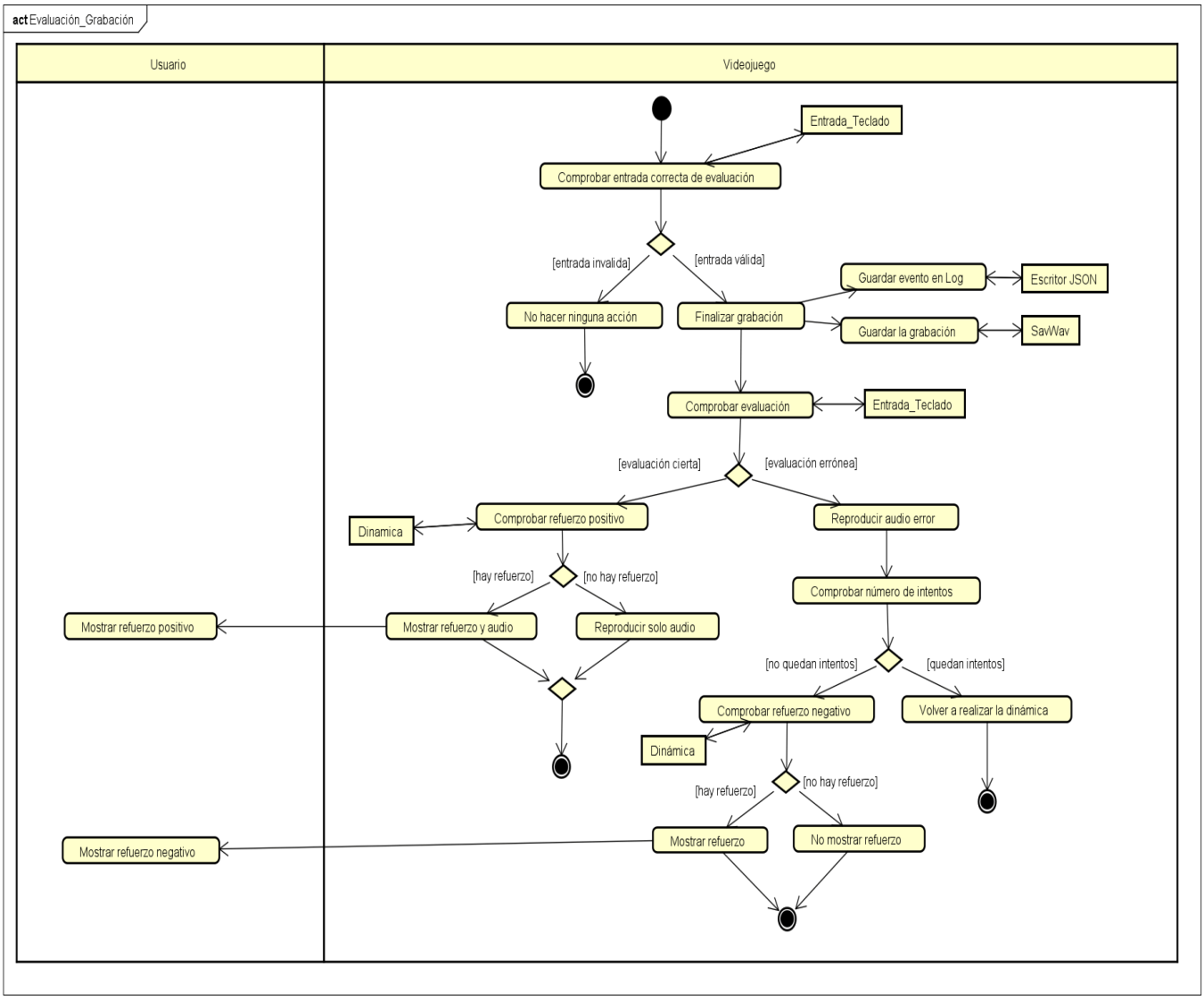

Figura 5.10: Diagrama de apoyo de la historia de usuario de producción (HU06)- Evaluar grabación

Dinámica de comprensión (HU07)

Se inicia la dinámica de comprensión. Primero se muestran los paneles y se reproduce el audio de instrucciones, para posteriormente reproducirse el audio con las opciones. Si el usuario lo desea, puede escuchar de nueva la frase, ejecutándose así el diagrama de escuchar la frase, contenido en la figura 5.8. Cuando el usuario está listo selecciona una de las opciones, ejecutándose así el diagrama de selección de frase, contenido en la figura 5.12.

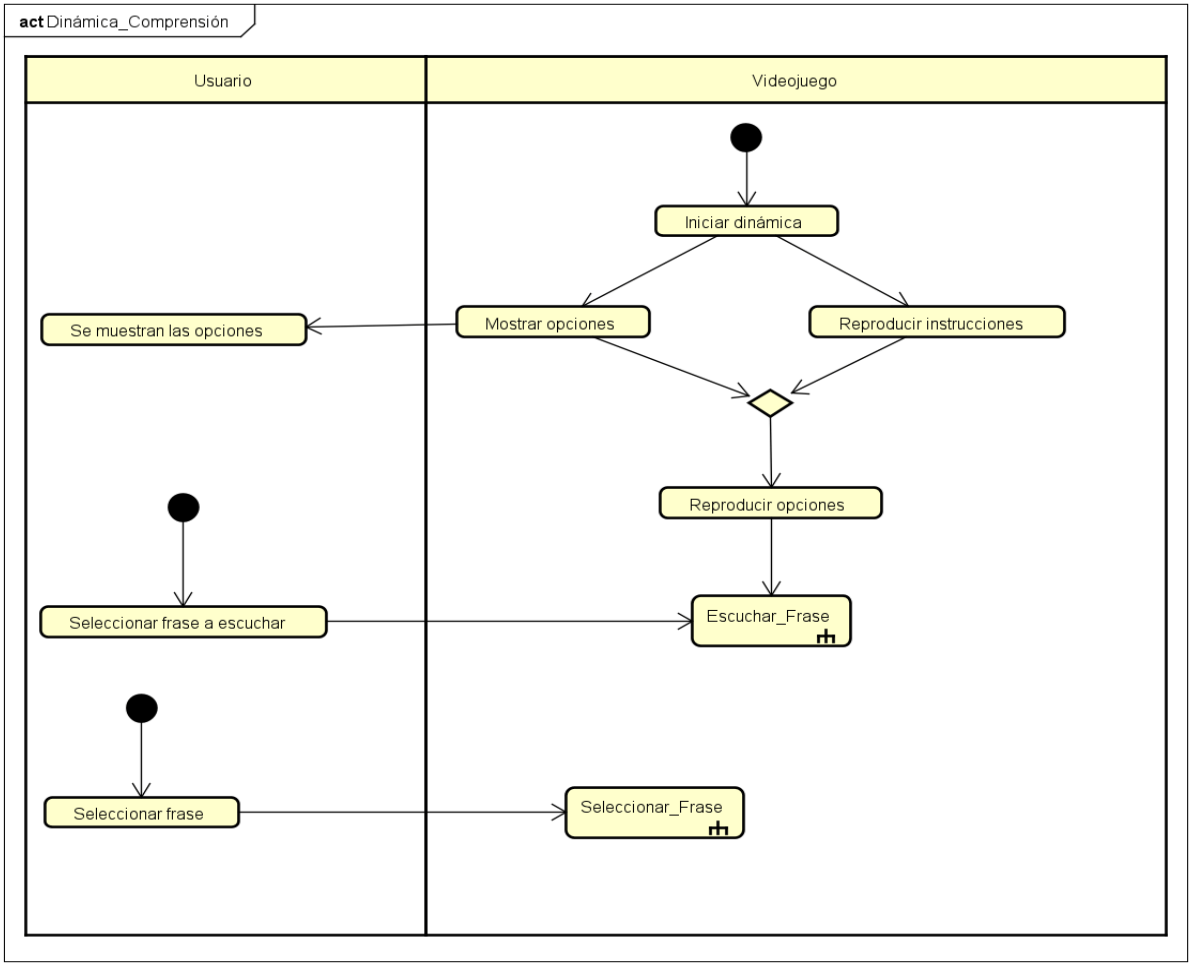

Figura 5.11: Diagrama de la historia de usuario de la dinámica de comprensión (HU07)

Seleccionar frase (Diagrama de apoyo)

Cuando el usuario ha seleccionado una frase, se comprueba si la respuesta es correcta, si es el caso, se guarda el evento y se comprueba si hay refuerzo. En caso de que lo hubiera, se acompaña el audio positivo con dicho refuerzo. Si la respuesta es incorrecta, se guarda el evento y se comprueban los intentos. Si quedan intentos se reproduce el audio de fallo y se reintenta la dinámica. Si no quedan intentos, se comprueba si se bloquean las opciones incorrectas, si es el caso, se reintenta la dinámica solo con la opción correcta activa. Si no es el caso, se comprueba si existe refuerzo negativo y se acompaña el audio de fallo con él si existe.

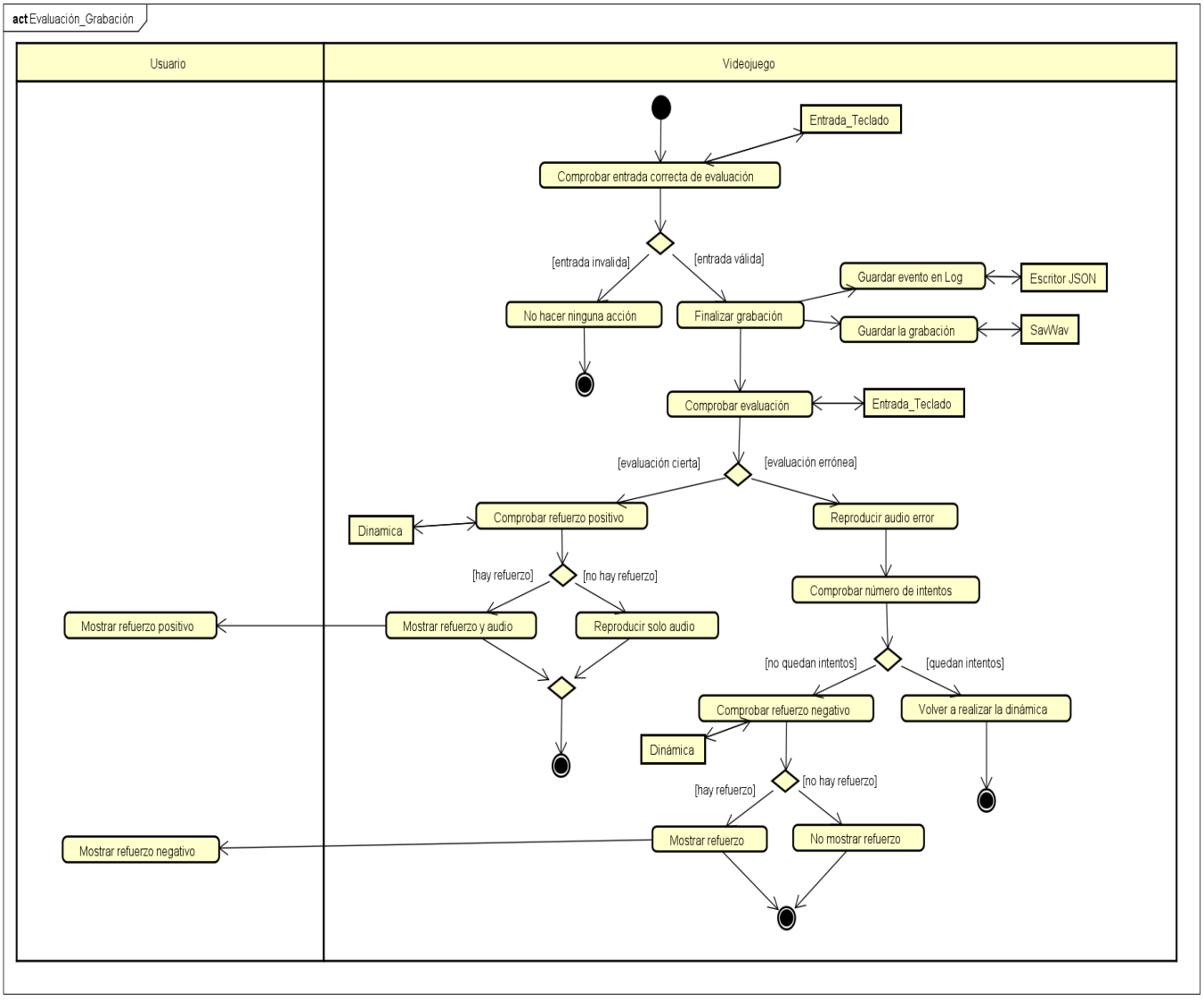

Figura 5.12: Diagrama de apoyo de la historia de usuario de comprensión (HU07)- Seleccionar frase

5.2. Estructura del proyecto

En la figura 5.13, podemos ver la estructura de la aplicación desarrollada. En ella se observa la estructura de ficheros real, la cual facilita ver las carpetas y recursos que forman el proyecto.

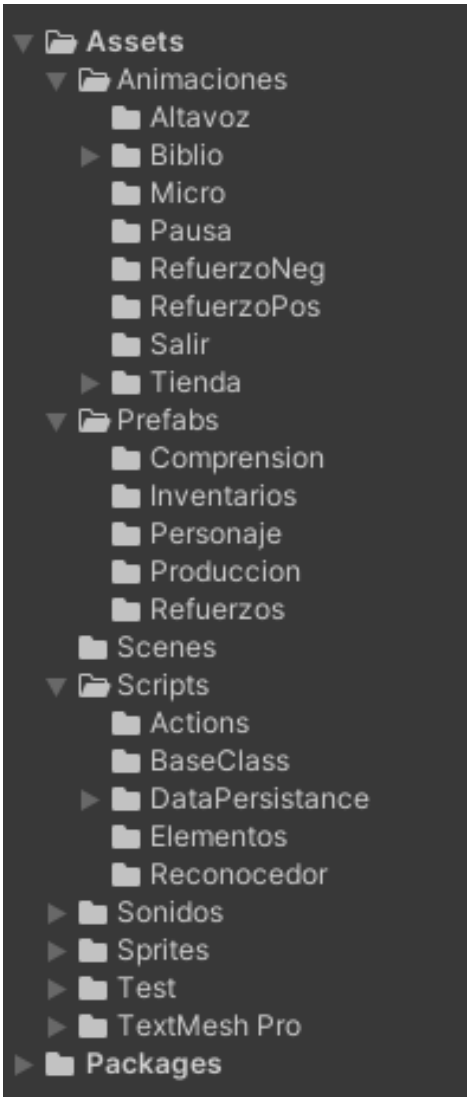

Figura 5.13: Estructura del proyecto

En primer lugar, se aprecia la carpeta de animaciones, la cual se subdivide en varias carpetas a su vez. En este apartado se almacenan los elementos necesarios para que toda animación funcione, es decir, los animation clips, que forman las distintas animaciones referentes a un elemento, y el animator controller, el cual se encarga de gestionar los estados de las animaciones y transicionar entre ellos. Por este motivo, se reparten por similitud, ya que la escena de la biblioteca o la tienda engloba una serie de animaciones entorno a esas pantallas.

En cuanto a los prefabs, podemos ver como se almacenan todos los objetos prefabricados y que serán reutilizados en varias escenas. Por ello, se observa la necesidad de agruparlos por dinámicas de comprensión o producción, así como los refuerzos, ya que se repiten en distintas escenas. Lo mismo

pasa tanto con el personaje principal, como con el inventario, los cuales están presentes en todas las pantallas.

Por otra parte, la carpeta scenes almacena las distintas escenas que contiene el juego. En nuestro caso, está formada por las cuatro pantallas que engloba el mismo, es decir, el interior y exterior de la librería y biblioteca. Si se desea añadir alguna escena nueva, bastaría con incluirla en este apartado para mantener la estructura y el orden del proyecto.

En cuanto a los scripts, engloba una gran parte del desarrollo, ya que se encuentran los elementos que dan sentido a la parte visual creada en Unity. De este modo, podemos ver la jerarquía interna de carpetas. Para orientarnos mejor, se subdividen en:

- Actions: Contiene todos los elementos que heredan de la clase *Actions* y por tanto poseen el método *act()*, necesario para poder ejecutarse secuencialmente en la lista de operaciones de cada nivel.
- BaseClass: Contiene la clase *Actions*, la cual define el método *act()* para que el resto puedan heredar de ella.
- DataPersistance: Contiene la parte relacionada con el guardado de datos. No hay una base de datos, pero si existen ficheros log donde almacenar la información. Por tanto, aquí podemos encontrar la clase que instancia la información del juego que se va a guardar, así como el escritor que modifica dicha información para cada elemento y la almacena en un fichero.
- Elementos: Contiene los elementos que no pertenecen a las acciones pero si implementan funcionalidad a los elementos del juego, como pueden ser botones o elementos seleccionables.
- Reconocedor: Guarda la información relacionada con el reconocedor sobre el que se realizan las peticiones.

Por otra parte, encontramos otra carpeta importante en nuestro desarrollo, la cual es la de sonidos. Aquí se almacenan todos los elementos de audios que se van a utilizar en el juego, tanto los sonidos de fondo como los que se reproducen en las dinámicas.

Para el apartado de los sprites, se engloban todos los sprites utilizados para poder crear las animaciones. En algunos casos, se han proporcionado de forma individual y en otros en forma de secuencia, por lo que ha sido necesario separarlos mediante el editor de sprites para conseguir sprites individuales. De esta forma, se engloban tanto fondos, como personajes, como objetos que posteriormente pueden ser o no animados.

Por último, remarcamos la carpeta de *test*, donde se encuentran las pruebas realizadas para los test unitarios. Como se explicará en el capítulo 6, se separan en dos carpetas, las cuales son *edit mode* y *play mode*, dependiendo de las pruebas que queramos realizar.

5.3. Diagramas de diseño

A lo largo de esta sección se presentarán los patrones y técnicas de diseño utilizadas durante el proceso de migración de la aplicación. Además, se expondrán los paquetes involucrados, así como las clases de cada paquete que los forman, y finalmente un diagrama de despliegue de la aplicación.

5.3.1. Arquitectura

Para la definición de la arquitectura software, se ha decidido aplicar el patrón Programación por Capas [53], ya que permite separar la interfaz, la lógica del juego y los datos en capas independientes 5.14. La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que ocurra algún cambio, solo se modifica el nivel requerido sin tener que modificar ninguna clase de las otras capas.

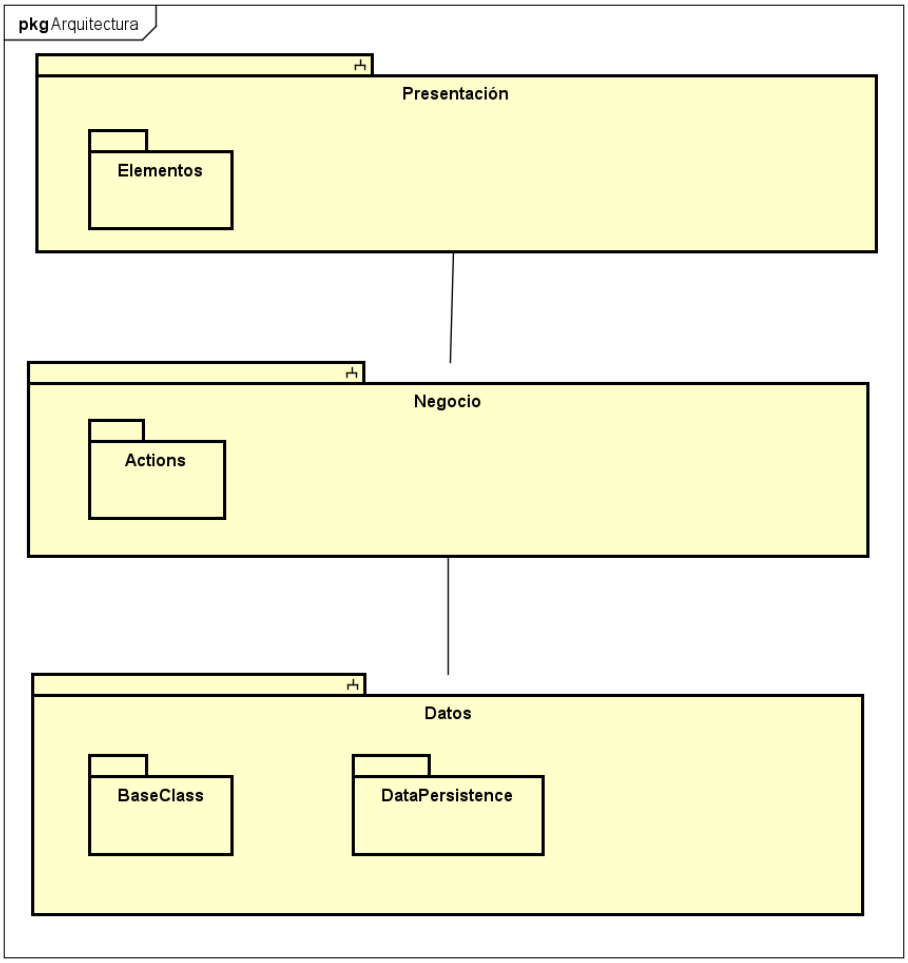

Figura 5.14: Capas de la aplicación

Por tanto, el código se organiza en 3 capas, repartiendo las funciones la siguiente manera:

- Presentación: Destinada a la interfaz del usuario y lo relacionado con ella.
- Negocio: En dicha capa reside el modelo de negocio.
- Datos: No existe una persistencia con la base de datos, pero si una capa referente al almacenamiento de los mismos.

5.3.2. Patrones de diseño

Los patrones de diseño representan soluciones generales a problemas repetitivos de software [54]. Son una serie de consejos y principios que ayudan a organizar el código para que sea mantenible, ofreciendo a su vez una mayor calidad final. Se exponen a continuación los empleados.

5.3.2.1. Patrón experto

Este patrón [55] indica que sólo se debe asignar la responsabilidad a aquellas clases que contengan la información necesaria para manejarla. De esta manera, permite:

- Encapsular la información.
- Mantener un bajo acoplamiento: Los módulos interactúan entre sí por medio de pequeñas comunicaciones, sin necesidad de conocer la implementación del otro módulo. Caracteriza a los sistemas robustos y fácilmente mantenibles.
- Alta cohesión: La medida en la que una clase realiza solamente la tarea para la cual fue diseñada, sin involucrarse en otras acciones. De esta manera, es reutilizable.

En nuestro proyecto, se ha utilizado por ejemplo, en el escritor en ficheros log, el cual contiene la información necesaria para almacenar los registros.

5.3.2.2. Patrón Singleton

Este patrón [56] permite mantener una única instancia de un objeto en memoria. Para llevarlo a cabo, el patrón almacena su propio estado como atributo y mantiene privado su constructor. De esta manera, si otro objeto B desea hacer uso del objeto Singleton, dicho objeto Singleton creará la instancia o devolverá la creada anteriormente (ver Figura 5.15). En nuestra aplicación, este patrón ha sido utilizado para la parte de escritura en ficheros log. De este modo, se ha usado una sola instancia del escritor en dichos ficheros para poder almacenar los datos pertinentes.

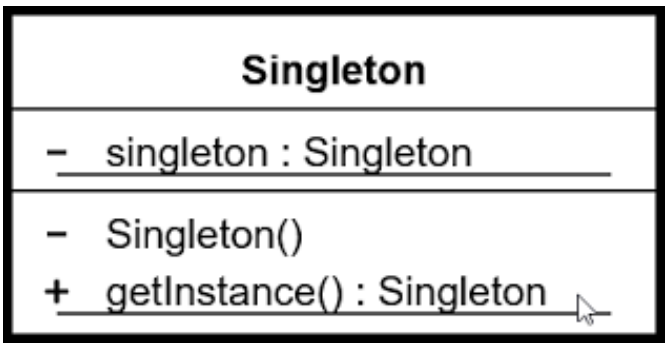

Figura 5.15: Estructura del Patrón Singleton

5.3.2.3. Patrón Observador

Este patrón de diseño [57] permite que un objeto A se suscriba a otro B, de manera que cuando el objeto B sufre algún cambio en su estado, este notifica a los objetos que se han suscrito a él que ha habido cambios. De esta manera, el objeto A puede reaccionar a estos cambios (ver Figura 5.16). También es útil dada la asincronía del lenguaje, ya que en ciertas ocasiones se necesita esperar por una respuesta de una función para poder utilizar sus datos. En este proyecto, se ha implementado al usar las corrutinas, ya que también utilizan este patrón al notificar cuando finalizan. Además, se ha utilizado para realizar las llamadas a la API del reconocedor, el cual contiene las capacidades necesarias para validar el audio.

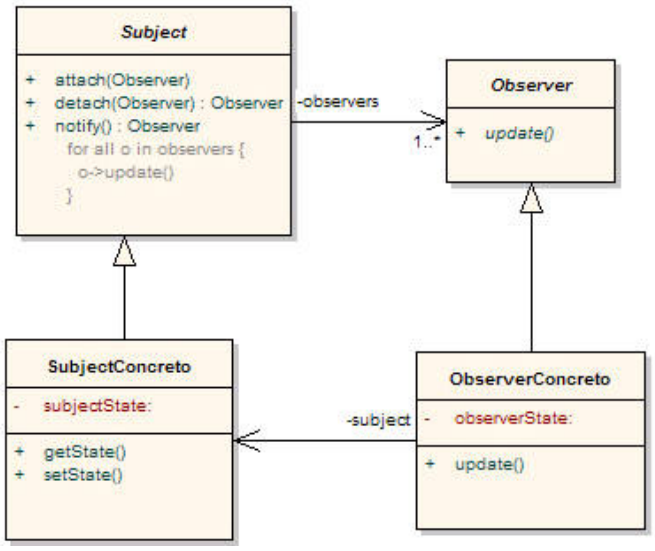

Figura 5.16: Estructura del Patrón Observador [58]

5.3.2.4. GRASP: Polimorfismo

Los patrones GRASP en la Programación Orientada a Objetos, son los patrones generales para la asignación de responsabilidades. Entre ellos se encuentran los ya comentados Alta cohesión y Bajo acoplamiento. Pese a que más que patrones son buenas prácticas, considero valioso incluirlo en este

apartado.

Entre ellos se encuentra el Polimorfismo [59], lo que permite enviar mensajes a las diferentes instancias de una misma clase. Los PreFabs de Unity [47], cumplen la función habitual del polimorfismo, esto se ve reflejado, por ejemplo, en el inventario común para todas las escenas o en los paneles de las dinámicas, los cuales comparten la estructura en las distintas escenas.

5.3.2.5. Patrón Fachada

El Patrón Fachada [60], como se ve en la figura 5.17, permite acceder a un paquete con varias clases a través de una única clase y así centralizar todas las llamadas entre paquetes en un único lugar. Para nuestro proyecto, hemos observado su uso, por ejemplo, a lo hora de almacenar las grabaciones. Para ello, los clientes realizaban una serie de llamadas a la fachada, la cual delegaba las peticiones en las clases del subsistema correspondientes de almacenar los audios.

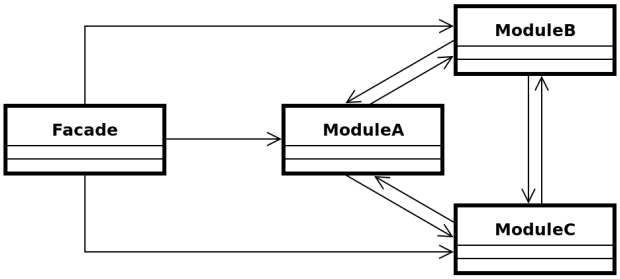

Figura 5.17: Estructura del Patrón Fachada

5.3.2.6. Patrón Prototype

Este patrón [61], permite crear tantos duplicados de un objeto como se desee. Dicho patrón ya existe en Unity en la forma del método instanciar. De esta manera, cuando añadimos los prefabs al script y queremos instanciarlos nuevamente, se llama al método Instantiate de Unity para obtener un duplicado del objeto original.

5.3.2.7. Patrón State

Este patrón [62], también se conoce como máquina de estado y es muy útil si el juego está dividido en diferentes estados, como lo están los personajes o elementos que lo forman. De esta forma, al tener una cantidad finita de estados, se obtiene una máquina de estados finitos. Dicha máquina se utiliza y transiciona para cambiar de comportamiento en vez de utilizar múltiples sentencias condicionales. En

este proyecto, se ha utilizado a la hora de realizar las distintas animaciones. A parte de la propia animación, se crea una máquina de estados para poder transicionar entre los que hay, utilizando un trigger o una variable para disparar el estado que queramos en un momento determinado.

5.3.2.8. Patrón Update

Este patrón [63], ya se ha implementado en Unity mediante el método de Update. Dicho método comprueba constantemente las sentencias que encierra procesándolas en cada instante. En nuestro desarrollo, ha sido útil para comprobar información de manera constante, como la entrada por teclado para evaluar las dinámicas, en vez de estar continuamente realizando peticiones en distintos puntos del código.

5.3.3. Diagrama de paquetes

En este apartado, se mostrará en detalle el diagrama de paquetes que compone la aplicación final.

En primer lugar, en la Figura 5.18, se muestra la estructura típica de código en Unity, la cual no afecta a la arquitectura y es la que hemos empleado en este proyecto. Se puede ver como se hace énfasis en el paquete de Assets, ya que en él se encuentra la aplicación como tal.

A continuación, podemos observar con más detalle las figuras que muestran en detalle los paquetes más importantes mencionados en 5.18. Estas figuras son las siguientes (Figura 5.19, Figura 5.20, Figura 5.21, Figura 5.22).

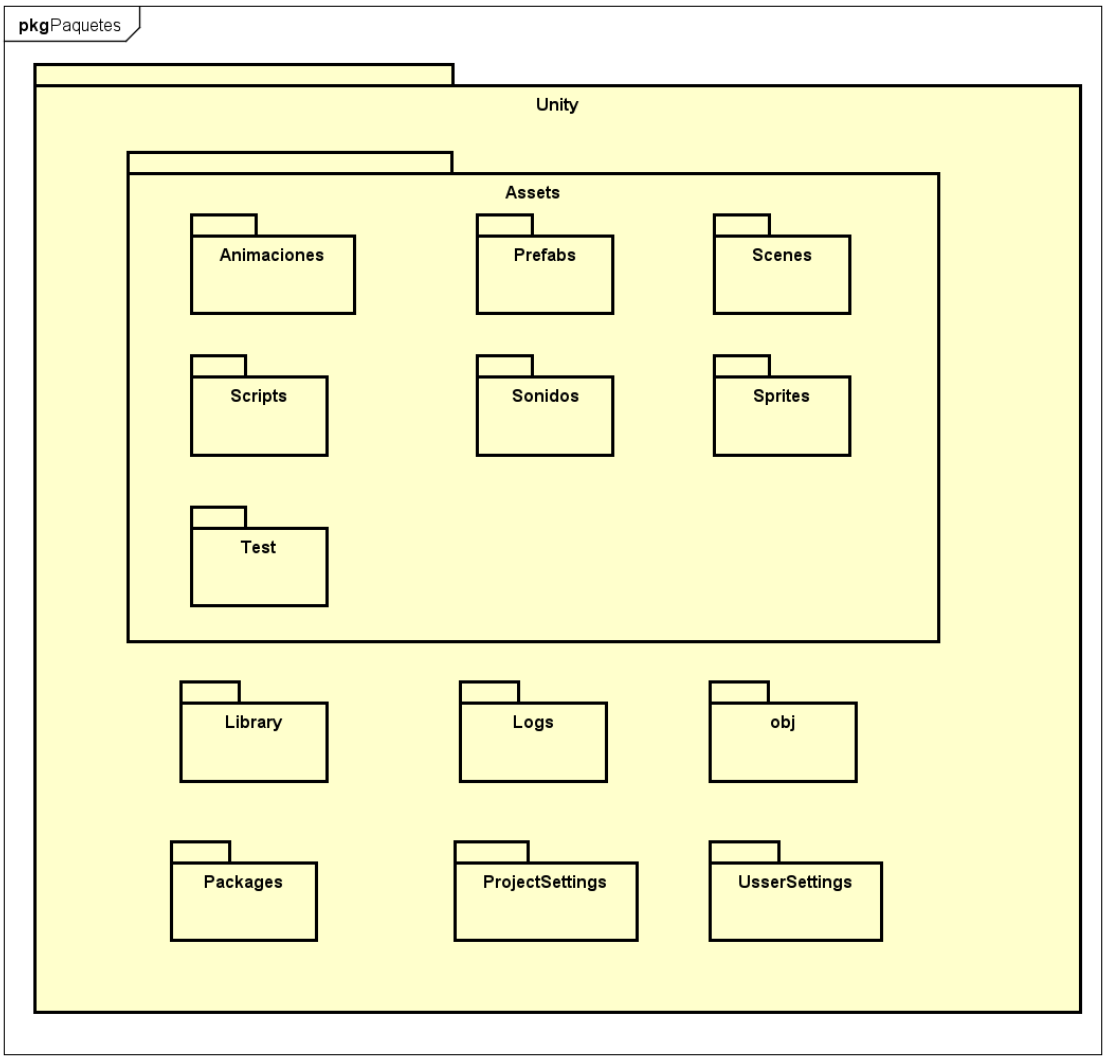

Figura 5.18: Estructura de Unity por paquetes

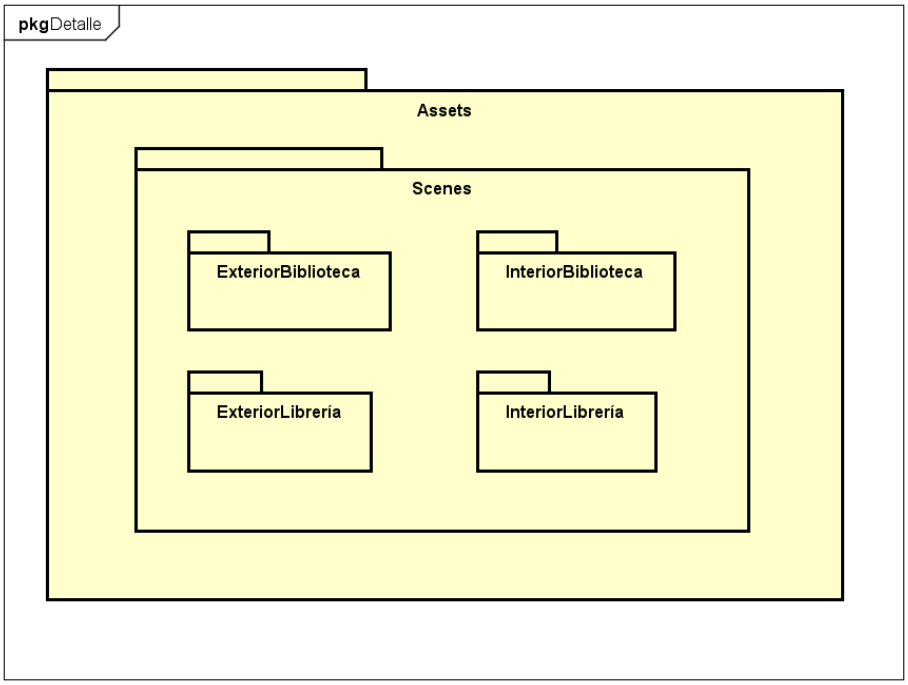

Figura 5.19: Diseño detallado del paquete Scenes

Se observa el diseño en profundidad del paquete de Scenes, donde se aprecian las 4 escenas que componen el juego.

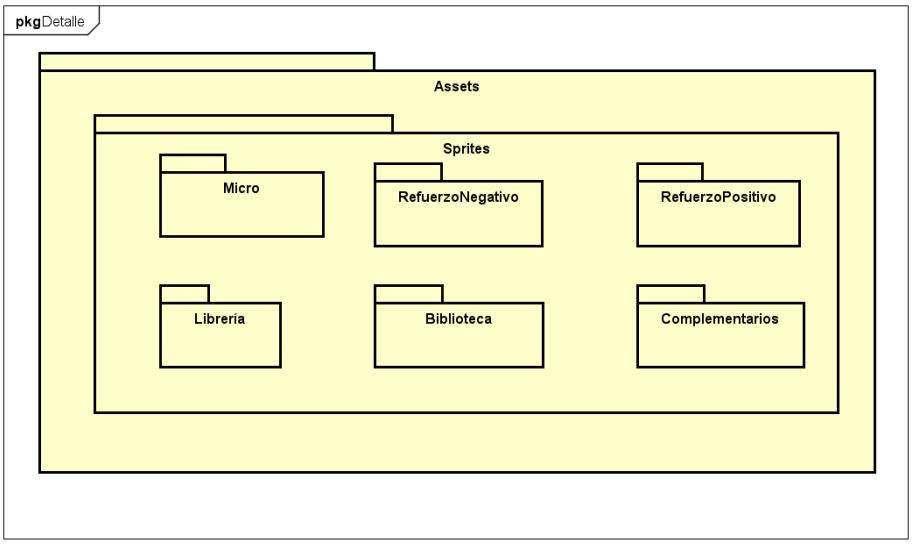

Figura 5.20: Diseño detallado del paquete Sprites

Se puede apreciar en detalle la organización de los Sprites, como vemos separados principalmente por las escenas existentes, los refuerzos y los elementos complementarios.

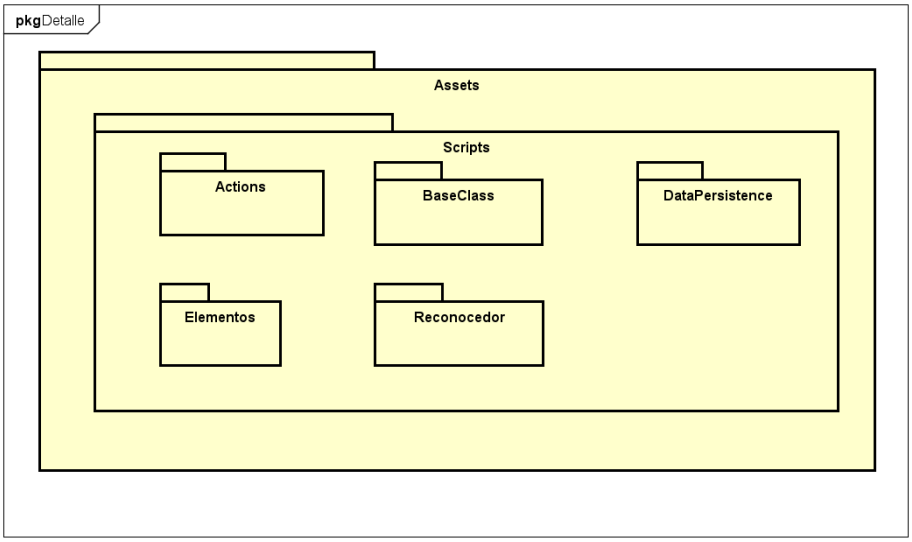

Figura 5.21: Diseño detallado del paquete Scripts

Se observan los paquetes de Scripts detalladamente, separados como se ha comentado con anterioridad.

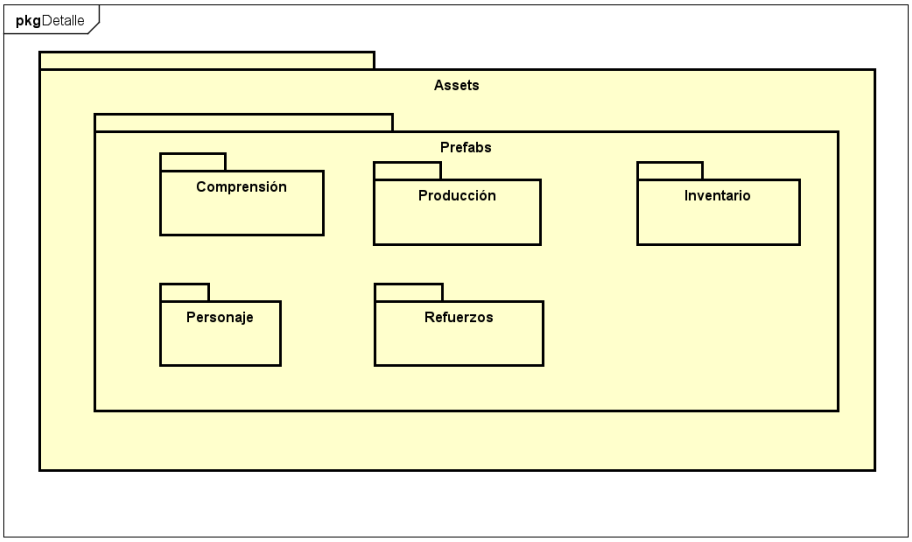

Figura 5.22: Diseño detallado del paquete Prefabs

En los Prefabs, podemos ver como se separan principalmente por dinámicas y refuerzos, así como la parte del inventario y el personaje, también reutilizables.

5.3.4. Diagrama de despliegue

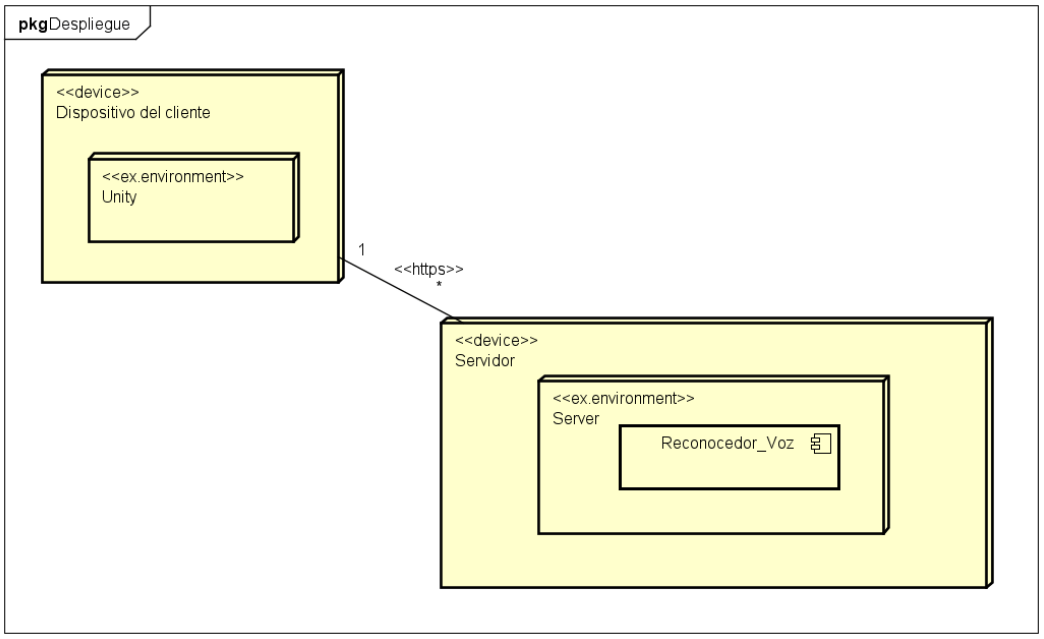

Figura 5.23: Diagrama de despliegue

El diagrama de despliegue explica de forma visual cómo se han organizado las diferentes partes de la aplicación. En nuestro caso, se puede apreciar que existe un dispositivo donde se está realizando el despliegue, el cual puede ser un ordenador o un dispositivo móvil, dependiendo donde se haya exportado y se esté utilizando la aplicación. Se puede apreciar que se conecta al servidor que contiene el reconocedor de voz, para realizar las peticiones al mismo. De la misma forma, la conexión https con el servidor se realiza para verificar los tokens JWT que se envían en las peticiones, ya que los servidores necesitan de autenticación para ello.

5.3.5. Diseño de la interfaz de usuario

Para comenzar con este apartado cabe destacar que, para el desarrollo de este TFG, se han seguido las mismas líneas de diseño que en el TFM de Mario Corrales en el momento en el que se desarrolló Pradia por primera vez [1]. Por ese motivo, se asume la validez de la interfaz gráfica realizada debido a dicho diseño inicial. Por otra parte, si nos fijamos en este proyecto, podemos ver que tiene como finalidad su ejecución en cualquier ordenador, así como la intención de ser exportado a cualquier dispositivo móvil. De todas formas, el rasgo esencial que debe caracterizar al proyecto es la simplicidad de uso, debido al público al que va dirigido. Por ello, debe haber una interfaz sencilla que permita a todos los usuarios, independientemente de sus aptitudes informáticas, manejar sin problemas todas las historias de usuario a implementar. En esta sección, nos damos cuenta de la interacción clave entre el usuario y la máquina, ya que esto puede marcar el éxito o fracaso de una aplicación. Una interacción nefasta y poco intuitiva puede destruir por completo un juego bien realizado.

Según lo visto anteriormente, podemos afirmar que la interfaz es aquella parte de la aplicación que permite al usuario interactuar con la máquina, intercambiando información entre ambos. En la interfaz, también se tiene en cuenta el hardware del dispositivo. En una primera instancia, es el caso del ratón y teclado para los distintos ordenadores donde se ejecute, pero también puede ser el caso de una pantalla táctil si se exporta a dispositivos móviles. Por todo ello, el interés en realizar una buena interfaz es elevado, debido a su papel fundamental a lo largo de la vida de una aplicación. De este modo, se deben seguir unas guías de diseño que ayuden a enfocar el diseño correctamente, así como realizar pruebas con usuarios reales para detectar cualquier dificultad en su uso y poder modificarla. A continuación, se comentarán por encima los conceptos vistos, así como la interfaz de usuario. No se han realizado en profundidad estos apartados, ya que han sido realizados en el proyecto de Pradia como veremos.

Usabilidad

Según se ha comentado, es fundamental que la aplicación sea usable, es decir, la capacidad que tiene para alcanzar los objetivos propuestos en un contexto. Entre los criterios fundamentales destacan:

- Eficiencia: Conseguir el cumplimiento de los objetivos de una manera óptima con los recursos disponibles
- Eficacia: Capacidad para conseguir los objetivos marcados
- Satisfacción: El usuario debe utilizar la aplicación de forma cómoda

En nuestro caso, no vamos a entrar en detalle con este apartado, ya que se ha desarrollado de manera extensa en el proyecto de Pradia. Como se puede observar en dicho proyecto [1], en el capítulo 4, referente a los test de usabilidad realizados. Para ello, se ha comprobado la eficacia de las decisiones tomadas en relación con la interfaz de usuario, lo cual es muy importante para la correcta interacción de las personas con discapacidad intelectual. En dicho apartado se exponen las pruebas realizadas, así como las conclusiones, las cuales determinan el positivismo de las pruebas, así como las mejoras a implementar en futuros desarrollos.

Guías de diseño

Aquí podemos encontrar una serie de ideas que se pueden aplicar a un producto para facilitar su diseño, marcando los pasos que se pueden y no dar. No corresponden a resoluciones de todos los problemas, sino que son plantillas generalistas para afrontarlos. Al igual que con la usabilidad, este apartado ya ha sido desarrollado en profundidad en el desarrollo de Pradia. Por ello, como se puede observar en dicho proyecto [1], en el capítulo 2, en el apartado de guías de diseño, se pueden apreciar las mismas, así como una explicación detallada. Entre ellas, podemos observar que se profundiza entre:

- Navegación
- Gráficos, imágenes y multimedia
- Ayuda al usuario

- Motivación

Diseño de la interfaz de usuario

Para este apartado, se pueden observar los prototipos diseñados para implementar la interfaz, destacando algunas de las decisiones que han sido consideradas. Además, se detallan tanto las dinámicas que componen el juego, como las decisiones de diseño planteadas con el objetivo de reducir al máximo los problemas que ocurren en la interacción persona-computador que se han observado en las personas con síndrome de Down. Para ello, al igual que en los apartados previos, no vamos a entrar en detalle en este punto, ya que se ha desarrollado en profundidad en el TFM de Pradia. Se puede observar lo mencionado, en dicho proyecto [1], en el capítulo 3, en la sección del diseño de la interfaz.

5.4. Creación de nuevos niveles

En este apartado, se va a explicar paso a paso como crear un nuevo nivel dentro del juego, gracias a la generalización que se ha conseguido en el mismo. De esta manera, el siguiente desarrollador que quiera avanzar con el código, podrá implementar nuevos escenarios con detalles y dinámicas concretos de una manera simple.

En primer lugar, a la hora de crear una nueva escena, sería una buena práctica almacenarla en la carpeta donde se encuentran el resto de las escenas (Scenes), para mantener una jerarquía en el proyecto. Además, dicha escena debe ser añadida a la configuración del proyecto, es decir, en los Build Settings del proyecto, para que al construirlo, se pueda mostrar e interactuar con ella. Una vez creada, solo dispondremos de una cámara, por lo que debemos empezar a crear nuestro nivel con los elementos reutilizables que aporta la plantilla o añadiendo ciertos elementos nuevos. Según se ha visto en la sección de Unity 2.4.1, en la parte izquierda de la interfaz se encuentra la ventana de jerarquía, donde debemos añadir los objetos y elementos que deseamos. Para ello, empezaremos eligiendo el fondo del nivel, por lo que se arrastrará el sprite a dicha sección y se ajustará a las medidas que deseemos para que quede en un estado óptimo, según se puede aprecia en la imagen 5.24.

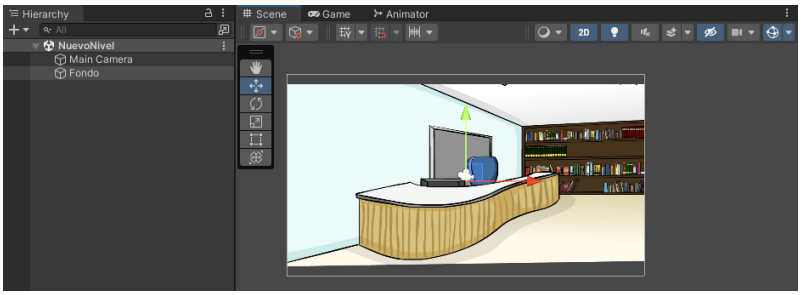

Figura 5.24: Paso 1 para crear nivel- Añadir fondo

Una vez elegido el fondo, se procede a completar la escena con los elementos principales de la misma. Todos ellos, al ser utilizados en distintas escenas, se encuentran en la carpeta de prefabs,

para poder reutilizarlos cuando se desee. Entre los elementos principales destaca el inventario, el cual se encuentra en todas las escenas. Bastaría con arrastrar el prefab ya creado sobre la interfaz de Unity y ajustarlo a nuestro gusto. También deberá aparecer el protagonista del videojuego, el cual está presente en todas las escenas. Por ello, debemos realizar el mismo procedimiento que con el inventario. Además, habrá que añadirle el script de Animaciones (encontrado en la carpeta Elementos), el cual se utiliza para cambiar de estado en las animaciones de todos los personajes. Por último, si queremos añadir nuevos elementos complementarios a la escena, los cuales no se encuentran entre los objetos prefabricados o los sprites disponibles, solo debemos realizar el mismo procedimiento que en los pasos anteriores, es decir, añadirlos a la escena y ajustar su tamaño. Cabe resaltar, que para conseguir una buena disposición espacial de los elementos y superponerlos a nuestra elección, se permite la opción de ajustar el nivel de los elementos por capas, para poder darles mayor o menos profundidad en la escena. Como vemos en la imagen 5.25, se puede conseguir crear una escena básica casi al instante. Además, al haber añadido objetos prefabricados, ya disponemos de sus animaciones y de sus funcionalidades otorgadas en los scripts que las componen. Obviamente, si queremos añadir nuevas animaciones o scripts se puede realizar según hemos explicado en capítulos anteriores.

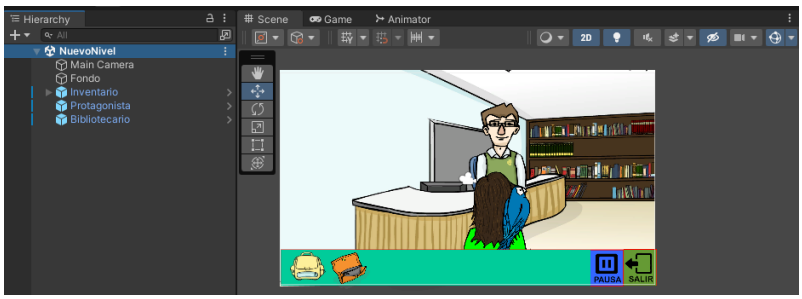

Figura 5.25: Paso 2 para crear nivel- Añadir prefabs y elementos necesarios

Una vez añadidos los elementos que componen la escena, se procede a realizar las actividades y dinámicas que la componen según nuestras necesidades. Para ello, se debe añadir un Game Object a la ventana de jerarquía de Unity, el cual gestione los eventos que suceden en la escena. Al añadir dicho objeto, se le asigna el script Interactable (situado en la carpeta Actions), el cual se encarga de gestionar los eventos que suceden de forma secuencial según se añaden. Como se ve en la imagen 5.26, se añaden los eventos que queremos que conformen nuestra escena por orden de aparición. De esta manera, se puede añadir, eliminar y modificar de forma sencilla la ejecución de nuestras tareas. Entre las diferentes acciones que podemos añadir, destacan principalmente las dinámicas, las cuales son clave para el desarrollo de los niveles. Así mismo, la interacción de los personajes para hablar también es importante en todos los niveles cuando recibimos explicaciones o diálogos. De esta manera, se pueden añadir los scripts ya creados, los cuales contienen todas las dinámicas y funciones claves para cada escena. En caso de que en una escena decidamos crear un nuevo comportamiento o acción, bastaría con diseñar el script si no existe y añadirlo al gestor de eventos.

Una vez visto como se añaden los scripts, vamos a comentar como modificar su funcionalidad dependiendo de la escena, es decir, poder añadirles diferentes audios, textos, dificultades o intentos entre otros. Para las funcionalidades ya predefinidas comentadas antes, no hace falta añadir ningún comportamiento nuevo, ya que están implementados. Solo habría que añadir los elementos que deseamos sobre el script, mediante la ventana de inspección (como vimos en la sección de Unity 2.4.1). En este

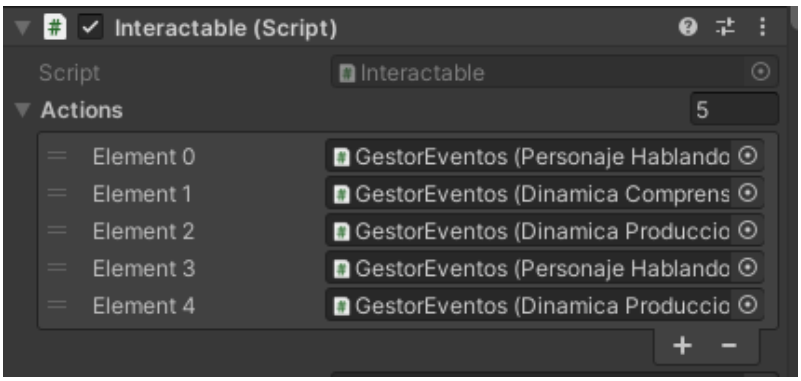

Figura 5.26: Paso 3 para crear nivel- Añadir gestor de eventos

punto, para la locución de un personaje por ejemplo, según vemos en la imagen 5.27, basta con añadir el audio que queremos reproducir, el personaje que queremos que reproduzca ese audio y el nombre de la variable que dispara la animación que queremos realizar. Si no sabemos que variable dispara cada estado de un personaje, basta con pulsar sobre el objeto que contiene el personaje, y en la ventana Animator de Unity podremos ver los diferentes estados de animaciones y las variables o trigger que los disparan.

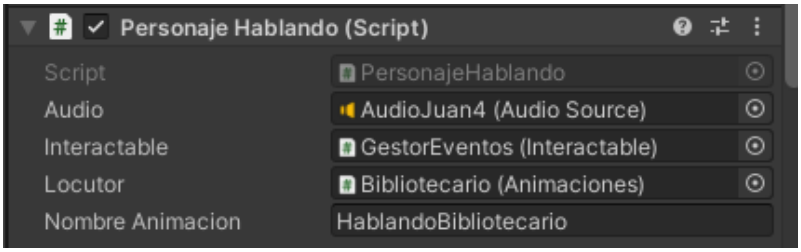

Figura 5.27: Paso 4 para crear nivel- Añadir elementos al script de hablar

Para el caso de las dinámicas tenemos dos ejemplos, aunque el procedimiento es similar para ambos. Para la dinámica de producción, como se ve en la imagen 5.28, basta con completar los campos que queramos desde la interfaz de Unity. Solo se necesita arrastrar los audios que necesitemos a los campos necesarios, como por ejemplo el audio de ayuda o error. También tenemos la opción de rellenar todos los campos disponibles, como el número de repeticiones, la dificultad o el texto a grabar. Para añadir el panel correspondiente a esta dinámica, basta con arrastrar el prefab correspondiente a la dinámica, el cual contiene dicho panel. En caso de que queramos un refuerzo positivo o negativo habría que realizar la misma operación, arrastrando dicho objeto prefabricado sobre el campo necesario.

En el caso de la dinámica de comprensión, como vemos en la imagen 5.29, habría que rellenar los campos al igual que para la anterior dinámica, completando así los audios o variables correspondientes. En este caso, no hay un solo panel de grabación, sino que existen dos. Para ello, debemos acceder a los prefabs correspondientes a dichos paneles y arrastrarlos sobre el campo necesario, igual que en la dinámica de producción.

Una vez hemos realizado los pasos comentados, ya disponemos de una escena funcional, con los elementos necesarios y las actividades y dinámicas añadidas correctamente. Como se ha explicado antes, puede haber elementos nuevos que se deseen implementar. Por lo que si existe algún sprite

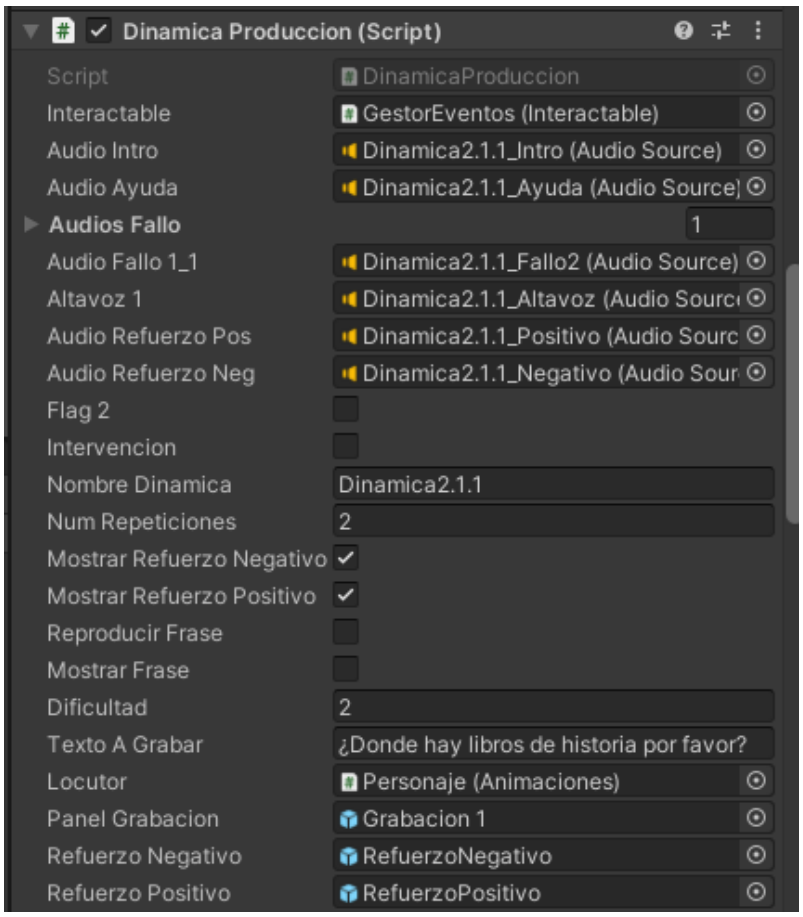

Figura 5.28: Paso 4 para crear nivel- Añadir elementos a la dinámica de producción

Figura 5.29: Paso 4 para crear nivel- Añadir elementos a la dinámica de comprensión

nuevo o alguna nueva acción o dinámica que surja, bastaría con crearla y añadirla como los anteriores elementos correspondientes a la plantilla.

Capítulo 6

Pruebas

En este apartado se van a comentar las pruebas realizadas para la comprobación de un uso correcto de la aplicación.

6.1. Test unitarios

Según lo comentado anteriormente, se van a realizar solamente test unitarios para comprobar el correcto funcionamiento del desarrollo, así como los resultados esperados para el mismo. Cabe remarcar, que un test unitario es una forma efectiva de comprobar el correcto funcionamiento de las unidades individuales más pequeñas de la aplicación. De esta manera, para realizar el código con las pruebas, se ha necesitado la integración en Unity de la biblioteca NUnit [64], la cual es una biblioteca de pruebas unitarias de código abierto para lenguajes .Net. En cuanto a las pruebas realizadas, cabe remarcar la posible diferencia a la hora de realizar los test, ya que existen 2 modos distintos para ello [65]:

- Edit Mode: También conocido como pruebas del editor, solo se ejecutan en el editor de Unity y tienen acceso al código del editor además del código del juego. En dichas pruebas, se verifica todo lo que no esté relacionado con el modo de reproducción, es decir, excluye las funciones Start, Update o Awake para centrarse en elementos que deban probarse explícitamente antes de ingresarlos. Además, este método no necesita inicializar todo antes de poder realizar las pruebas, por lo que resulta más rápido.
- Play Mode: Este tipo de pruebas permiten ejercitar el código de juego en tiempo de ejecución, ya que las pruebas se ejecutan como corrutinas si están marcadas con el atributo UnityTest, por lo que también necesitan definir un tiempo de retorno para su correcto uso. Se busca por tanto verificar los scripts que requieren estar en tiempo de ejecución, es decir, los relacionados con las funciones Start, Update o Awake. Además, es necesario realizar este tipo de test cuando se necesita instanciar elementos en tiempo de ejecución, como por ejemplo el escritor Json en nuestro caso.

Principalmente, hemos realizado nueve test unitarios, donde nos hemos centrado en verificar la funcionalidad referente a las dinámicas, para comprobar que todos los comportamientos se realizaban correctamente. Además, como veremos a continuación, también se han verificado los comportamientos clave de otras clases necesarias para poder realizar el flujo correcto del proyecto. Las distintas pruebas realizadas en ambos modos ya explicados, se pueden ver en el apéndice, en el apartado B, donde se encuentran todas las tablas con la descripción correspondiente de cada una, así como los resultados esperados y obtenidos.

Aunque los resultados obtenidos se encuentren en el apéndice, se van a mostrar unos extractos de código para visualizar mejor las pruebas realizadas. Como se ha comentado antes, se han definido dos ficheros distintos, uno para los test referentes al Edit Mode y otro para los referentes al Play Mode. En ambos casos, se ha necesitado crear los objetos necesarios para implementar varias de las pruebas. Para ello, se han cargado como Game Objects los Prefabs necesarios residentes en el proyecto, para poder utilizarlos, como se ve en la figura 6.1.

```
private GameObject panelGrabacion = AssetDatabase.LoadAssetAtPath<GameObject>("Assets/Prefabs/Produccion/Grabacion 1.prefab");
private GameObject opcion1 = AssetDatabase.LoadAssetAtPath<GameObject>("Assets/Prefabs/Comprension/Opcion1.prefab");
private GameObject opcion2 = AssetDatabase.LoadAssetAtPath<GameObject>("Assets/Prefabs/Comprension/Opcion2.prefab");
```

Figura 6.1: Objetos definidos en ambos ficheros de prueba

Por otra parte, podemos ver una de las pruebas realizadas en el Test Mode, la cual es referente a la prueba encontrada en la tabla B.8. En ella, se comprueba el funcionamiento de la dinámica de comprensión cuando se desactivan los elementos en medio de una dinámica o audio, corroborando así que los elementos clave están desactivados.

```
[Test]
0 referencias
public void EstadoDesactivado()
{
    DinamicaComprension dc = new DinamicaComprension();
    dc.Inicializar(opcion1, opcion2);
    dc.EstadoDesactivado();
    Assert.IsFalse(dc.getBoton1().gameObject.GetComponent<Boton>().getActivo());
    Assert.IsFalse(dc.getBoton2().gameObject.GetComponent<Boton>().getActivo());
}
```

Figura 6.2: Extracto prueba realizada en el fichero Edit Mode

Además de la anterior, podemos ver otra de las pruebas realizadas, esta vez en el fichero de Play Mode, la cual es referente a la prueba encontrada en la tabla B.3. En ella, se comprueba el funcionamiento de la dinámica de producción cuando se detiene una grabación. Para ello, se comprueba que el micrófono pase de estar activo a desactivado y tras un tiempo de espera nulo, se afirma que está desactivado y por tanto no está realizando ninguna grabación.

```
[UnityTest]
0 referencias
public IEnumerator DetenerGrabacion()
{
    Micro micro = new Micro();
    Diccionario d = new Diccionario();
    Escritor.Instance.save(d.getDiccionario("Entrada_Produccion"), "DinamicaTest");
    micro.setNombreDinamica("DinamicaTest");
    micro.UsandoMicro();

    micro.NoUsarMicro();
    yield return null;
    Assert.IsFalse(micro.getPulsado());
}
```

Figura 6.3: Extracto prueba realizada en el fichero Play Mode

Capítulo 7

Conclusiones

En este capítulo se van a abordar las conclusiones observadas una vez finalizado el proyecto, tomando como partida los objetivos que se plantearon al inicio. Además, se va a hablar también del trabajo futuro a realizar, como pueden ser posibles implementaciones y futuras mejoras del desarrollo.

El objetivo principal del presente TFG es migrar parte de las funcionalidades existentes en Pradia al motor gráfico de Unity, ya que Flash se había quedado anticuado. De esta manera y mediante el uso de estas herramientas, se ha conseguido la migración de forma exitosa, mediante el aprendizaje y uso de una plataforma en auge como es Unity y de un lenguaje de programación innovador y sencillo como lo es C#. De este modo, se ha conseguido cerrar exitosamente uno de los objetivos principales y evitar así uno de los grandes problemas con el que nos habíamos encontrado y que surge más a menudo de lo usual, el cual es el deterioro y olvido de herramientas con las que se ha desarrollado un proyecto. Por lo tanto, nos aseguramos varios años de desarrollo prometedor, ya que tanto la nueva plataforma como su comunidad y mantenimiento auguran muchos años de vida.

En cuanto a otro de los objetivos principales, como es la de realizar la migración de una forma generalista para poder ser reutilizada, podemos observar como también se ha cumplido. Dentro del propio juego, cuando hemos querido añadir nuevos niveles que contenían las dinámicas o actividades ya desarrolladas, hemos comprobado como la capacidad de reutilización ha reducido el tiempo de desarrollo de esas pantallas de un modo cuantioso. Esto, extrapolado a otros juegos educativos que incluyan tecnologías del habla, resultará igual de eficiente y prometedor, ya que va a ahorrar mucho tiempo a los desarrolladores. De esta manera, se ha buscado realizar un código limpio para su mantenimiento o utilización por otras personas ajenas al proyecto.

Otro punto importante, sobre todo para las personas a las que va dirigido, es el hecho de conseguir tanto una interfaz amigable como unas dinámicas que consigan enganchar al usuario. Para las personas con síndrome de Down, es muy frecuente perder la atención sobre una actividad si no resulta interesante o si gráficamente no es atractiva. Por este motivo se ha buscado asemejarse lo máximo a

la interfaz propuesta por Pradia, así como a las actividades implementadas en ese proyecto. Para ello, podremos afirmar que también ha sido un éxito. En nuestro caso, al haber seguido las pautas y guías de diseño del TFM original, podemos afirmar que al menos se obtendrán los mismos resultados que la versión pionera. Por lo que podemos afirmar que para dicho desarrollo, resultó todo un éxito para los usuarios, ya que permanecieron jugando sin apenas distracciones y encontraron las dinámicas interesantes en vez de aburridas o frustrantes. De este modo, debido a la semejanza que comparten ambos proyectos, podemos afirmar que también se ha cumplido con esta necesidad básica y tan importante mencionada.

También cabe destacar la importancia de las muestras de voz que se recojan durante todo el juego, en las actividades relacionadas con el habla y la grabación de frases. Dichas muestras permitirán realizar un análisis en profundidad de los problemas lingüísticos de esta población, así como comprobar si el videojuego les ha permitido mejorar en ciertos aspectos del lenguaje.

Otro punto importante que se ha concluido del trabajo es la dificultad de realizar una planificación correcta y precisa en cuanto a las horas exactas de trabajo. Pese a una planificación y unas iteraciones marcadas, siempre pueden surgir problemas. En nuestro caso, debido a la inclusión de un mes más de prácticas extracurriculares, las cuales no se contemplaban al inicio de TFG, han supuesto un cierto retraso en la planificación inicial hecha. Este hecho sumado a ciertos requisitos añadidos en iteraciones intermedias del proyecto, han propiciado un pequeño retraso en relación con la planificación inicial. De todas formas, gracias a la metodología que hemos utilizado, se ha podido mostrar el progreso a lo largo de las iteraciones e incluso dentro de ellas. De esta manera los tutores han podido corregir conceptos erróneos y dar nuevas ideas sin tener que tirar todo el trabajo realizado. El hecho de comprobar el progreso en las iteraciones con las fechas y tiempo planificado permite conocer si hay retrasos y cómo de grandes son.

En cuanto al apartado personal, el trabajo ha resultado muy interesante y enriquecedor, ya que me ha permitido recuperar un juego que se encontraba en una plataforma sin opción de desarrollo futuro. De esta forma, se ha conseguido dar pie a futuros desarrollos e implementaciones a partir de lo trabajado en este proyecto. En cuanto a la aportación que ha supuesto para una futura carrera profesional, cabe destacar la importancia de realizar un videojuego con uno de los motores gráficos más importantes del momento, lo cual sumado al aprendizaje de un nuevo lenguaje como es C#, supone una experiencia plena. En cuanto a las tecnologías utilizadas para ello, pese a que ya las conocía un poco debido al interés que me habían suscitado en años anteriores, he conseguido aprender enormemente el desarrollo de un videojuego. Tanto el manejo de Unity como el scripting y patrones de diseño software que dan sentido a todo. Todo ello, lo he aprendido de forma autodidacta, pero cabe destacar el uso continuo de Java en la universidad, que al ser similar a C#, ha facilitado su aprendizaje. Además, en las diferentes asignaturas cursadas, como Planificación y Gestión de Proyectos por ejemplo, he podido guiarme para realizar la planificación del proyecto, o Diseño de Software, dónde he aprendido a realizar todo tipo de diagramas de diseño implementados en dicho documento. Respecto del mantenimiento de aplicaciones, queda comprobado la necesidad de mantener actualizado tanto el código fuente de la

aplicación como la propia herramienta que se utiliza, así como su versión. Si esto no se hace, pueden aparecer una gran cantidad de problemas asociados debido a la eliminación o cambio de muchas bibliotecas. Además de ello, es necesario resaltar la necesidad de una arquitectura bien definida que permite diferenciar los módulos de la aplicación por capas y extraer la lógica de negocio de todas las vistas. Todas estas acciones aprendidas, han sido posibles en gran medida gracias a toda la documentación previa realizada, así como la comparación con otros proyectos ya realizados del mismo ámbito. Para ello, como se ha explicado anteriormente, ha sido esencial el uso de tutoriales relacionados con Unity y C#, así como la lectura de documentos explicativos sobre las herramientas utilizadas, y la comparativa la implementación de otros videojuegos de carácter educativo.

Por último, este proyecto puede ser el punto de partida para la realización de otro TFG, ya que se ha visto que el desarrollo de aplicaciones educativas y, más concretamente videojuegos orientados específicamente para la población con discapacidad intelectual, es un área aún por explorar.

7.1. Limitaciones y trabajo futuro

Como en todo proyecto, debido al tiempo disponible para su desarrollo, ha sido posible realizar los objetivos marcados, pero no ha sido posible completar todo el juego, ya que eso llevaría más meses de desarrollo. Por este motivo, se proponen una serie de tareas en el futuro las cuales buscan solucionar dicho problema y favorecer otros comportamientos dentro del desarrollo. Para auxiliar el trabajo futuro, cabe resaltar la importancia de las pautas explicadas sobre la creación de nuevos niveles de juego en la sección 5.4, ya que el hecho de implementar el juego en Unity de manera genérica favorece el desarrollo de nueva funcionalidad.

En primer lugar, según lo comentado, se busca poder realizar el juego al completo. De esta manera y habiendo dado pie al inicio del desarrollo, completarlo ahora será una tarea más sencilla. El juego original de Pradia, contaba con más niveles, pero los comportamientos y dinámicas fundamentales están realizados. De este modo, la realización de los niveles restantes no supondrá tantas horas de trabajo como si no se hubiera generalizado todo el trabajo anterior. Las únicas situaciones donde se contempla una dedicación de mayor tiempo son en aquellas pantallas no generalizadas anteriormente, como puede ser el menú principal, por ejemplo.

En cuanto a la implementación de las llamadas http para el reconocedor, se busca tanto la mejora del código implementado, como la introducción de varios reconocedores diferentes. De esta manera se busca cumplir todas las llamadas de forma correcta sin ningún inconveniente, así como facilitar el código y estructuración lo más limpia posible. Además, la segunda premisa referente a los múltiples reconocedores, facilitará el desarrollo en proyectos futuros para distintas acciones del juego, no solo reduciéndose al caso específico de la grabación.

En tercer lugar, uno de los puntos que se comentó pero no llegó a fijarse, ha sido el de implementar varios idiomas para el videojuego. Dicho objetivo busca la capacidad de poder implementar, además del castellano, los idiomas más hablados comúnmente, para poder expandir el desarrollo a otros países, facilitando así su uso. La implementación no es complicada, aunque en una primera instancia se buscar modificar el idioma de los textos mostrados únicamente. En cuanto a los audios reproducidos, conllevaría una idea para un desarrollo más lejano, ya que supondría grabar los mismos de nuevo. Esto se debe a que existe la posibilidad de transcribir el audio, modificar el idioma y reproducirlo, pero no nos asegura que la locución sea igual de interesante para los usuarios que la ya establecida, ya que sería realizada por un sintetizador automático de voz sin el mismo énfasis ni entonación.

Por otra parte, pese a que ya se han realizado los test de usabilidad en el desarrollo de Pradia y no se ha visto necesario repetirlos en dicho proyecto, se buscará realizar también dichos test. Este objetivo busca el hecho de confirmar las suposiciones que se habían obtenido en la conclusión, ya que pese a la similitud de los proyectos, puede haber diferencias significativas que no se hayan apreciado. De este modo, se busca realizar dichas pruebas en centros escolares o especiales, con la inclusión de personas con síndrome de Down, para poder verificar nuestros resultados y mejorarlos. De la misma manera, se busca colaborar con logopedas para comprobar que las herramientas que se les facilitan para corregir las locuciones son sencillas de utilizar y correctas.

Apéndices

Apéndice A

Acrónimos

- **2D**: Dos Dimensiones.
- **3D**: Tres Dimensiones.
- **API**: Application Programming Interface.
- **BD**: Base de Datos.
- **CLST**: Centre for Language and Speech Technology.
- **COVID-19**: Coronavirus-Disease-19.
- **ECA-SIMM**: Grupo de investigación de entornos de computación avanzada y sistemas de interacción multimodal.
- **GUI**: Graphical User Interface.
- **GB**: Gigabyte.
- **IVA**: Impuesto sobre el Valor Añadido.
- **JSON**: JavaScript Object Notation.
- **JWT**: Json Web Token.
- **SSD**: Solid-State Drive.
- **TFG**: Trabajo Fin de Grado.
- **TFM**: Trabajo Fin de Máster.
- **UVa**: Universidad de Valladolid.

Apéndice B

Resultados de los test unitarios

Caso Prueba 01	Inicialización correcta en la dinámica de producción
Descripción	Al crear una dinámica de producción, se inicializa el panel con los elementos necesarios para su funcionamiento
Entrada	Iniciar la dinámica de producción
Resultado esperado	El panel ya no es nulo y los elementos que lo forman están inicializados
Resultado	Correcto

Tabla B.1: Caso Prueba 01 (Inicialización correcta producción)

Caso Prueba 02	Iniciar una grabación
Descripción	Se quiere iniciar una grabación en la dinámica de producción
Entrada	Se pulsa el micrófono para grabar
Resultado esperado	Se crea un audio que guarda la grabación
Resultado	Correcto

Tabla B.2: Caso Prueba 02 (Iniciar grabación)

Caso Prueba 03	Detener una grabación
Descripción	Se quiere detener una grabación una vez evaluada la dinámica
Entrada	Se pulsa 1 o 2 por teclado para evaluar
Resultado esperado	El micrófono deja de estar pulsado para guardar la grabación
Resultado	Correcto

Tabla B.3: Caso Prueba 03 (Detener grabación)

Caso Prueba 04	Esperar la explicación
Descripción	Al iniciar la dinámica de producción no se puede intervenir hasta que termine la explicación
Entrada	Iniciar la dinámica de producción
Resultado esperado	La posibilidad de intervención está desactivada
Resultado	Correcto

Tabla B.4: Caso Prueba 04 (Esperar explicación)

Caso Prueba 05	Inicialización correcta en la dinámica de comprensión
Descripción	Al crear una dinámica de comprensión, se inicializan ambos paneles con los elementos necesarios para su funcionamiento
Entrada	Iniciar la dinámica de comprensión
Resultado esperado	Los paneles ya no son nulos y los elementos que los forman están inicializados
Resultado	Correcto

Tabla B.5: Caso Prueba 05 (Inicialización correcta comprensión)

Caso Prueba 06	Seleccionar la opción correcta en la dinámica de comprensión
Descripción	Se selecciona el botón que corresponde a la opción correcta en la dinámica
Entrada	Pulsado el botón correcto
Resultado esperado	Los paneles se desactivan
Resultado	Correcto

Tabla B.6: Caso Prueba 06 (Seleccionar opción correcta)

Caso Prueba 07	Seleccionar la opción incorrecta en la dinámica de comprensión
Descripción	Se selecciona el botón que corresponde a la opción incorrecta en la dinámica
Entrada	Pulsado el botón incorrecto
Resultado esperado	El número de intentos disponibles se decrementa
Resultado	Correcto

Tabla B.7: Caso Prueba 07 (Seleccionar opción incorrecta)

Caso Prueba 08	Estado desactivado en la dinámica
Descripción	Estado en el que todos los elementos se desactivan visual y funcionalmente mientras está ocurriendo otra acción que no les permite estar activos
Entrada	Mientras se está reproduciendo un audio
Resultado esperado	Elementos de los paneles desactivados
Resultado	Correcto

Tabla B.8: Caso Prueba 08 (Estado desactivado en la dinámica)

Caso Prueba 09	Estado activado en la dinámica
Descripción	Estado en el que todos los elementos se vuelven a activar visual y funcionalmente cuando ya no ocurre otra acción que no les permite estar activos
Entrada	Finalización de la reproducción de un audio
Resultado esperado	Elementos de los paneles activados
Resultado	Correcto

Tabla B.9: Caso Prueba 09 (Estado activado en la dinámica)

Apéndice C

Manual de despliegue

C.1. Despliegue para desarrollador

C.1.1. Instalación de git y clonación del repositorio

En primer lugar, si no disponemos de git en nuestro dispositivo, procedemos a instalarlo, para poder obtener la información que reside en el repositorio. Una vez instalado, basta con abrir una consola desde el directorio que deseemos, y ejecutar el comando:

- **git clone https://gitlab.inf.uva.es/gonzfer/tfg_pruebas_gonzalo.git**

De esta forma tenemos acceso a toda la estructura del proyecto para poder clonarlo en nuestra máquina local.

C.1.2. Despliegue del proyecto en Unity

A continuación vamos a comentar como poder desplegar nuestro proyecto:

- En primer lugar, si no desponemos de Unity Hub para gestionar los proyectos de Unity, deberemos descargar la version 3.1 en adelante. Según indica en la página oficial [66], basta con descargarse el ejecutable e instalarlo para poder disfrutar de la aplicación. Una vez instalado, deberemos crearnos una cuenta para poder gestionar nuestros proyectos. Para ello podemos introducir una licencia gratuita hasta de tres formas diferentes.
- Cuando ya dispongamos de UnityHub, bastaría con añadir una versión de Unity para poder ejecutar nuestros proyectos. De esta forma, dentro de UnityHub, en el apartado Installs, podremos añadir la versión que deseemos o necesitemos en cualquier momento.
- Ya instalada, nos dirigimos dentro de UnityHub a la sección Projects, donde podremos añadir nuestro proyecto, por lo que previamente hemos tenido que clonarlo del repositorio. Una vez

hecho y dentro de la sección Projects de UnityHub, seleccionamos el botón Open y buscamos la dirección donde tenemos almacenado el proyecto. Una vez seleccionado, ya podremos disponer del proyecto y el código necesario.

- Para ejecutar el proyecto, bastaría con abrirlo en Unity y pulsar el botón de Run que se encuentra en medio de la pantalla para previsualizarlo.
- Por otra parte, si deseamos generar el archivo ejecutable, debemos abrir el proyecto Unity y dirigirnos a la pestaña superior izquierda de Files, al apartado Build Settings, como se ve en la imagen C.1. Dentro, podemos seleccionar la plataforma a la que queremos exportar el proyecto, así como modificar las opciones de exportación. Una vez seleccionado el sistema operativo y la arquitectura del ordenador, procedemos a construirlo pulsando el botón Build en la parte inferior, como se ve en la imagen C.2. De esta manera, ya disponemos del archivo ejecutable que contiene nuestro juego.

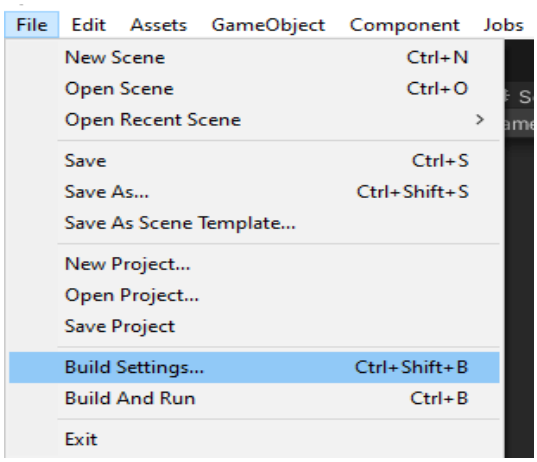

Figura C.1: Exportación del proyecto, parte 1

C.2. Despliegue para cliente

Para poder iniciar la aplicación, debemos tener un ordenador con un sistema operativo Windows, Mac o Linux. Una vez dentro, debemos clonar el proyecto del repositorio para acceder a la carpeta con los archivos necesarios para ejecutar el proyecto. Si no disponemos de git en nuestro dispositivo, procedemos a instalarlo, para poder obtener la información que reside en el repositorio. Una vez instalado, basta con abrir una consola desde el directorio que deseemos, y ejecutar el comando:

- **git clone https://gitlab.inf.uva.es/gonzfer/tfg_pruebas_gonzalo.git**

Una vez hecho, buscaremos la carpeta donde hayamos almacenado el proyecto en nuestro ordenador, y buscamos el fichero **TFG_Ejecutable**, el cual contiene todos los archivos necesarios para ejecutar directamente el juego. Para iniciar la aplicación, basta simplemente con hacer clic sobre el archivo ejecutable (como se ve en la imagen C.3) y ya podremos empezar el juego, sin necesidad de instalar ningún otro programa externo.

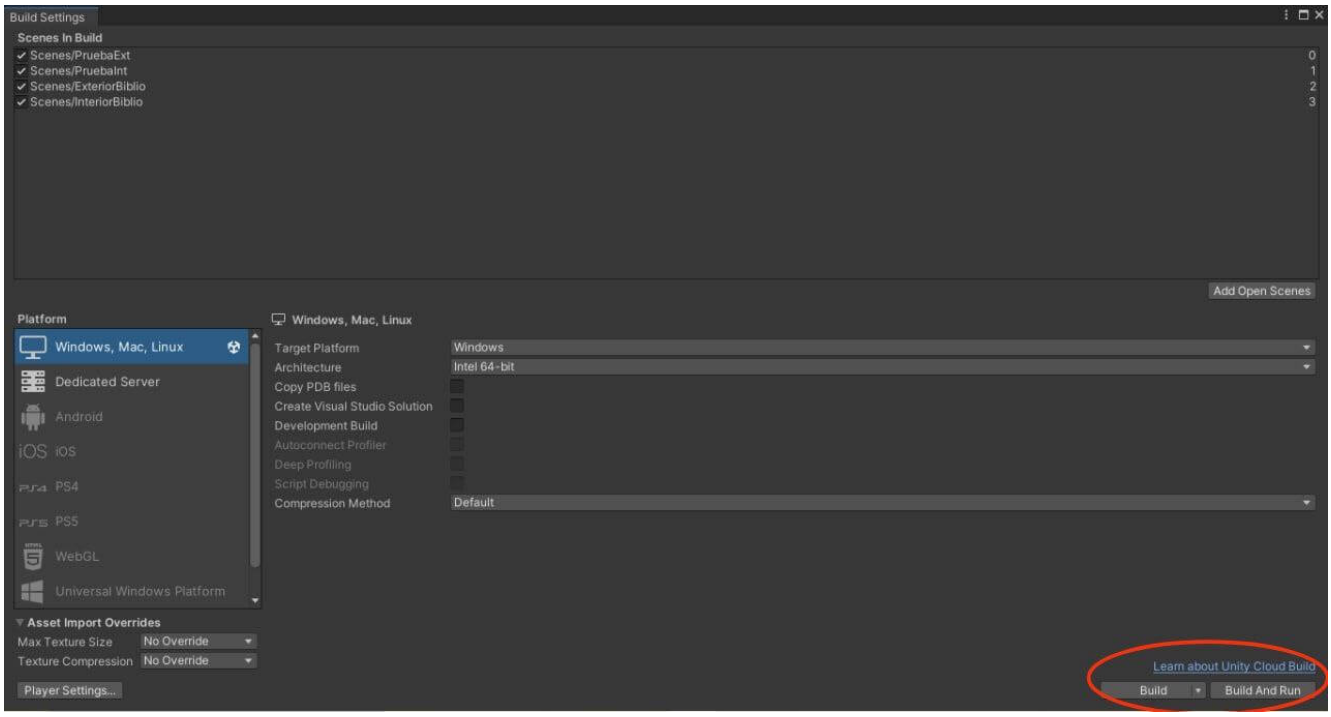

Figura C.2: Exportación del proyecto, parte 2

Figura C.3: Ejecutable del juego

Apéndice D

Manual de uso

En primer lugar, cuando se inicia la aplicación se inicia directamente el juego, por lo que no hace falta seleccionar ningún nivel ni configuración en un menú previo. De esta manera, como se puede observar en la figura 2.1, se accede directamente en la primera escena, es decir, el exterior de la librería.

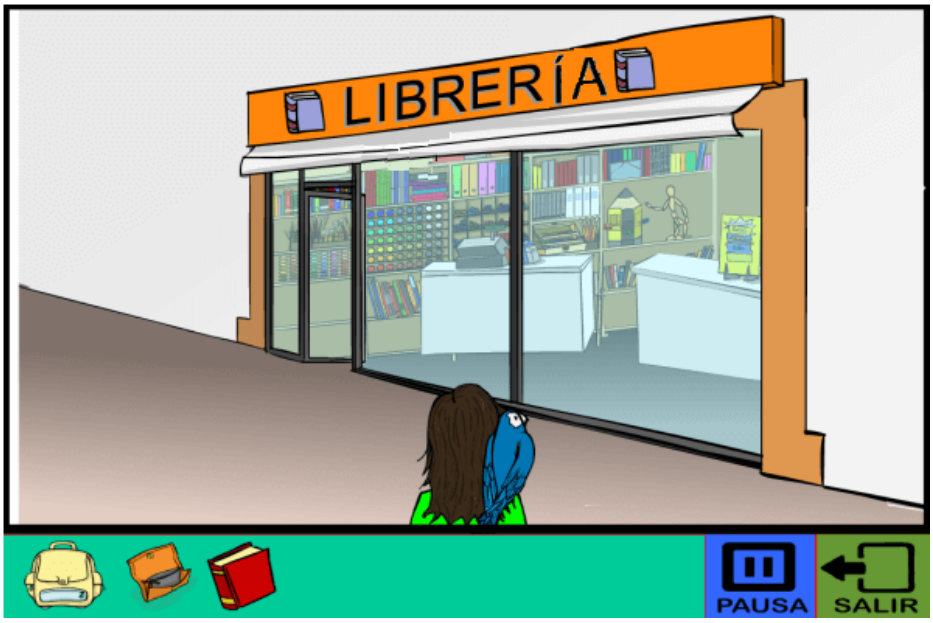

Figura D.1: Interfaz de la primera escena del juego

Como se puede observar, disponemos del inventario con los elementos iniciales del juego, aunque si intentamos interactuar con alguno de ellos no disponen de utilidad alguna en este nivel. Podemos observar que si seleccionamos el botón de pausa de la imagen D.1, obtendremos un menú de pausa, como se ve en la figura D.2, donde se detendrá tanto el juego como la música. Para salir de la pausa basta con volver a pulsar el propio botón. Por otra parte, al igual que en el resto de las escenas, en las cuales dispondremos también del panel inferior con el inventario y ambos botones, si pulsamos el botón de salir, conseguiremos terminar el juego.

Para este nivel, podemos observar que se puede cambiar de escena mediante la interacción con

Figura D.2: Menú de pausa

el objeto determinado. En este caso, es la puerta de la librería, la cual está remarcada como el resto de escenas donde se tiene esta disposición para cambiar de escenas. Por tanto, si seleccionamos la puerta, se produce la animación hasta el siguiente nivel.

Para la segunda escena, al entrar empezará la interacción con el dependiente, para seguidamente iniciar la dinámica de producción como se aprecia en la figura D.3.

Figura D.3: Interfaz de la segunda escena del juego

Como se observa, si seleccionamos el icono del micrófono, podremos empezar la grabación cuando estemos listos, observando en la animación como cambia el estado del micrófono para otorgar dicha sensación. Cuando el logopeda crea conveniente evaluar, selecciona la entrada de teclado correspondiente. En nuestro caso, si se selecciona la tecla 1, el logopeda habrá considerado la opción correcta y por tanto se obtendrá un refuerzo positivo como se aprecia en la figura D.4.

De la otra forma, si el logopeda considera la grabación como incorrecta, debe seleccionar la tecla 2.

Figura D.4: Refuerzo positivo en la dinámica de producción

Si se disponen de más intentos, se muestra el audio de error y aparece el panel anterior con un altavoz, para volver a escuchar la frase que se debe grabar, como se ve en la figura D.5. Si ya no se disponen

Figura D.5: Interfaz de error en la dinámica de producción

de intentos, se acompaña el audio de error con un refuerzo negativo, como se aprecia en la figura D.6.

Figura D.6: Refuerzo negativo en la dinámica de producción

Una vez superada la primera dinámica, se produce otra interacción con el dependiente, para posteriormente dar paso a la segunda dinámica de producción. En este caso, podemos observar el panel con el texto que se debe grabar inscrito, así como un altavoz para volver a escuchar la frase a grabar y el

propio micrófono para poder grabar. Todo ello se aprecia en la figura D.7. El procedimiento para iniciar

Figura D.7: Segunda dinámica de producción de la escena

la grabación es el mismo que se ha explicado antes, al igual que el de evaluación. En caso de que la grabación sea incorrecta, se producen los mismos pasos que para la otra dinámica de producción. Si todavía quedan intentos aparecerá un audio de error para después poder volver a intentarlo. En caso de no quedar intentos y existir el refuerzo negativo, se mostraría igual que en la figura D.6. En caso de haber realizado bien la grabación y existir un refuerzo positivo, se mostraría como en la figura D.4.

Una vez superada la dinámica, aparecerá una lupa en el escenario como se ve en la figura D.8. Para interactuar con ella y obtenerla, se necesita pagar con la cartera, la cual está resaltada. Una vez se paga, se añade la lupa al inventario y se procede a avanzar al siguiente nivel.

Figura D.8: Escena 2 obtención de la lupa

En el tercer nivel, como podemos observar en la figura D.9, tenemos los mismos elementos que en la escena inicial, al igual que sus mismas funcionalidades. Por tanto, para cambiar de escena deberemos pulsar el objeto resaltado, en este caso la puerta para entrar a la biblioteca. Por lo que comenzará la animación y se pasará a la escena 4.

Una vez dentro del cuarto nivel, podemos observar que se comenzará con la dinámica de comprensión, la cual muestra los paneles y la explicación previa de todos sus elementos, como se observa en la figura D.10.

En este momento podemos seleccionar tanto el altavoz para volver a reproducir cualquiera de las

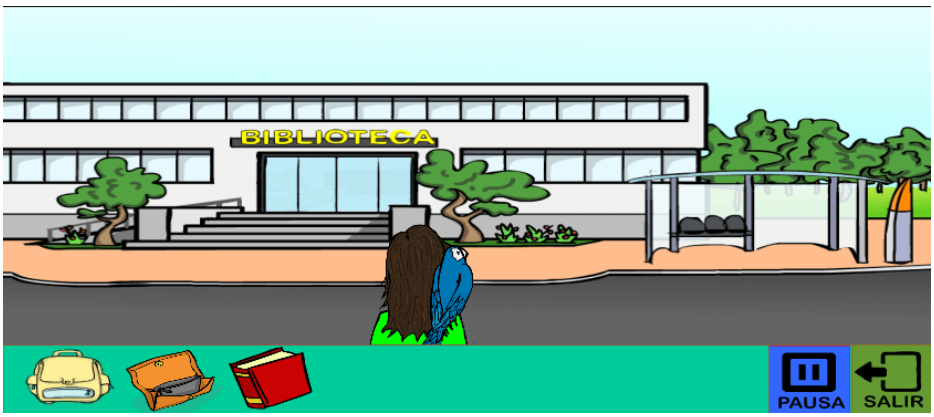

Figura D.9: Escena 2 obtención de la lupa

Figura D.10: Escena 4, dinámica de comprensión

frases, como seleccionar uno de los botones resaltados para elegir una opción. Si se comete un error se mostrará el audio de fallo. Si ya no se disponen de más intentos se mostrarán las opciones incorrectas como desactivadas, para poder elegir solo la correcta, como se ve en la imagen D.11. Si por el contrario

Figura D.11: Máximos intentos en la dinámica de comprensión

se acierta la dinámica, se mostrará el audio de acierto y si hubiera un refuerzo positivo lo acompañaría. Posteriormente, se suceden dos dinámicas más de producción, las cuales tienen el mismo funcionamiento que las ya explicadas anteriormente. Una vez superadas, se muestra un audio y se cambia de escena. Con lo que de esta forma quedarían vistas las diferentes escenas y la forma de interactuar con

ellas.

Apéndice E

Contenido del TFG

Junto con la memoria del TFG, cuyo nombre es MemoriaTFGGonzaloFernandezGonzalez.pdf encontramos otro archivo PDF (DatosTFGGonzaloFernandezGonzalez.pdf) que contiene una URL a un repositorio del GitLab de la Escuela de Ingeniería Informática de la Universidad de Valladolid. Dentro de este repositorio podemos encontrar:

- Fichero con el contenido y estructura del proyecto *TFG_Pruebas_Gonzalo*
- Fichero con el ejecutable de la aplicación lista para utilizar: *TFG_Ejecutable*

Bibliografía

[1] M. Corrales-Astorgano, “Videojuego para la mejora de la prosodia en personas con discapacidad intelectual,” 2015, Universidad de Valladolid. [Online]. Available: https://uvadoc.uva.es/handle/10324/15244

[2] Fin del soporte de Adobe Flash Player. Visitado: 2022-30-06. [Online]. Available: https://docs.microsoft.com/es-es/lifecycle/announcements/update-adobe-flash-support

[3] ¿Qué es Unity y para qué puedo utilizarlo? Visitado: 2022-14-06. [Online]. Available: https://www.ifp.es/blog/que-es-unity-y-para-que-puedo-utilizarlo

[4] ¿Qué es y para qué sirve la herramienta Adobe Flash. Visitado: 2022-14-06. [Online]. Available: https://es.ryte.com/wiki/Flash

[5] Adobe corrige vulnerabilidades críticas en Flash. Visitado: 2022-30-06. [Online]. Available: https://www.profesionalreview.com/2017/03/15/adobe-corrige-vulnerabilidades-criticas-flash-player/

[6] D. Escudero-Mancebo, M. Corrales-Astorgano, V. Cardeñoso-Payo, and C. González-Ferreras, “Evaluating the impact of an autonomous playing mode in a learning game to train oral skills of users with down syndrome,” *IEEE Access*, vol. 9, pp. 93 480–93 496, 2021.

[7] C. González-Ferreras, D. Escudero-Mancebo, M. Corrales-Astorgano, L. Aguilar-Cuevas, and V. Flores-Lucas, “Engaging adolescents with down syndrome in an educational video game,” *International Journal of Human–Computer Interaction*, vol. 33, no. 9, pp. 693–712, 2017.

[8] Desarrollo iterativo e incremental. Visitado: 2022-06-06. [Online]. Available: https://proyectosagiles.org/desarrollo-iterativo-incremental

[9] Esquema desarrollo iteraivo e incremental. Visitado: 2022-06-06. [Online]. Available: https://slideplayer.es/slide/3917246/

[10] Software de animación en 2d y Flash. Visitado: 2022-28-06. [Online]. Available: https://www.adobe.com/es/products/animate.html

[11] ¿Qué es Adobe Air? Visitado: 2022-28-06. [Online]. Available: https://www.nobbot.com/tecnologia/aplicaciones-de-escritorio/%C2%BFque-es-adobe-air/

[12] Finalización del soporte técnico de flash. Visitado: 2022-28-06. [Online]. Available: https://docs.microsoft.com/es-es/lifecycle/announcements/adobe-flash-end-of-support

[13] Navegadores compatibles- unity. Visitado: 2022-30-06. [Online]. Available: https://unity.com/es/supported-browsers

[14] Unity-manual: Sprites. Visitado: 2022-30-06. [Online]. Available: https://docs.unity3d.com/es/530/Manual/Sprites.html#:~:text=Los%20Sprites%20son%20objetos%20gr%C3%A1ficos,y%20conveniencia%20durante%20el%20desarrollo.

[15] Unity is a new 3d game engine. Visitado: 2022-14-06. [Online]. Available: https://logos-world.net/unity-is-a-new-3d-game-engine-with-a-new-identity/

[16] Unity User Manual 2021.1. Visitado: 2022-14-06. [Online]. Available: https://docs.unity3d.com/es/2021.1/Manual/UnityManual.html

[17] Unity-scripting API. Visitado: 2022-14-06. [Online]. Available: https://docs.unity3d.com/ScriptReference/index.html

[18] ¿Qué es unity hub? Visitado: 2022-15-06. [Online]. Available: https://unity.com/es/unity-hub

[19] ¿Qué es unity hub? Visitado: 2022-15-06. [Online]. Available: https://docs.unity3d.com/Manual/system-requirements.html

[20] Comenzar con github desktop. Visitado: 2022-15-06. [Online]. Available: https://docs.github.com/es/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop

[21] ¿Sabes qué es C#? Visitado: 2022-15-06. [Online]. Available: https://www.tokioschool.com/noticias/c-que-es/

[22] S. M. Cano, “Desarrollo de videojuegos en unity para educación,” 2019, Universidad Autónoma de Madrid. [Online]. Available: https://repositorio.uam.es/handle/10486/688946

[23] G. C. Gosp, “Desarrollo de un videojuego en unity para el aprendizaje de matemáticas de tercero y cuarto curso de educación primaria,” 2021, Universidad Politécnica de Valencia. [Online]. Available: https://riunet.upv.es/bitstream/handle/10251/173974/Caravaca%20-%20Desarrollo%20de%20un%20videojuego%20en%20Unity%20para%20el%20aprendizaje%20de%20matematicas%20de%20tercero%20y%20c....pdf?sequence=1

[24] G. T. Leiva, “Cirquizz: desarrollo de un videojuego para ayudar en el aprendizaje de análisis de circuitos usando unity,” 2018, Universidad Politécnica de Madrid. [Online]. Available: https://oa.upm.es/56609/1/TFG_GABRIELA_TORRES_LEIVA.pdf

[25] M. N. Seño, “Creando un videojuego educativo: Recycle,” 2015, Universitat Oberta de Catalunya. [Online]. Available: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/40576/6/mnavarro17TFC0115memoria.pdf

[26] Astah, Premier Diagramming, Modeling Software and tools. Visitado: 2022-27-06. [Online]. Available: https://astah.net/

[27] What is Google chrome. Visitado: 2022-27-06. [Online]. Available: https://www.computerhope.com/jargon/c/chrome.htm

[28] Excel-software de hojas de cálculo. Visitado: 2022-27-06. [Online]. Available: https://www.microsoft.com/es-es/microsoft-365/excel

[29] Git. Visitado: 2022-28-06. [Online]. Available: https://git-scm.com/

[30] The one devops platform ,gitlab. Visitado: 2022-25-06. [Online]. Available: https://about.gitlab.com/

[31] Google Drive, plataforma de almacenamiento personal en la nube. Visitado: 2022-23-06. [Online]. Available: https://www.google.com/intl/es_es/drive/

[32] Jitsi meet. Visitado: 2022-22-06. [Online]. Available: https://meet.jit.si/

[33] Microsoft OneDrive, Almacenamiento personal en la nube. Visitado: 2022-22-06. [Online]. Available: https://www.microsoft.com/es-es/microsoft-365/onedrive/online-cloud-storage

[34] Qué es Overleaf. Visitado: 2022-21-06. [Online]. Available: https://www.adrinerja.com/que-es-overleaf/

[35] What is Telegram? Visitado: 2022-21-06. [Online]. Available: https://www.businessinsider.com/what-is-telegram

[36] What is new in Unity 2021.2.9. Visitado: 2022-21-06. [Online]. Available: https://unity3d.com/es/unity/whats-new/2021.2.9

[37] Visual studio, el ide de microsoft. Visitado: 2022-15-06. [Online]. Available: https://conectasoftware.com/apps/visual-studio/

[38] Salario de un programador en España. Visitado: 2022-18-06. [Online]. Available: https://www.hackaboss.com/blog/salario-programador-espana

[39] Horas de trabajo anuales. Visitado: 2022-18-06. [Online]. Available: https://asesorias.com/empresas/normativas/laboral/jornada/horas-trabajo-anuales/

[40] ¿Cuántos kwh consume una casa? Visitado: 2022-19-06. [Online]. Available: https://tarifaluzhora.es/info/calcular-consumo-electrico-casa

[41] Cursos udemy. Visitado: 2022-20-06. [Online]. Available: https://www.udemy.com/

[42] Tutorial completo unity2d desde cero. Visitado: 2022-01-02. [Online]. Available: https://www.youtube.com/watch?v=GbmRt0wydQU

[43] Unity learn, guía para el kit de juego en 2d. Visitado: 2022-02-03. [Online]. Available: https://learn.unity.com/tutorial/2d-game-kit-walkthrough-1?language=es&projectId=5d827673edbc2a002039aa21#5d829e29edbc2a02ce6a1986

[44] Trigger. Visitado: 2022-21-06. [Online]. Available: https://www.glosarioit.com/Trigger

[45] Unity- scripting api serializable. Visitado: 2022-21-06. [Online]. Available: https://docs.unity3d.com/ScriptReference/Serializable.html

[46] Unity- manual: Corrutinas. Visitado: 2022-21-06. [Online]. Available: https://docs.unity3d.com/es/530/Manual/Coroutines.html

[47] Unity manual: Prefabs. Visitado: 2022-22-06. [Online]. Available: https://docs.unity3d.com/es/530/Manual/Prefabs.html

[48] Unity manual: Spawning de objetos. Visitado: 2022-22-06. [Online]. Available: https://docs.unity3d.com/es/530/Manual/UNetSpawning.html

[49] Unity scripting api: Keywordrecognizer. Visitado: 2022-22-06. [Online]. Available: https://docs.unity3d.com/ScriptReference/Windows.Speech.KeywordRecognizer.html

[50] C# dictionaries- unity forum. Visitado: 2022-22-06. [Online]. Available: https://forum.unity.com/threads/c-dictionaries.420371/

[51] About Unity test framework. Visitado: 2022-22-06. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html

[52] Language and speech tools. Visitado: 2022-10-07. [Online]. Available: https://webservices.cls.ru.nl/

[53] Programación por capas. Visitado: 2022-07-07. [Online]. Available: https://www.virtuniversidad.com/greenstone/collect/informatica/archives/HASH0195.dir/doc.pdf

[54] Patrones de diseño. Visitado: 2022-01-07. [Online]. Available: https://refactoring.guru/es/design-patterns

[55] Patrón experto, solución. Visitado: 2022-11-07. [Online]. Available: https://docplayer.es/1662489-Patrones-experto-solucion.html

[56] Unity3d patrón singleton. Visitado: 2022-01-07. [Online]. Available: https://mrcalderon3d.wordpress.com/2013/12/13/unity3d-patrn-singleton/

[57] Patrones de diseño-observer. Visitado: 2022-01-07. [Online]. Available: https://thepowerups-learning.com/patrones-de-diseno-observer/

[58] Patrón observador. Visitado: 2022-07-07. [Online]. Available: http://migranitodejava.blogspot.com/2011/06/observer.html

[59] GRASP: Polimorfismo. Visitado: 2022-11-07. [Online]. Available: https://juan-garcia-carmona.blogspot.com/2012/09/grasp-polimorfismo.html

[60] Patrones estructurales: Patrón fachada. Visitado: 2022-11-07. [Online]. Available: https://danielggarcia.wordpress.com/2014/03/03/patrones-estructurales-ii-patron-facade/

[61] Programming-patterns. Visitado: 2022-05-07. [Online]. Available: https://github.com/Habrador/Unity-Programming-Patterns#4-prototype

[62] State. Visitado: 2022-11-07. [Online]. Available: https://refactoring.guru/es/design-patterns/state

[63] Update method, sequencing patterns. Visitado: 2022-11-07. [Online]. Available: https://gameprogrammingpatterns.com/update-method.html

[64] About unity test framework. Visitado: 2022-08-07. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html

[65] Edit mode vs play mode tests. Visitado: 2022-08-07. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/edit-mode-vs-play-mode-tests.html

[66] Download Unity. Visitado: 2022-08-07. [Online]. Available: https://unity3d.com/es/get-unity/download